Aut-75.200 Kuvatekniikan erikoistyöt
WWW-sivujen aika-arvioija

Markus Peuhkuri
37681E TiN

12. toukokuuta 1997

Käytetyt lyhenteet ja termit

ActiveX Microsoftin[1] kehittämä menetelmä verkko-ohjelmien toteuttamiseksi.

ANSI American National Standards Institute. Standardisoimisjärjestö.

Apache Vapaasti levitettävä HTTP-palvelinohjelmisto.[2]

ARPA Advanced Research Projects Agency. Yhdysvaltain puolustusministeriön alainen tutkimuslaitos.

ARPANET Advanced Research Projects Agency Network. ARPA:n 1969 perustama verkko, josta kehittyi Internet.

B_w Kaistanleveys. [kB/s]

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

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

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

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

ICMP kts. Internet Control Message Protocol.

IETF Internet Engineerin Task Force. Vastaa Internetin teknologian kehityksestä ja standardisoinnista.[3]

IGMP kts. Internet Group Management Protocol.

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

---

[1] http://www.microsoft.com
[2] http://www.apache.org
[3] http://www.ietf.org

IP kts. Internet Protocol.

ISO International Standards Organization. Kansainvälinen standardoimisorganisaatio.[4]

Java Sun Microsystemssin[5] kehittämä kieli, joka mahdollistaa tavukoodin suorittamisen arkkitehtuurista riippumatta. Alunperin kehitetty TV:tä ja muita sulaitettuja laitteita varten, mutta saanut suuren suosion WWW:ssä hyvien turvallisuusominaisuuksien ansiosta.

JavaScript Netscapen[6] luoma kieli, jolla HTML-dokumentteihin voidaan listätä toiminnallisuutta. Muistuttaa jonkin verran Javaa.

K_d Dokumentin koko, tavuissa.

K_s Segmentin koko, tavuissa.

kB/s Siirtonopeuden yksikkö. Yleensä 1024 tavua sekunnissa eli 8 196 bittiä sekunnissa mutta voi joskus olla 8000 bit/s.

libwww-perl Aliohjelmapaketti, joka helpottaa erilaisten WWW-sovellusten ja -palvelinten ohjelmointia perl-kielellä.

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

Linux Suomalaisen Linus Torvaldsin alunperin kehittämä UNIX:in kaltainen käyttöjärjestelmä.[7]

LWP kts. libwww-perl.

NIC Network Information Center. Sisälsi ARPANET:n tietokannat kuten tiedot koneiden nimistä.

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

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

P_{loss} Solujen tai pakettien katoamistodennäköisyys.

S_w Siirtoikkunan koko.

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].

[4] http://www.iso.ch/
[5] http://www.sun.com
[6] http://home.netscape.com
[7] http://www.linux.org/

Silicon Graphics Eräs tietokonevalmistaja.[8]

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

SNMP Simple Network Management Protocol. Verkonhallintaprotokolla [SFDC90].

t_p Lyhyen ICMP-paketin edestakainen kulkuaika.

t_s Suuren ICMP-paketin edestakainen kulkuaika.

TCP kts. Transmission Control Protocol.

Telnet Virtuaalipääteprotokolla verkon yli [PR83b, PR83a].

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

UDP kts. User Datagram Protocol.

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

URI Uniform Resource Idenifier. Yleinimitys kaikille lyhyistä merkkijonoista muodostuville resursseille, jotka osoittavat resursseihin.

URL Universal Resource Locator. Määrittelee dokumentin verkko-osoitteen, hakumekanismin ja sijainnin [BLMM94]. kts. URI.

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

WAIS Wide Area Information Server. ANSI:n standardi Z39.50 tekstihakuihin.

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

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

[8] http://www.sgi.com/

Sisältö

Luku 1

Johdanto

Internet-huumassa jokaisen yrityksen, yhteisön tai henkilön on saatava omat WWW-sivunsa. Näiden sivujan laatijaksi tarjoituu jos jonkinlaista tekijää, niinpä tuloksetkin ovat usein kyseenalaisia. Mahdollisesti sivut näyttävät hyvältä ainoastaan tietyn selaimen tietyllä versiolla, tietyllä näytön koolla ja värimäärällä. Perustavaa laatua olevat käytettävyystekijät unohdetaan ja tuloksena on helposti sivu, jolla navigointi on vaikeaa [Nie96, Nie93]

Toinen käytettävyyteen vaikuttava seikka on sivun latautumisaika. Nopealla tietokoneella samassa lähiverkossa, lähes mikä tahansa sivu latautuu nopeasti. Useimmat sivujen suunnittelijat osaavat nykyään kiinnittää huomiota sivuilla käytettyjen kuvien kokoon. Latautumisajoissa voi kuitenkin olla modeemiyhteydellä tai toiselta mantereelta suuria eroja, vaikka sivujen tavumäärät itsessään olisivat samoja.

Tässä työssä perehdytään ensin tiedonsiirron toimintaan kuvaamalla Internetin yleistä rakennetta ja protokollien toimintaa. Tämän jälkeen kehitetään työkalu, jolla sivujen latausaikaa voidaan arvioida erilaislla yhteyksillä. Lopuksi esitellään saatuja tuloksia ja arvoidaan työkalun käyttömahdollisuuksia.

Luku 2

Tiedonsiirto Internet-verkossa

Internet-verkkoa ei voida täysin yksikäsitteisesti ja kiistattomasti määritellä, tarkottaahan nimi ”verkkojen välistä”. Samoin ”internet protokollat” on myöskin samalla tavalla epämääräinen käsite: internet-protokollia on määritellyt mm. ISO. Nykyään internet-protokollilla kuitenkin tarkoitetaan alunperin Yhdysvaltain puolustusministeriön ARPA-projektissa 70- ja 80-luvuilla kehitettyä IP-protokollaa ja siihen liittyviä ylemmän tason protokollia. Nykyään näiden protokollien kehittämisestä ja standardisoinnista vastaan Internet Engineering Task Force (IETF). Eräänä Internetin määritelmänä voidaan pitää ”suurin IETF:n määrittelemää IP-protokollaa käyttävävien verkkojen yhteenliittymä”.

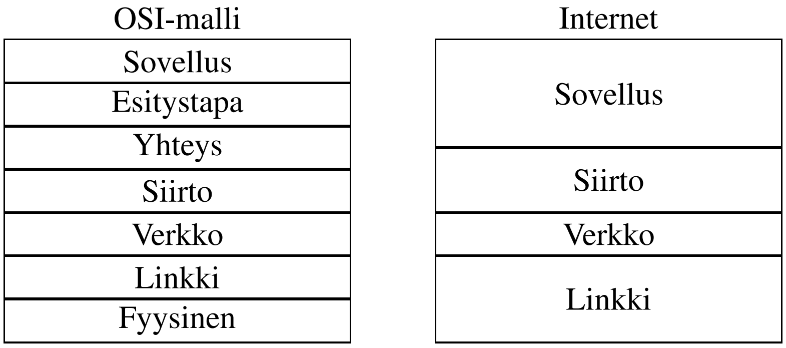
2.1 Tietoliikenteen kerrosrakenne

Tietoliikenteessä on käytetty ISO:n standardisoinnin tueksi määrittelemää 7-kerroksista OSI-viitemallia, jossa eri tehtävät on jaoteltu eri kerroksille. Näin voidaan määrittää eri standardien keskinäinen asema ja mahdollistaa eri osajärjestelmien vaihtaminen ilman, että muihin osajärjestelmiin jouduttaisiin puuttumaan. OSI-malli sopii hyvin tietoliikenteen havainnollistamiseen vaikka siitä ollaan standardiointipuolella luopumassa koska se johtaa helposti hyvin kankeisiin ja monimutkaisiin järjestelmiin: kaupallisella puolella se on vielä merkittävästi käytössä erilaisissa automaatio- ja säätöjärjestelmissä sekä perinteisissä televerkoissa.

Internet- ja OSI-malli eroavat toisistaan kerrosjaoltaan, internet-mallissa on neljä kerrosta ja samannimiset kerrokset ovat siten eri laajuisia. Mallien kerrosrakennetta on vertailtu kuvassa 2.1.

2.1.1 Kerrosten tehtävät

Internet-mallin kerroksien tehtävät ovat:



Kuva 2.1: OSI- ja Internet-mallien kerrosten vertailua [Bra89b].

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 osaan:

1. Suoraan käyttäjille tarjottavat palvelut, esimerkiksi pääteyhteydet Telnet [PR83b], sähköposti SMTP [Pos82, Cro82], 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 eroittamista eli tietueita: yhteys muodostuu kahdeksan bittiä pitkistä okteteista. UDP sensijaan 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].

linkkikerros Käytettävälle siirtotielle ja -menetelmälle tulee määritellä kuinka IP-sähkeet kuljetetaan kyseisessä mediassa. Määritelyjä on lukuisia alkaen äänteistä ja kirjekyyhkyistä [Eri96, Wai90].

Tämän työn kannalta mielenkiintoisia osia ovat siirtokerrokselta TCP ja palveluista HTTP sekä DNS. Reaaliaikainen multimedialiikenne käyttää yleensä UDP-protokollaa, mutta se on tämän työn ulkopuolella.

2.2 TCP:n toiminta

Useimmiten tiedonsiirron tarkoituksena on saada tieto menemään perille sellaisenaa, ilman että siitä katoaa tai siihen tulee mitään. Verkon kannalta on tärkeää, että sovellus sopeutuu kullonkin käytössä olevaan siirtokapasiteettiin, joka voi vaihdella huomattavastikin. TCP tarjoaa sovelluksille tälläisen palvelun mutta ei aivan ilmaiseksi: tyypillisessä toteutuksessa TCP:n toiminnan toteuttaminen vie 4500 riviä C-koodia, vertailukohtana UDP vaatii vain 700 riviä.

2.2.1 Yhteyden muodostus

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

2.2.1.1 Palvelinohjelma

Palvelinohjelma luo ensin liikennöintipisteen[1], joka sidotaan tiettyyn TCP-porttinumeroon, joka on varattu kyseiselle sovellukselle joko globaalisti [RP94] tai paikallisesti, HTTP-protokollan tapauksessa porttinumero on 80.

UNIX-järjestelmissä porttinumeroltaan alle 1 024:n olevan liikennöintipisteen avaaminen vaatii pääkäyttäjän oikeuksia. Tähän perustuen sovellukset on jaettu kahteen luokkaan: pääkäyttäjän oikeuksia vaativiin, jotka käyttävät palvelinportteina alle 1 024 olevia, sekä muihin, jotka käyttävät porttinumeroita 5 000 tai yli. Välissä olevat porttinumerot ovat asiakasohjelmien käytettävissä, yleensä asiakasohjelma varaa pienimmällä numerolla olevan vapaan portin. Tämä on vain yleinen käytäntö; mikään ei yleensä estä käyttäjän omaa sovellusta avaamasta liikennöintipistettä porttinumerolla 5 000 tai yli samoin kuin yhdenkäyttäjän koneessa porttinumerolla alle 1 024.

Palvelinprosessi ilmoittaa tämän jälkeen olevansa valmis ottamaan yhteyksiä vastaan. Liikennöintipisteeseen liittyy useimmissa järjestelmissä jono: mikäli kutsuja tulee enemmän kuin ehditään hyväksyä, jonon täytyttyä yhteyspyynnöt hylätään suoraan. Useimmissa käyttöjärjestelmissä palvelinohjelma voi määrittää jonon pituuden, kunhan se on alle maksimiarvon: joissakin järjestelmissä maksimi on vain 5, joissakin jopa 128 tai suurempi.

[1] socket

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

<table>
<tr><td>Version</td><td>IHL</td><td colspan="7">Type of Service</td><td colspan="3">Total Length</td></tr>
<tr><td colspan="9">Identification</td><td>Flags</td><td colspan="2">Fragment Offset</td></tr>
<tr><td colspan="2">Time to Live</td><td colspan="7">Protocol</td><td colspan="3">Header Checksum</td></tr>
<tr><td colspan="12">Source Address</td></tr>
<tr><td colspan="12">Destination Address</td></tr>
<tr><td colspan="11">Options</td><td>Padding</td></tr>
</table>

<table>
<tr><td colspan="9">Source Port</td><td colspan="3">Destination Port</td></tr>
<tr><td colspan="12">Sequence Number</td></tr>
<tr><td colspan="12">Acknowledgment Number</td></tr>
<tr><td>Data Offset</td><td colspan="2">Reserved</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="3">Window</td></tr>
<tr><td colspan="9">Checksum</td><td colspan="3">Urgent Pointer</td></tr>
<tr><td colspan="11">Options</td><td>Padding</td></tr>
<tr><td colspan="12">data</td></tr>
</table>

Kuva 2.2: IP- ja TCP-otsikot [Pos81b, Pos81c]

2.2.1.2 Asiakasohjelma

Asiakasohjelma luo liikennöintipisteen ja sitoo sen tiettyyn paikalliseen osoitteeseen kuten palvelinohjelma. Paikallisena porttinumerona sovellus käyttää jotain vapaana olevaa porttia, joka on tyypillisesti ensimmäinen vapaana oleva suurempi kuin 1 023. Eräissä sovelluksissa ohjelmaa ajetaan pääkäyttäjän oikeuksin, jolloin se voi varata portteja alle 1 024:n ja näin koettaa vakuuttaa vastapäälle olevansa erikoisoikeutettu prosessi esimerkiksi käyttäjän tunnistamista varten.

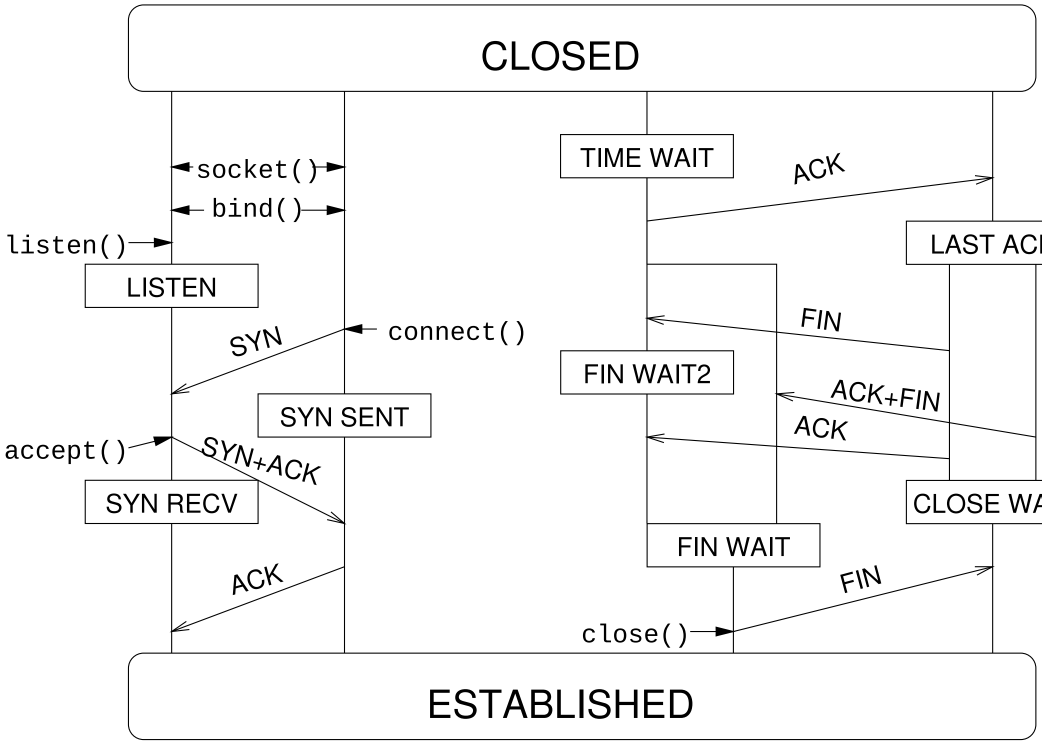
Luotuaan oman liikennöintipisteen asiakasohjelma ottaa yhteyden palvelinohjelmaan antamalla palvelimen osoitteen ja porttinumeron. Useimmiten asiakasohjelma selvittää palvelimen osoitteen tämän nimen perustella joko paikallisesta tietokannasta tai DNS-järjestelmältä (kappale 2.4). TCP-yhteys tunnistetaan neliköstä <*Lähdeosoite, Kohdeosoite, Lähdeportti, Kohdeportti*>.

2.2.1.3 Verkossa

Yhteyden luonnissa käytetään kolmivaiheista kättelyä.[2]. Asiakasohjelma lähettää kuvan 2.2 mukaisen IP- ja TCP-kehyksen verkkoon. Kehykseen on täytetty lähettävän[3] ja vastaanottavan[4] koneen IP-osoitteet, TCP-porttinumerot[5], sarjanumero[6]. Lisäksi SYN-lippu on päällä. Muiden kenttien käytöstä on tarkemmin lähteissä [Pos81b, Pos81c].

Vastaanottava palvelinkone vastaa tähän samanlaisella kehyksellä jossa lähde- ja kohdeosoitteet sekä -portit on vaihdettu keskenään. Sarjanumerona on riippumaton nume-

[2] three-way handshake
[3] Source Address
[4] Destination Address
[5] Source Port, Destination Port
[6] Sequence Number



Kuva 2.3: TCP-tilakone normaalitapauksessa.

ro. Kuittausnumeroksi[7] on asetettu alkuperäisen viestin sarjanumero lisättynä yhdellä sekä SYN ja ACK-liput on asetettu. Kun asiakaskone saa tämän, yhteys on sen kannalta luotu ja se lähettää kuittauksen, jossa ACK-lippu on päällä ja kuittausnumerona palvelimelta tulleen viestin sarjanumero lisättynä yhdellä. Yhteyden luominen on esitetty kuvassa 2.3. Tämän jälkeen liikennöinti voi alkaa.

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

2.2.1.4 Yhteyksien sarjanumeroiden valitseminen

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].

[7] Acknowledgment Number

2.2.2 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 lähettävä 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 osottautunut monissa sovelluksissa liian pieneksi, esimerkiksi maksimisiirtonopeus 5 000 kilometrin yhteysvälillä on 10 Mbit/s jo pelkästään valon fyysisen etenemisnopeuden vuoksi — verkkoelementtien aiheuttamat viiveet rajoittavat nopeutta edelleen [BBJ92].

Yhteyttä luotaessa voidaan neuvotella optiolla ikkunan skaalausparametrista N, jolloin ikkunan koko on $2^N \times window$. Koska lisäys on melko uusi — vuodelta 1992 — skaalaus tulee käyttöön yhteydellä vain mikäli molemmat ovat asetteneet tämän option yhteyttä luotaessa [BBJ92]. Muita optioilla neuvoteltavia ominaisuuksia ovat maksimi segmentin koko, jonka lähettäjä on valmis hyväksymään sekä aikaleimaoptio, jota voidaan käyttää siirtoviiveen määrittämiseen. Se toimii myöskin sarjanumeron jatkeena nopeilla ja viiveisillä yhteyksillä suojaten laskurin pyörähtämiseltä.

Yhteydelle varataan tietty määrä vastaanottopuskuria. Tämä on joko järjestelmäkohtainen vakio tai aseteltavissa yhteyskohtaisesti.

2.2.2.1 Datan vastaanotto

Vastaanottavassa koneessa linkkikerroksen rutiini tunnistaa verkkokerroksen protokollan ja siirtää kehyksen sisällön tämän protokollan käsiteltäväksi. Mikäli se on IP-sähke, IP-kerros: [Bra89b]

1. tarkistaa, että sähke on oikean muotoinen (pituus ja tarkistussumma täsmäävät)
2. tarkistaa, että se on osoitettu tähän järjestelmään
3. käsittelee optiot
4. kokoaa sähkeen, mikäli tarpeen
5. antaa viestin oikealle siirtokerroksen protokollalle

Saatuaan IP-paketin sisällön eli TCP-segmentin otsikoineen IP-kerrokselta, TCP-kerros laskee tarkistussumman, johon lasketaan varsinaisen TCP-otsikon ja -segmentin lisäksi 96 bittiä pitkä kuvan 2.4 mukainen lumeotsikko, joka muodostuu lähde- ja kohdeosoitteista, protokolla- sekä pituuskentästä, joka on TCP-otsikon ja datan pituuden summa.

Segmentti hylätään mikäli tarkistussumma on virheellinen. Samoin tarkistetaan, että segmentin sarjanumero on laillinen (kuva 2.5); ellei näin ole, lähetetään kuittaus takai-

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

Source Address		
Destination Address		
tyhjä	Protocol	TCP Length

Kuva 2.4: TCP-lumeotsikko [Pos81c].

Taulukko 2.1: TCP:n tilamuuttujat [Pos81c].

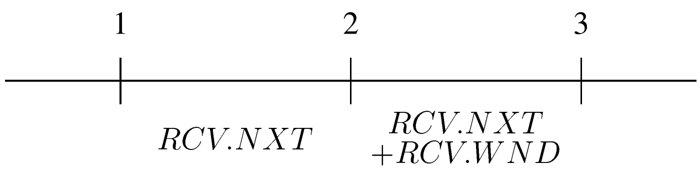
Muuttuja	Selitys
SND.UNA	ensimmäinen lähetetty mutta kuittaamaton (send unacknowledged)
SND.NXT	seuraavana lähetettävä (send next)
SND.WND	lähetysikkuna (send window)
SND.UP	lähetettävän kiirellisen datan osoitin (send urgent pointer)
SND.WL1	viimeisimmän ikkunan päivityksen sarjanumero (segment sequence 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)

Taulukko 2.2: Laajennetut lähettäjän TCP:n tilamuuttujat [Jac88].

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

1. kuitatut sarjanumerot
2. lähetetyt mutta kuittaamattomat sarjanumerot (täytyy olla puskuroituna)
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 2.5: TCP lähetys- ja vastaanottoavaruudet [Pos81c, s. 69].

sin laillisilla arvoilla ja segmentti hylätään. Loppuosa käsittelystä tapahtuu segmenttien mukaisessa järjestyksessä.

Seuraavaksi tarkistetaan lippujen sekä laskureiden arvot (taulukko 2.1) 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 Mikäli kuittausnumeron arvo on oikea eli $SND.UNA < SEG.ACK \leq SND.NXT$, kuittauslaskuri päivitetään ($SND.UNA = SEG.ACK$). lähetysikkuna, jos $SND.UNA < SEQ.ACK \leq SND.NXT$.

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 tyhjä. Mikäli PSH-lippu on päällä, käyttäjälle ilmoitetaan tästä palauttamalla vajaa puskuri.

Mikäli FIN-lippu on asetettu, lähettävällä päällä ei ole enää lähetettävää: annetaan sovellusohjelmalle kaikki puskureissa oleva tieto ja ilmoitetaan yhteyden loppumisesta. Lähetetään kaikki puskureissa oleva tieto vastaanottajalle kunnes sovellus sulkee yhteyden omaltakin puolelta. Yleisten ohjelmistorajapintojen ja TCP:n semantiikassa on erovuuksia, joka joudutaan huomioimaan sovellusten suunnittelussa. Tätä on selitetty tarkemmin kappaleessa 2.2.3 sivulla 11.

Mikäli siirto on jonkin aikaa yksisuuntaista, vastaanottaja lähettää pelkkiä kuittaussähkeitä takaisin. Näissä on voimassa oleva sarjanumero ($SEG.SEQ = SND.NXT$), päivitetty kuittausnumero ($SEG.ACK = RCV.NXT$), ikkunankoko ($SEG.WND = RCV.WND$) ja segmentin dataosa on tyhjä ($SEG.LEN = 0$).

2.2.2.2 Datan lähettäminen

Ohjelman kirjottaessa verkkoon, TCP-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 ($SND.NXT - SND.UNA$).

Yhteyden todetaan olevan ruuhkautunut, mikäli yksi tai useampi lähetetty segmentti katoaa. Segmenttien vaurioituminen on nykyisissä verkoissa hyvin epätodennäköistä ($\ll 1\%$), joten ruuhkatilanteen aiheuttama paketin hylkääminen on todennäköisin syy segmentin katoamiseen. Segmenttien katoaminen todetaan joko uudelleenlähetysajastimen laukeamisesta tai moninkertaisena kuittauksena, joka on osoitus siitä, että välistä on jäänyt segmentti tai useampi pois. Tällöin hitaan käynnistyksen kynnysarvoksi ($SSTRESH$) asetetaan puolet käytössä olevasta ikkunakoosta $min(SND.WIN, CWND)$, kuitenkin vähintään kaksi segmenttiä ($2 \times SEG.LEN$) [Ste97].

Monet WWW-selaimet avaavat nykyään useita rinnakkaisia yhteyksiä. Vaikka tämä useimmiten nopeuttaakin dokumenttien lataamista, heikentää se yleisesti Internetin suorituskykyä: kaikki rinnakkaiset yhteydet kärsivät samalla tavalla ruuhkasta ja näin menetetään mahdollisesti neljä pakettia yhden asemasta. Tämä ruuhkauttaa muitakin kuin vain ruuhkaisinta osuutta, koska uudenlleenlähetysten määrä kasvaa.

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

1. Maksimikokoinen segmentti voidaan lähettää.
2. Sovellus työntää[8] dataa ja kaikki jonossa oleva data voidaan lähettää.
3. Vähintään Fs:n suuruinen osa suurimmasta ikkunakoosta voidaan lähettää.[9]
4. Sovellus työntää dataa ja aikavalvonta (0,1 – 1,0 s) laukeaa.

Mikäli TCP- voi lähettää, se antaa segmentin IP-kerrokselle, joka [Bra89b]

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

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

2.2.3 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

[8] push
[9] Fs:n suositeltu arvo on 1/2.

mahdollista. Tämä vaatii ylempien kerrosten sovelluksia ottamaan tämän huomioon siten, että viimeksi lähettävä osapuoli sulkee yhteyden.

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

2.3 Hypertext Transfer Protocol

Yksi WWW:n nopean suosion perusteita oli sen kyky olla riippumaton käytetystä siirtoprotokollasta yhtenäisen resurssien nimeämiskäytännön (URI, URL) perusteella: näinollen olemassa olevia tietonsiirto- ja -hakumenetelmiä (WAIS, gopher, FTP) voitiin hyödyntää: sisältöä oli jo olemassa ja tavoitettavissa eikä käyttöönotossa ollut yleistä muna-kana -ongelmaa.

WWW-käyttöön suunniteltu HTTP-protokolla on nykyään Internetin suosituin protokolla sekä yhteyksien määrän että siirretyn tiedon suhteen. Se on melko yksinkertainen mutta kuitenkin joustava eri tietotyypeille. HTTP-protokollan nykyisin käytössä oleva versio 1.0 on määritelty lähteessä [BLFF96] ja uudempi versio 1.1 lähteessä [FGM$^+$97]. Pääosa asiakas- ja palvelinohjelmista on version 1.0 mukaisia, joskin uusin versio (1.2) yleisestä Apache-palvelinohjelmasta tukee jo versiota 1.1.

Dokumentin resurssinimestä asiakasohjelma tulkitsee:

tiedonsiirtomenetelmän Tämä voi olla esimerkiksi HTTP, FTP tai gopher; määrittää käytettävän protokollan.

verkko-osoite Verkko-osoite muodostuu yksinkertaisimmillaan koneen nimestä, mutta voi sisältää myös käyttäjäinformaation (tunnus, mahdollisesti salasana) ja käytettävän porttinumeron, mikäli se eroaa protokollan oletusarvosta.

paikallinen osa Paikallisesta osan asiakasohjelma välittää palvelimelle.

HTTP-protokollan tapauksessa asiakasohjelma luo TCP-yhteyden palvelinkoneeseen, porttiin 80, mikäli muuta ei ole määrätty. Luotuaan yhteyden asiakasohjelma lähettää pyynnön, esimerkkipyyntö on esitetty kuvassa 2.6. Palvelin vastaa tähän palatuttamalla pyydetyn dokumentin, mikäli se löytyi ja asiakasohjelma oli oikeutettu saamaan sen. Esimerkki vastauksen alusta on kuvassa 2.7.

2.3.1 Hypertext Markup Language

Yleisin WWW-dokumentti on esitetty HTML-muodossa. HTML on SGML-pohjainen yksinkertainen tekstin kuvauskieli, jolla voidaan esittää dokumentin looginen rakenne

```
GET /~puhuri/ HTTP/1.0
User-Agent: Mozilla/4.0b3 [en] (Win95; I)
Referer: http://www.hut.fi/~puhuri/
Host: keskus.hut.fi
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Charset: iso-8859-1,*,utf-8
```

Kuva 2.6: HTTP-pyyntö.

```
HTTP/1.0 200 OK
Date: Fri, 09 May 1997 20:32:59 GMT
Server: Apache/1.1.1
Content-type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML><HEAD>
<TITLE>Markus Peuhkuri: researcher’s home page</TITLE>...
```

Kuva 2.7: HTTP-vastaus.

ja littää tekstin kuvia. Loogisen rakenteen esittäminen ulkoasun asemasta mahdollistaa dokumenttien esittämisen optimaalisesti millä tahansa laitteella: esimerkiksi tekstipäätteellä, teksti-TV:ssä, matkapuhelimessa, paperilla ja puhesyntetisaattorilla normaalin graafisen näytön asemesta [Nie97].

HTML-määrittelyyn on nyttemmin lisätty enemmän mahdollisuuksia ulkoasun määrittelyyn, mikä on tuhonnut jonkin verran alkuperäistä ideaa. Samoin HTML-merkintöjä käytetään määritysten vastaisesti halutun ulkoasun saavuttamiseksi: niinpä sivu näyttää yleensä väärältä muilla selaimilla kuin mille se on suunniteltu.

Suurin osa dokumenteista sisältää varsinaisen sivun lisäksi myös kuvia. Nämä ovat erillisiä dokumentteja ja vaativat kukin oman yhteyden.

2.4 Nimipalvelu — DNS

Internet-osoitteet ovat 32-bittisiä numeroita, joka esitetään yleensä okteteittain desimaalisina pistein eroitettuna, esimerkiksi ”130.233.161.48”. Tälläiset numerosarjat ovat kuitenkin hankalia muistaa, joten on käytännöllisempää käyttää havainnollisia nimiä. Alunperin tämä tietokanta oli yksi Network Information Centerissa (NIC) ylläpidetty tiedosto, johon aina lisättiin koneita tarvittaessa. Muutosten jälkeen kaikki Internetin — silloisen ARPANET:n – koneet hakivat sen FTP:llä [HSF85, FHS85]. Tämä oli käyttökelpoista niin kauan kun koneita ei ollut paljoa.

Pian kuitenkin huomattiin, että pian tiedoston koko kasvaisi niin suureksi, että sen siirtäminen veisi ison osan siirtokapasiteetista. Tavittava siirtokapasiteetti riippuu konei-

Taulukko 2.3: `fi`-juuren nimipalvelimet.

palvelin	IP-osoite	sijainti (arvio)
NS1.EUNET.fi	192.26.119.7	Helsinki, Punavuori
HYDRA.HELSINKI.fi	128.214.4.29	Helsinki, Vallila
NS.EU.NET	192.16.202.11	Hollanti, Amsterdam
NS.UU.NET	137.39.1.3	USA
R2D2.JVNC.NET	128.121.50.2	USA, New York
NS.TELE.fi	193.210.19.19	Helsinki
NS.TELE.fi	193.210.18.18	Helsinki

den määrästä suhteessa $O(N^2)$ eli koneiden määrän kymmenkertaistuminen kasvattaa tarvittavan siirtokapasiteetin tarpeen satakertaiseksi.

Ratkaisuksi kehtitettiin Domain Name System (DNS), jossa nimien hallinta hajautettiin hallinta-alueittain [Moc87a, Moc87b]. Organisaatiot on jaoteltu hierrarkisesti. Ylimmällä tasolla on kahdeksan juuripalvelinta, jotka tietävät kaikki nimipalvelimet, jotka vastaavat päätasoista eli maista (`fi`, `se`) sekä loogisista, pääosin yhdysvaltalaista hallinta-alueista (`com`, `org`).

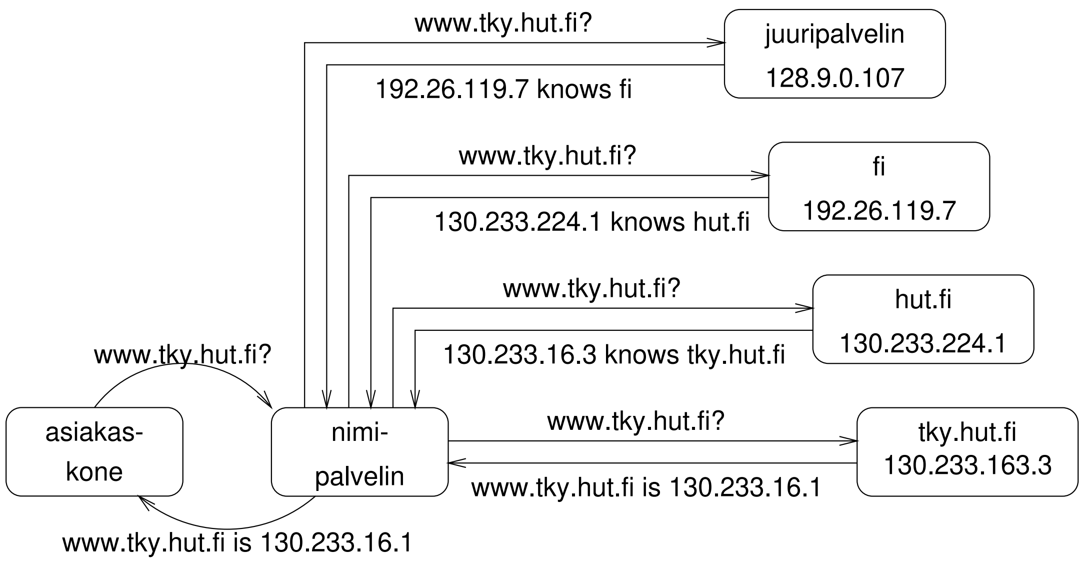
Päätasojen nimipalvelimet tietävät edelleen alla olevien organisaatioiden (`hut.fi`) nimipalvelimet ja näin edelleen. Lopulta hallinta-alueen nimipalvelin tietää sillä hallinta-alueella olevan koneen osoitteen. Jokaisella hallinta-alueella on oltava vähintään kaksi nimipalvelinta, jolloin toinen voi vastata mikäli toinen on alhaalla. Mitä isommasta ja tärkeämmästä hallinta-alueesta on kyse, sitä enemmän ja enemmän hajallaan nimipalvelimia on. Suomen päätason nimipalvelimet on esitetty taulukossa 2.3.

Sovellusohjelma kysyy käyttöjärjestelmältä `gethostbyname()`-kutsulla tai vastaavalla järjestelmään määritellyltä nimipalvelimelta koneen nimeä, esimerkiksi `www.tky.hut.fi`. Mikäli se ei tiedä sitä, se kysyy ensin juuripalvelimelta `www.tky.hut.fi`:n osoitteen. Juuripalvelin ei tätä yleensä tiedä, mutta se antaa listan `fi`-alueen palvelimien osoitteista. Alkuperinen nimipalvelin kysyy joltain näistä palvelimelta `www.tky.hut.fi`:n osoitetta ja saa todennäköisesti vastaukseksi `hut.fi`-alueen palvelimet. Näin jatkuu, kunnes joku palvelin vastataa koneen osoitteella tai ilmoittaa, että kyseistä konetta ei ole olemassa. Toiminta on esitetty kuvassa 2.8.

Järjestelmä vaikuttaa hyvin kuormittavalta, tarvitaanhan tässä tapauksessa neljä kyselyä yhden koneen osoitteen selvittämiseen. Aina ei kuitenkaan tarvita kyselyä jokaiselta tasolta vaan ylemmän tason palvelin voi pystyä vastaamaan suoraan. Esimerkiksi `ns1.hut.fi`, joka on yksi `hut.fi`-alueen palvelimista, on myös `tky.hut.fi`-alueen palvelimin ja pystyy siten vastaamaan suoraan kyselyyn.

Nimipalvelin tallentaa löytämänsä tiedot ja pystyy vastaamaan kyselyihin suoraan omasta tietokannastaan, niinpä päätasojen ja suosituimpien koneiden osotteet löytyvät todennäköisesti nimipalvelimen muistista. Jos esimerkiksi kohta `www.tky.hut.fi`:n haun jälkeen samalle nimipalvelimelle tehdään kysely `www.gra.hut.fi`:n osoitteesta, nimipalvelin osaa ohjata kyselyn suoraan `hut.fi`:n palvelimelle; juuripalvelimia ja `fi`-alueen palvelimia ei tarvitse vaivata.

Jotta koneiden nimiä ja osotteita voisi muuttaa, täytyy olla joku mekanismi, jolla nimet



Kuva 2.8: Nimipalvelukyselyn kulku.

voidaan poistaa muista tietokannoista. DNS:ssä tämä on toteutettu antamalla jokaiselle tiedolle elinaika, tyypillisesti päiviä tai tunteja. Tämän seurauksena päätasojen nimipalvelimien ja suosittujen koneiden tiedot ovat hyvin todennäköisesti nimipalvelimen tietokannassa valmiina ja se voi palauttaa ne välittömästi. Myös eräät käyttöjärjestelmät ja sovellusohjelmat pitävät omaa pientä käteismuistia viimeksi kysytyistä osotteista.

Luku 3

WAKA — WWW aikakäytettävyysanalysaattori

Erilaiset tekijät vaikuttavat sivun hakemiseen kuluvaan aikaan, niinpä yksiselitteisen lukuarvon laskemine on käytännössä mahdotonta. Ainoa mittari on aika, joka kuluu sivun lataamiseen käyttäjältä. Tässä vaiheessa sivun muuttaminen on tietenkin liian myöhäistä, jotten jonkinlaista etukäteisarvio aikarajoista olisi paikallaan.

Kehitetty ohjelma määrittää sivun lataukseen kuluvaa aikaa lataamalla sivun komponentteineen ja mittaamalla tähän kuluvan ajan. Tämän lisäksi ohjelma määrittää kulkuviiveitä verkossa ja näiden perustella laskee eri aikarajoa eri olettamuksille.

3.1 Ohjelman toiminta

Ohjelma ottaa komentoriviltä annetun resurssimääreen (URL) ja hakee tämän osoittaman dokumentin. Mikäli dokumentti on HTML-dokumentti, se käydään läpi etsien kuvia, appletteja ja kehysmääritteitä. Mikäli sivu muodostuu kehyksistä, kehyksen määrittävät sivut käydään myös läpi vastaavasti. Ohjelma hakee sivu(i)lla olevat kuvat ja appletit ja mittaa niiden siirtämiseen kuluvan ajan. Tämän jälkeen yhteydet nimipalvelimiin ja WWW-palvelimiin määritetään ICMP-paketeilla.

3.1.1 Nimipalvelun suorituskyky

Nimipalvelun suorituskyky mitataan selvittämällä jokaisen tason nimipalvelimet, yhteensä N kappaletta. Tämän jälkeen kuhunkin (i) nimipalvelimeen mitataan kolmen edestakaisen 200 tavun mittaisen ICMP-paketin aika (t_{dns_i}). Kustakin ajasta vähennetään aika, joka kuluu paikalliseen nimipalvelimeen, aika on kuitenkin vähintään 1 ms. Tämä perustuu olettamukseen, että tämä nimipalvelin on ”matkan varrella” ts. lähellä reititintä, josta ulkomaailman liikenne lähtee. Tällä vähennyksellä on erityisen suuri merkitus silloin kun yhteys omaan nimipalvelimeen ja internetiin on hitaalla modeemiyhteydellä.

Paketin kooksi valittiin 200 tavua, koska keskimääräinen DNS-vastaus on hiukan yli 200 tavua, kysely on yleensä alle 50 tavua. Nimipalvelukyselyn tehollinen aika (t_{dns}) lasketaan seuraavalla kaavalla:

$$t_{dns} = 0.5 \times \left(0.1 t_{dns_1} + \left(\sum_{1<i<N} 0.5^{N-i} \times t_{dns_i} \right) + t_{dns_N} \right) \tag{3.1}$$

Kaava perustuu olettamukseen, että vain joka toinen kerta tarvitsee tehdä nimipalvelukysely, jonka lisäksi ylempien tasojen kyselyitä tarvitsee tehdä tätäkin harvemmin. Tämä aika lasketaan jokaiselle erilliselle palvelimelle, josta dokumentteja joudutaan hakemaan.

3.1.2 Tiedonsiirron suorituskyky

Yhteyksien laatua arvioitiin kuhunkin käytetyyn WWW-palvelimeen lähettämällä ensin kolme 40 tavun mittaista ja tämän jälkeen kolme 576 tavun ($K_s + 40$) mittaista ICMP-pakettia ja mittaamalla näihin kulunut aika. Ensimmäisten pakettien koko ja aika (t_p) vastaa kuittauspaketin välitykseen kulunutta aikaa ja jälkimmäiset (t_s) datapaketteja.

Käytettävissä oleva tiedonsiirtonopeus arvoitiin pakettien pituuksien perusteella seuraavasti kaavan 3.2 mukaan. Oletuksena on, että pidempiin paketteihin kuluu pitempi aika. Johtuen Internet-verkon dynaamisesta luonteesta ja mittauksen eriaikaisuudesta, näin ei välttämättä ole. Tässä tapauksessa ohjelma tulostaa varoituksen ja suorittaa mittaukset toisen kerran ja mahdollisesti kolmennen kerran. Mikäli kolmaskaan kerta ei tuota järkeviä tuloksia, kaistanleveydeksi asetetaan 128 kbit/s.

$$B_w = \frac{2 \times K_s}{t_s - t_p} \tag{3.2}$$

Näiden perusteella laskettiin kuluva aika seuraavasti (tiedoston koko on K_d, ikkunankoko on S_w tavua, segmentin dataosan koko K_s)

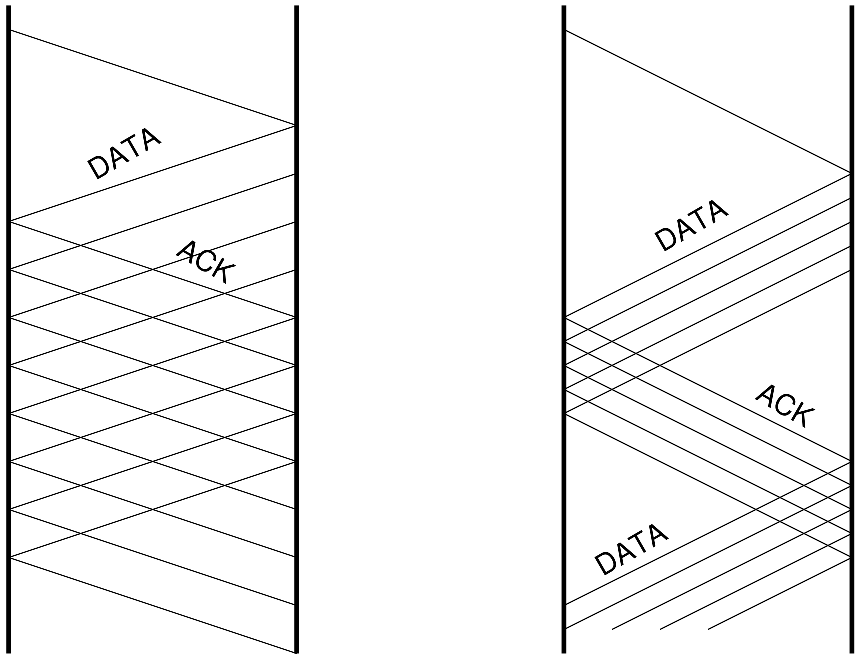
$$t_{tot} =$$

$$t_p + t_s + \frac{\lceil \frac{K_d}{K_s} - 1 \rceil \times K_s}{B_w} \quad , \quad K_d \leq S_w \text{tai} \frac{t_p + t_s}{2} \leq t_w \tag{3.3}$$

$$t_p + t_s + \frac{\lceil \frac{K_d}{K_s} - 1 \rceil \times K_s}{B_w} + \lceil \frac{K_d}{S_w} \rceil \times (t_w - t_p), \quad , \quad K_d > S_w \text{ja} \frac{t_p + t_s}{2} > t_w \tag{3.4}$$

$$\tag{3.5}$$

Tiedonsiirrossa voidaan eroittaa kaksi eri tapausta: tapaus jossa siirto on kaistanleveyden rajoittamaa (kaava 3.3, kuva 3.1(b)). ja siirto, joka on siirtoikkunan koon ja viiveen (S_w) rajoittamaa (kaava 3.4, kuva 3.1(a)). Kaavassa esiintyvä t_w on täyden ikkunan siirtoon kuluva aika eli $t_w = W_s / B_w$.



(a) Kaistanleveyden rajoittama siirto

(b) Viiveen rajoittama siirto

Kuva 3.1: Kaksi eri tapausta siirron aikajakaumasta.

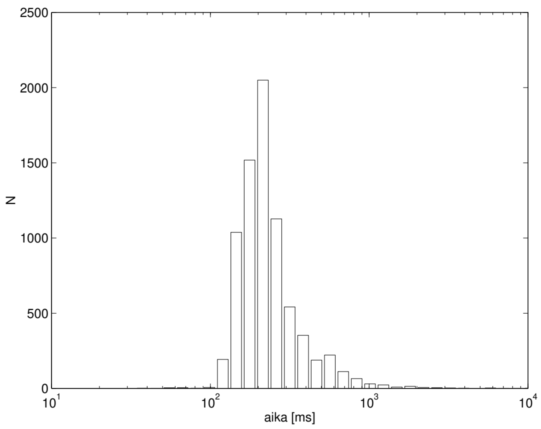
Edelliset kaksi kaavaa edellyttivät, että yhtään segmenttiä ei katoa siirron aikana. Valitettavasti yleensä tilanne ei ole näin hyvä vaan todellisuudessa segmenttejä katoaa. Tässä segmenttien katoaminen on katsottu todennäköisyytenä, jona aikoja määritettäessä ICMP-paketteja katosi. Suurin sallittu häviö on $\frac{2}{3}$ koska molemmista mitttauksista on saatava vähintään yksi läpimennyt paketti. Mikäli paketteja katosi todennäköisyydellä P_{loss}, lisätään siirtoaikaa kaavan 3.6 mukaan.

$$t_{tot} = t_{tot} + P_{loss} \times \lceil \frac{K_d}{K_s} \rceil \times \left(\frac{t_p + t_s}{2} + \frac{K_s}{B_w} \right) \tag{3.6}$$

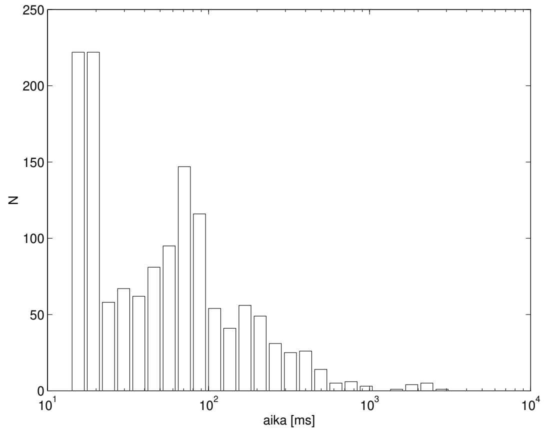
Kaava olettaa, että segentin katoaminen huomataan heti seuraavasta segmentistä eli ajan $\frac{K_s}{B_w}$ jälkeen, jonka jälkeen uudelleenlähetykseen kuluu aika $\frac{t_p+t_s}{2}$. Uudelleenlähetyksen viemä tehollinen aika voi olla pienempikin mikäli ikkunankoko on riittävä säilyttämään jatkuvan vuon. Toisealta taas havaitsemisaika voi olla merkittävästi suurempi, mikäli häviö sattuu yhteyden viimeisiin segmentteihin, jolloin häviö huomataan vasta aikavalvonnan lauettua edestakaisen viiveen jälkeen. Mikäli jompikumpi ensimmäisistä segmenteistä katoaa, tämä havaitaan vasta parin minuutin jälkeen.

3.2 Tuloste

Ohjelma tulostaa tiivistelmän dokumentin rakenteesta ja sekä suoraan mitatun että mittausten perusteella arvioidun siirtoajan. Tämän jälkeen ohjelma tulostaa taulukon 3.1



(a) com-hallinta-alueen vasteaikojen jakauma.



(b) fi-hallinta-alueen vasteaikojen jakauma.

Kuva 3.2: Vasteaikojen jakauma 512 tavun suuruiselle ICMP-paketille.

Taulukko 3.1: Esimerkkitapauksien parametrit.

Yhteys	sijainti	t_p [ms]	t_s [ms]	kaistanleveys [kbit/s]	P_{loss} %
Ethernet	sama LAN	1	1.35	3000	0
Kiinteä yhteys	sama maa	43	50	150	6
	toinen mantere	185	220	30	22
Modeemiyhteys	sama maa	105	150	24	6
	toinen mantere	250	320	15	22

mukaisten tapausten latausajat kyseiselle dokumentille. Arvot perustuvat taulukossa 3.2 esitettyihin mittaustuloksiin, jotka oli tehty marraskuun puolessa välissä 1996. Näiden mittauksien histogrammit on esitetty kuvissa 3.2(a) ja 3.2(b). Koneet oli valittu satunnaisesti koneista `www.*.com` ja `www.*.fi`.

Aikoihin lisätään nimipalvelukyselyn aika $\frac{1}{2}t_p$, mikä on ehkä $\frac{1}{3}$ yhden kyselyn ajasta, mutta vastaa todennäköisyyttä, jolla nimipalvelu joudutaan tekemään.

3.3 Kehitettävää

Ohjelman käyttämä HTML-jäsennin ei ole niin virhesietoinen kuin useimmat selaimet. Mikäli jonkun sivun osalta dokumenttimäärä ei vastaa selaimen ilmoittamaa, kannattaa sivu tarkistaa HTML-tarkastimella.

Ohjelman siirtämä tietomäärä on mahdollisesti liian pieni, mikäli sivulla on käytetty appletteja, jotka lataavat edelleen osadokumentteja, itse appletti kyllä ladataan. Tämän

Taulukko 3.2: Edestakaiset ajat [ms] 512 tavun ICMP-paketeille.

Domain	minimi	mediaani	keskiarvo	maksimi	P_{loss}	N
`.fi`	13.9	48.3	100.8	3138.9	5,9 %	1391
`.com`	35.7	215.3	268.8	11448.0	21,9 %	7531

mittaaminen olisi vaatinut kokonaisen Java-tulkin toteuttamista, mikä on reilusti tämän työn ulkopuolella. Myöskään mahdollisia JavaScript- ja ActiveX-ohjelmia ei suoriteta. Tämän osalta käyttökelpoisinta olisi mitata liikennettä verkosta ja käyttää normaalia selainta.

Tiedonsiirrossa ei käytetä apuna mahdollisan välimuisteja. Tämän käyttöönotto on kuitenkin helppoa määrittelemällä oikeat ympäristömuuttujat ja lisäämällä ohjelmakoodin nämä aktivoiva funktiokutsu. Ohjelman tuottama tuloste on kovin karu. Paremman ulkoasun tulostukselle saisi käyttämällä selainta käyttöliittymänä.

Ohjelma toimii käytännöllisesti katsoen kaikissa UNIX-järjestelmissä. Vaatimuksena on perl-tulkki varustettuna libwww-perl eli LWP-paketilla sekä ping- ja nslookup-ohjelmat. Ohjelma on riippuvainen ohjelmien tulostusmuodosta ja niiden vastaanottamista argumenteista, joten näitä voidaan joutua muuttamaan. Kyseisten toimintojen integroimista ohjelmaan haittaa se, että ICMP-pakettien lähettäminen vaatii pääkäyttäjän oikeuksia.

Ohjelman laskenta perustuu HTTP-protokollan versioon 1.0. Versio 1.1 nopeuttaa moniosaisten dokumenttien latausta [NGBSP97]. Lisäksi useat 1.0-protokollaa käyttävät ohjelmat kännistävät useita rinnakkaisia yhteyksiä (4 – 6 oletuksena), mikä nopeuttaa tiedonsiirtoa mutta toisealta ruuhkauttaa verkkoa koska kestää hetken ennenkuin TCP ehtii mukautua nopeasti lisääntuvään liikenteeseen.

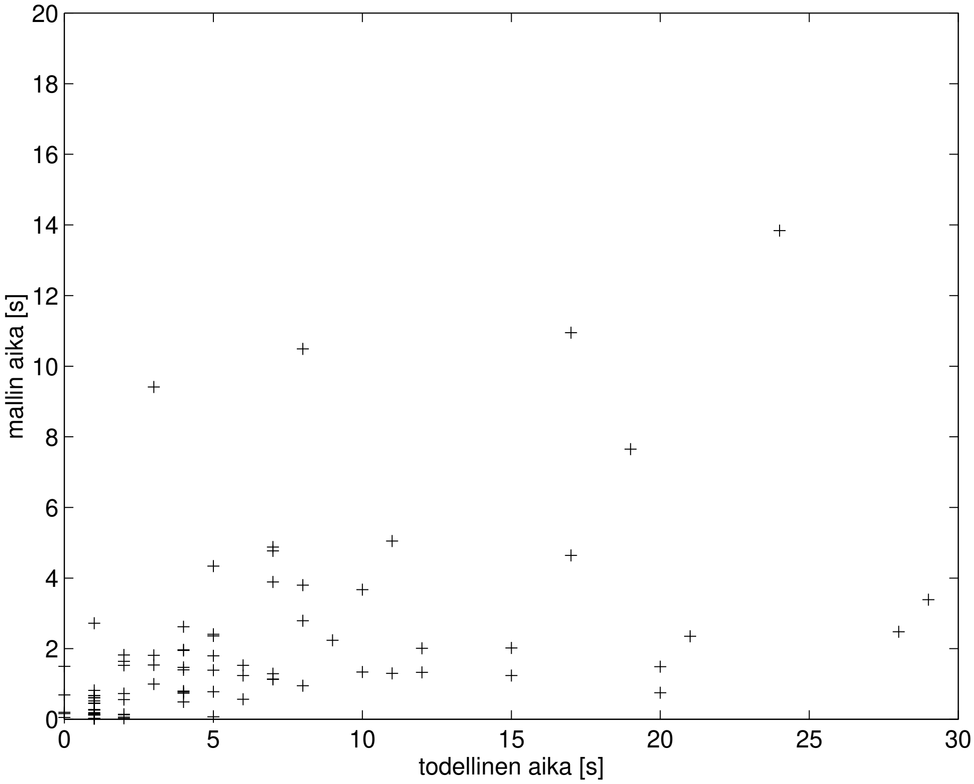
Luku 4

Tulokset

Ohjelmalla mitattiin 96 dokumentin latausajat. Dokumentit oli valittu melko satunnaisesti niiden dokumenttien joukosta, jotka löytyivät käyttämäni WWW-selaimen historiatietokannasta. Karsinta suoritettiin poistamalla ensin kaikki kuvat ja dokumentiti, jotka oli ohjelmallisesti tuotettu (dokumentin nimessä oli merkkinono ”cgi” tai ”?”). Tämän jälkeen karsintaa jatkettiin siten, että kustakin palvelimesta valittiin korkeintaan kolme dokumenttia; vähemmän, mikäli samalla organisaatiolla oli useita palvelimia.

Mittaustulos saatiin 93 dokumentista. Tuloksista laskettiin korrelaatio todellisen mitatun ajan ja mallin mukaisen ajan välillä. Lineaarikorrelaation arvoksi saatiin 0,63, mikä on arvausta parempi mutta ei kovin hyvä. Malli antoi yleensä vastaukseksi pienempiä arvoja: keskiarvo oli 4,5 s (mediaani 1,3) kun todellisissa mittauksissa keskiarvo oli 10,4 s (mediaani 4). Yksittäisten mittausten tulokset on esitetty kuvassa 4.1; kuvasta on jätetty pois 10 suurinta tulosparia selvyyden vuoksi; maksimit olivat 144 ja 101 sekunttia.

Muutamat HTTP-palvelimet eivät vastanneet ICMP-viesteihin vaikka dokumenttien lataaminen onnistui. Tämä on mahdollisesti estetty turvallisuussyistä [Cen96]. Mittauksessa ei käytetty nimipalvelun suorituskyvyn määritystä koska sen mittaaminen osoittautui hyvin hitaaksi. Tämä jätettiin pois myös vertailutapauksista.



Kuva 4.1: Tulosparit.

Yhteenveto

Ohjelman kehittämisessä törmättiin kahteen ongelmaan: suurimmaksi ongelmaksi muodostui ajanmittauksen tarkkuus. Käytetyssä LINUX-käyttöjärjestelmässä verkkoliikenteen ajanmittaus ei onnistu luotettavasti kuin 10 ms tarkkuudella. Sovellusohjelmat pystyvät tosin parempaan (1 μs) tarkkuuteen, mutta verkosta tulleen kehyksen tuloaikaa ei tällä tarkkudella pystytä määrittämään johtuen muista ajossa olevista ohjelmista.

Näinollen ICMP-viesteihin perustuvat mittaukset antavat epäluotettavia tuloksia. Lähteessä [CC96] käytettiinkin Silicon Graphics'in työasemissa olevaa tarkkaa kelloa, jolla päästään alle mikrosekunttien tarkkuuteen. Toinen aikaan liittyvä ongelma on perl-kielessä olevan aikarutiinin resoluutio, joka on yksi sekuntti, minkä seurauksena nopeilla yhteyksillä dokumenttien mitatut latausajat ovat joko 0 tai 1 sekunttia. Näinollen siirrosta lasketun kokonaisajan ero todella kuluneeseen aikaan sekuntteina saattioi olla sivulla olevien elementtejen lukumäärän suuruinen.

Toinen ongelma liittyy käytettävissä oleviin oikeuksiin. ICMP-pakettien lähettäminen vaatii pääkäyttäjän oikeuksia joten kenen tahansa käytettävissä olevan työkalun luominen edellyttää valmiiden rutiinien, tässä tapauksessa ping-ohjelman käyttöä. Tästä seuraavat myös ongelmat kaistanleveyden määrittämisessä, lähteessä [CC96] käytettyä menetelmää ei pystytty käyttämään.

Kehitetty ohjelma on, vaikka onkin raakile, kohtuullisen käyttökelpoinen, näkeehän siitä helposti, mistä ja miten suurista osasista dokumentti muodostuu. Eräs yksinkertainen lisäys olisi tuottaa taulukkomuotoinen tieto, jossa olisi näkyvissä kunkin osan vaikutus. Ajan mittausongelmien ratkaiseminen parantaisi todennäköisesti mittauksien luotettavuutta.

Kirjallisuutta

[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.

[BLMM94] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform resource locators (URL). Request for Comments (Proposed Standard) RFC 1738, Internet Engineering Task Force, December 1994. URL:ftp://ds.internic.net/rfc/rfc1738.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.

[CC96] Robert L. Carter and Mark E. Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance Evaluation*, 27&28:297 – 318, 1996.

[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.

[Cen96] CERT Coordination Center. Denial-of-service attack via ping. CERT(sm) Advisory CA-96.26, CERT Coordination Center, Dec 1996.

[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.

[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.

[FHS85] E. Feinler, K. Harrenstien, and M. Stahl. Hostname server. Request for Comments (Historical) RFC 953, Internet Engineering Task Force, October 1985. (Obsoletes RFC0811). URL:ftp://ds.internic.net/rfc/rfc953.txt.

[HSF85] K. Harrenstien, M. Stahl, and E. Feinler. DoD internet host table specification. RFC 952, Internet Engineering Task Force, October 1985. URL:ftp://ds.internic.net/rfc/rfc952.txt.

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

[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.

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

[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.

[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.

[Nie97] Jakob Nielsen. The difference between web design and gui design. Technical report, Usable Information Technology, May 1997. URL:http://www.useit.com/alertbox/9705a.html.

[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.

[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.

[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.

[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.

Hakemisto

Taulukot

Kuvat

Liite A

Ohjelman listaus

```perl
package FindElems;
use HTML::Entities ();

use strict;
use vars qw(@ISA
            %images %frames %links %bgimages %applets
            $base
            &canon
           );

require HTML::Element;
require HTML::Parser;
@ISA = qw(HTML::Element HTML::Parser links images frames links bgimages);

sub new
{
    my $class = shift;
    my $self = HTML::Element->new('html');  # Initialize HTML::Element part
    $self->{'_buf'} = '';  # The HTML::Parser part of us needs this

    # Initialize parser settings
    $self->{'_implicit_tags'}  = 1;
    $self->{'_ignore_unknown'} = 0;
    $self->{'_ignore_text'}    = 0;
    $self->{'_warn'}           = 0;

    # Parse attributes passed in as arguments
    my %attr = @_;
    for (keys %attr) {
      $self->{"_$_"} = $attr{$_};
    }
    $base = new URI::URL $self->{'_BASE'};

    # rebless to our class
    bless $self, $class;
}

sub start
{
    my($self, $tag, $attr) = @_;
    my($uri);

    if ($tag eq 'img') {
      $uri = new URI::URL $attr->{'src'}, $base ;
      $images{$uri->abs}++;
    } elsif ($tag eq 'frame') {
      $uri = new URI::URL $attr->{'src'}, $base ;
      $frames{$uri->abs}++;
    } elsif ($tag eq 'a') {
      if (defined $attr->{'href'}) {
        $uri = new URI::URL $attr->{'href'}, $base ;
        $links{$uri->abs}++;
      }
    } elsif ($tag eq 'applet') {
      my ($uri2) = $base;
      if (defined $attr->{'codebase'}) {
        $uri2 = new URI::URL $attr->{'codebase'}, $base ;
      }
      $uri = new URI::URL $attr->{'code'}, $uri2->abs;
      $applets{$uri->abs}++;
    } elsif ($tag eq 'body') {
      if (defined $attr->{'background'}) {
        $uri = new URI::URL $attr->{'background'}, $base ;
        $bgimages{$uri->abs}++;
      }
    } elsif ($tag eq "param"){
    }
}


sub end
{
    my($self, $tag, @stop) = @_;
    # End the specified tag
}


sub text
{
    my $self = shift;
    1;
}

sub links
{
  my (%tmp) = %links;
  undef %links;
  return %tmp;
}
sub images { my (%tmp) = %images; undef %images; return %tmp; }
sub frames { my (%tmp) = %frames; undef %frames; return %tmp; }
sub links { my (%tmp) = %links; undef %links; return %tmp; }
sub bgimages { my (%tmp) = %bgimages; undef %bgimages; return %tmp; }
sub applets { my (%tmp) = %applets; undef %applets; return %tmp; }

sub canon {
  my($self) = shift;
  my($url) = shift;
  my($bas) = shift;

  $bas = $self->{'_BASE'}
  if (! defined $bas);
  if ($url =~ m!^[a-zA-Z-]+:!) {
    return $url;
  } elsif ($url =~ m!^/!) {
    if ($bas =~ m!^([a-zA-Z-]+://[^/]+)!) {
      return $1 . $url;
    } else {
      warn "Strange base: $bas";
      return $url;
    }
  } else {
    if ($bas =~ m!^([a-zA-Z-]+://.+/)[^/]*!){
      return $1 . $url;
    } else {
      warn "Strange base: $bas";
      return $url;
    }
  }
}

1;
```

```
#!/usr/bin/perl

#$debug = 20;
use English;
use Socket;
use FileHandle;
use LWP::UserAgent;
use FindElems;
require "perfm2host";

#########################################
# Definations for measurements          #
#########################################
$winsize = 8196;                # common on PC's
$segsize = 536;                 # common over long links
$dnsperf = 0;                   # don't measure dns performance
#########################################

$ua = new LWP::UserAgent;
$ua->agent("UsabilityAnalyzer/0.1 " . $ua->agent);
# $ua->env_proxy(); # get from env

$debug = 0;
$documenturl = shift @ARGV;
&getdoc ($documenturl);

my($rhost);
for $rhost (keys %HOSTS) {
  my(@tim_s); my(@tim_p); my($Bw) = -1;
  my ($i) = 3; my ($ping_succ) = 0; my( $ping_tot) = 0;
  while ($Bw < 0) {
    printf STDERR "Negative BW ($Bw), retrying..$i\n"
        if ($i < 3);
    if ($i-- < 0) {
      print STDERR "Cannot determine bandwidth, assuming 128kbit/s!\n";
      $Bw = 128e3 / 8.0;
      last;
    }
    @tim_p = &pingstat ($rhost, 20, 3);
    @tim_s = &pingstat ($rhost, 557, 3);
    $ping_tot += 6;
    $ping_fails = 2 - $#tim_s + 2 - $#tim_p ;
    if ($#tim_s == -1 || $#tim_p == -1) {
      print STDERR "Either of pings to $rhost failed\n";
      next;
    }
    $Bw = 2 * 537 * 1000.0 / ( &avg(@tim_s) - &avg(@tim_p) );
  }
  $t_s{$rhost} = &avg(@tim_s);
  $t_p{$rhost} = &avg(@tim_p);
  $dnsp{$rhost} = &dnsperf($rhost) if $dnsperf;
  $Ploss{$rhost} = $ping_fails / $ping_tot;

  printf "%s: %g bit/s\n", $rhost, $Bw;
}

$realtime = 0;
$esttime = 0;
for $rq (sort keys %rq_time) {
  local ($h) = $url2host{$rq};
  $esttime += &time_est ($t_p{$h}/1.0e3, $t_s{$h}/1.0e3,
                         $Ploss{$h}, $rq_bytes{$rq});
  $realtime += $rq_time{$rq};

  printf "Esttime %f Real time %f\n", $esttime, $realtime if $debug;
}

printf "To retrive \'%s\' and all its components takes:\n", $documenturl;
printf "\tIn real:    %8.2f s\n", $realtime;
printf "\tBy model:   %8.2f s\n", $esttime;
printf "\tOn Ethernet %8.2f s\n", &get_comp (1e-3, 1.3e-3, 0, values %rq_bytes);
printf "\tOver fixline%8.2f s (same country)\n",
    &get_comp (43.0e-3, 50.0e-3, 0.06, values %rq_bytes);
printf "\tOver fixline%8.2f s (other continent)\n",
    &get_comp (185.0e-3, 220.0e-3, 0.26, values %rq_bytes);
printf "\tOver modem  %8.2f s (same country)\n",
    &get_comp (105.0e-3, 150.0e-3, 0.06, values %rq_bytes);
printf "\tOver modem  %8.2f s (other continent)\n",
    &get_comp (250.0e-3, 320.0e-3, 0.26, values %rq_bytes);

sub get_comp {
  local($t_p, $t_s, $ploss, @bytes) = @_;
  local($t_est) = 0.0;
  local($bytes);

  for $bytes (@bytes){
    $t_est += &time_est ($t_p, $t_s, $ploss, $bytes);
    printf " %f", $t_est if $debug > 2;
  }
  print "\n" if $debug > 2;
  return $t_est;
}
sub time_est {
  local($t_p, $t_s, $ploss, $bytes) = @_;
  local ($B_w) = 2 * 537.0 / ( $t_s - $t_p);
  local ($t_w) = $winsize / $B_w;
  local ($t_est) = 0.0;

  print "($t_p, $t_s, $ploss, $bytes [$t_w $B_w] " if $debug > 5;
  $t_est += $t_p + $t_s + int($bytes / $segsize) * $segsize / $B_w;

  if ($bytes > $winsize && ($t_p + $t_s) > 2 * $t_w) {
    $t_est += int($bytes / $winsize + 0.9999) * ($t_w - $t_p);
  }
  print "= $t_est" if $debug > 5;
  if ($ploss > 0) {
    $t_est += $ploss * int($bytes / $segsize + 0.9999) *
        (($t_p + $t_s) / 2 + $segsize / $B_w);
  }
  print " $t_est)" if $debug > 5;
  return $t_est;
}

sub getdoc {
  local ($baseurl) = shift;
  local ($url) = new URI::URL $baseurl;
  local ($t0, $t1);
  local ($req);
  local ($timetoget) = 0;
  local ($redirects);
```

```
  return
      if (defined $rq_bytes{$url->as_string});
  for ($redirects = 0; $redirects < 5; $redirects++) {
    $req = new HTTP::Request GET => $url;

    $url2host{$url} = $url->host;
    $HOSTS{$url->host}++;
    $t0=time;
    $response = $ua->simple_request($req);      # get it
    $rq_time{$url} = time - $t0;

    if ($response->is_error) {
      printf "Failed to get \'%s\'\n", $url;
      return;
    }

    printf "%s: %d bytes\n", $response->base, length($response->content());
    $rq_bytes{$response->base} = length($response->content());
    if (! $url->eq($response->base)) {
      printf STDERR "Base not equal to request without redirect!\n %s\n %s\n",
      $url, $response->base;
      $rq_bytes{$response->base} = 0;
    }

    last
        if (! $response->is_redirect);
    printf "Req:\t%s =>\n\t%s\n", $url,
    $response->header('Location');

    $url = new URI::URL $response->header('Location');
  }

  return
      if ($response->content_type ne "text/html");

  $parse = new FindElems(BASE => $response->base);
  $parse->parse($response->content());
  local (%links) = $parse->links;
  if ($debug > 20 && defined %links) {
    printf "Links: total %d\n", $#links;
    for $lnk (sort keys %links) {
      printf "%d %s\n", $links{$lnk}, $lnk;
    }
  }
  local (%images) = $parse->images;
  if (defined %images) {
    print "Images:\n";
    for $lnk (sort keys %images) {
#      printf "%d %s\n", $images{$lnk}, $lnk;
      &getdoc ($lnk);
    }
  }
  local (%bgimages) = $parse->bgimages;
  if (defined %bgimages) {
    print "Background images:\n";
    for $lnk (sort keys %bgimages) {
#      printf "%d %s\n", $bgimages{$lnk}, $lnk;
      &getdoc ($lnk);
    }
  }
  local (%frames) = $parse->frames;
  if (defined %frames) {
    print "Frames:\n";
    for $lnk (sort keys %frames) {
#      printf "%d %s\n", $frames{$lnk}, $lnk;
      &getdoc ($lnk);
    }
  }
  local (%applets) = $parse->applets;
  if (defined %applets) {
    print "Applets:\n";
    for $lnk (sort keys %applets) {
#      printf "%d %s\n", $applets{$lnk}, $lnk;
      &getdoc ($lnk);
    }
  }
}
```

```
#!/usr/bin/perl

use English;
sub dnsperf {
  my ($host) = shift;
  my (@doms) = split (/\./, $host);
  my ($ns);
  my ($dom) = "";
  my ($i);
  my (@tim);
  my (@resps);
  my ($total) = 0;
  my (%hostsknown);

  if (!defined $localns) {
    $localns = 100000;          # 100 s should be enought
    my ($myhost) = `hostname`;
    chop $myhost;
    for $ns (&nameservers ($myhost)) {
      ($ns, $nsia) = split (/,/, $ns);
      if ((@tim = &pingstat ($ns, 200, 3)) == undef) {
        printf STDERR "%s: all requests lost\n", $ns;
        next;
      }
      $localns = &min(($localns, &avg(@tim)));
    }
  }
  for ($i = $#doms; $i >= 0; $i--) {
    my (@nsweight) = ();
    my ($nscount) = 0;

    $dom = $doms[$i] . "." . $dom;
    printf STDERR "Querying nameservers for %s\n", $dom
        if $debug > 5;
    for $ns (&nameservers($dom)) {
      $nscount++;
      ($ns, $nsia) = split (/,/, $ns);
      if (++$hostsknown{$nsia} > 1) {
        printf STDERR "\t%s\t%s\n", $ns, "known"
            if $debug > 5;
        next;
      }
      if ((@tim = &pingstat ($ns, 200, 3)) == undef) {
        printf STDERR "%s: all requests lost\n", $ns;
        next;
      }
      push @resps, &max((&avg(@tim) - $localns, 1));
      push @nsweight, exp(2 - 2*$nscount);

      printf STDERR "\t%s\t%4.1f\t%g\n", $ns, $resps[$#resps],
      $nsweight[$#resps] if $debug > 5;
    }
    #
    # Here is the weighted average calculated
    #
    if (defined @resps) {
      if ($i == $#doms) {
        $total += &wavg(@resps, @nsweight) * 0.1;
      } elsif ($i == 0) {
        $total += &wavg(@resps, @nsweight);
      } else {
        $total += &wavg(@resps, @nsweight) * exp (i * log (0.5)) ;
      }
    }
    printf STDERR "\t%s\t%4.1f\n", "total", $total
        if $debug > 5;
    undef @resps;

  }
  return $total * 0.5;
}

sub pingstat {
  my ($host) = shift;
  my ($size) = shift;
  my ($count) = shift;
  my ($preload) = 2;
  my (@stat);
  my ($pingline) = $UID ?
      sprintf("ping -c %d -s %d -n %s|",
              $count, $size, $host) :
                  sprintf("ping -c %d -s %d -l %d -n %s|",
                          $count, $size, $preload, $host);

  printf "Executing: %s\n", $pingline if $debug > 8;
  open (PING, $pingline) || die "Cant $pingline $!";

  while (<PING>) {
    if (m/^(\d+) bytes from .* icmp_seq=(\d+) ttl=(\d+) time=(\S+) ms/) {
      push @stat, $4;
    }
  }
  return @stat;
}

sub nameservers {
  my ($hord) = shift;
  my ($auth) = -1;
  my (@nslist);
  my (@iplist);

  open (NSL, sprintf ("nslookup -type=any %s|", $hord)) ||
      die "nslookup not started: $!";
  while (<NSL>) {
    if (m/^Non-authoritative answer/) {
      $auth = 0; next;
    } elsif (m/^Authoritative answers can be found from:/){
      $auth = 1; next;
    } elsif (m/^(\S+)\s+canonical name = (\S+)/) {
      close NSL;
      return &nameservers($2);
    }
    if ($auth == 0) {
      if (m/orgin = (\S+)/){
      } elsif (m/mail addr = (\S+)/){
      } elsif (m/serial = (\S+)/){
      } elsif (m/refresh = (\S+)/){
      } elsif (m/retry = (\S+)/){
      } elsif (m/expire = (\S+)/){
      } elsif (m/minumun ttl = (\S+)/){
      }
    } elsif ($auth == 1) {
```

```
      if (m/^(\S+)\s+nameserver = (\S+)/){
      } elsif (m/(\S+)\s+internet address = (\S+)/ && $auth){
        push @nslist, $1 . "," . $2
            if (0 == grep(/,$2$/, @nslist));
      } else {
      }
    } elsif ($auth=-1) {
      if (m/^(\S+)\s+nameserver = (\S+)/){
      } elsif (m/(\S+)\s+internet address = (\S+)/){
        push @nslist, $1 . "," . $2
            if (0 == grep(/,$2$/, @nslist));
      } else {
      }
    }
  }
  close NSL;
  return @nslist;
}

sub min {
  my($min) = $_[0];
  for(@_) {
    $min = $_ if ($min > $_);
  }
  return $min;
}
sub max {
  my($max) = $_[0];
  for(@_) {
    $max = $_ if ($max < $_);
  }
  return $max;
}
sub avg {
  my($avg) = 0;
  if ($#_ == -1) {
    print STDERR "Average from empty array?\n";
    return 0;
  }
  for(@_) {
    $avg += $_;
  }
  return $avg / ($#_+1);
}
sub wavg {
  my(@val) = shift;
  my(@wei) = shift;
  my($avg) = 0;
  my($div) = &sum(@wei);
  for(@val) {
    $avg += $_ * shift @wei;
  }
  return $avg / ($div);
}
sub sum {
  my($sum) = 0;
  for(@_) {
    $sum += $_;
  }
  return $sum;
}

1;
```