Aalto-yliopisto
Sähkötekniikan korkeakoulu
Tietoliikenne- ja tietoverkkotekniikan laitos

Jarno Mikael Yliluoma

Tietoverkon liikenteen monitorointijärjestelmä hyödyntäen useita analysointimenetelmiä

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi
diplomi-insinöörin tutkintoa varten Espoossa 30.5.2011

25.5.2011

Työn valvoja:

TkT Jyri Hämäläinen

Työn ohjaaja:

TkL Markus Peuhkuri

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Jarno Yliluoma

Työn nimi: Tietoverkon liikenteen monitorointijärjestelmä hyödyntäen useita analysointimenetelmiä

Päivämäärä: 30.5.2011 Kieli: Suomi Sivumäärä: 4+68

Tietoliikenne- ja tietoverkkotekniikan laitos
Professuuri: Tietoliikennetekniikka Koodi: S-72

Valvoja: TkT Jyri Hämäläinen

Ohjaajat: TkL Markus Peuhkuri

Tämä työ käsittelee verkkoliikenteen analysointia usealla menetelmällä samanaikaisesti yhden mittapisteen kautta. Työssä on toteutettu yksinkertainen hajautettu reaaliaikainen tietoverkon liikenteen mittausjärjestelmä, jonka on tarkoitus toimia kehitysalustana jatkotutkimukselle. Sitä varten on kirjallisuusosassa luotu katsaus analysoitavaan liikenteeseen ja sen taustoihin ja tutustuttu mittausjärjestelmän malliin sekä toteutuksen kriteereihin. Tämän pohjalta on luotu suunnitelmat ja toteutus mittajärjestelmästä. Sillä on tutkittu annettua liikennejälkeä runkoverkosta.

Johtopäätöksenä yksinkertainen hajautettu reaaliaikainen tietoverkon liikenteen mittausjärjestelmä, jolla voidaan mitata tietoverkon liikennettä ja analysoida sitä käyttäen monia erilaisia menetelmiä. Analysoidusta liikenteestä havaittiin HTTP:llä siirrettyjen tiedostojen jakautuminen selkeästi kahteen eri ryhmään: pieniin tiedostoihin, jotka muodostavat lukumääräisesti suurimman osan ja muutamaan isoon tiedostoon, joista muodostuu suurin osa liikennemäärästä.

Avainsanat: tietoverkko, verkkoliikenne, mittaus, verkkoliikenteen mittausjärjestelmä

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Jarno Yliluoma

Title: Tietoverkon liikenteen monitorointijärjestelmä hyödyntäen useita analysointimenetelmiä

Date: 30.5.2011 Language: Finnish Pages: 4+68

Department of Communications and Networking
Professorship: Communications Engineering Koodi: S-72

Supervisor: Prof. Jyri Hämäläinen

Instructor: Lic.Sc. (Tech.)Markus Peuhkuri

Focus of this work is to analyze network traffic with multiple different methods with a single measuring point. A simple real-time distributed network traffic measurement system was created as the realization of this concept. System also works as a framework for further development.

A background study was done on the models of a measurement system and the criteria of an implementation of a such system. Some research was done on the traffic and parts of it where it had relevance with the system.

Based on the research plans for a measurement system were created. A measurement system was implemented according to these plans. System was used to analyze selected network traffic trail from core network.

It is plausible to create a simple distributed real-time measurement system to monitor traffic and analyze it using multiple methods. Files transferred with HTTP have a dualistic nature: they are divided into two groups. First one consist relatively small files and almost all files fall into this category. Second is very large files. There are few of them, but they make up most of the traffic (per byte).

Keywords: data network, network traffic, measurement, network traffic measurement system, sniffer

Alkusanat

Tahdon kiittää työn valvojaa TkT Jyri Hämäläistä ja ohjaajaa TkL Markus Peuhkuria. He ovat uskoneet työhön silloinkin, kun minä en ja antaneet apua silloin, kun sitä on tarvittu. He ovat kestäneet myös venyneet aikataulut ja viime hetken pyrähdykset.

Haluan kiittää myös Teknillistä korkeakoulua, nykyistä Aalto-yliopistoa, Valmistumisen Tuki 2010 ohjelmasta ja sen jälkeisistä myönnytyksistä valmistumisen edistämiseen.

Erityisen kiitoksen ansaitsevat rakkaat vanhempani, Sirkku ja Jouni, jotka laittoivat minut tälle tielle ostamalla minulle tietokoneen ennakkoon syntymäpäivälahjaksi vuonna 1986. Siitä asti ovat he tukeneet minua matkani jokaisella askeleella. En voisi heiltä enempää toivoa.

Kiitän myös isoäitiäni Mammaa. Hän on, minun lisäkseni, hoitanut ja auttanut kasvamaan kriittistä ajattelua, ongelmanratkaisukykyä ja avaraa maailmankuvaa eli kaikkia niitä ominaisuuksia, jotka katsotaan olevan eduksi teknisluonnontieteellisellä alla. En olisi kirjoittamassa tätä ilman sinua.

Viimeisenä mainitsen opiskelutoverini ja ystäväni; örkit ja muut humanoidit. Heille kuuluu kyseenalainen kunnia siitä, että ovat tehneet opiskelijaelämästäni niin värikästä ja hauskaa, että tuskin tohdin valmistua.

Espoossa 30.5.2011,

Jarno Yliluoma

Sisällysluettelo

Symbolit ja käsitteet

Käsitteet

API	Application Programming Interface
ARP	Address Resolution Protocol
ARPA	The Advanced Research Projects Agency
ARPANET	Advanced Research Projects Agency NETwork
CIDR	Classless Inter-Domain Routing
DNS	Domain Name Server
DoD	Department of Defence
FUNET	Finnish university network
GNU	GNU is not Unix
HTTP	HyperText Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IDS	Intrusion detection system
IIS	Internet Information Services, Microsoftin HTTP palvelin
IP	Internet Protocol
MIME	Multimedia Internet Message Extensions
NCP	Network Control Protocol
NTP	Network time protocol
NMS	Network Management System
NWG	Network Working Group
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
UCLA	University of California, Los Angeles
UDP	Universal Datagram Protocol
URI	Uniform Resource Identifier
VLAN	Virtual Local Area Networks
WWW	World Wide Web

1. Johdanto

Tässä työssä luodaan tietoverkon monitorointi- eli mittausjärjestelmä ja käytetään sitä erittelemään annettua liikennettä useilla analysointimenetelmillä. Kirjallisen osuuden sisältö on seuraava:

Luku 2, Teoreettinen tausta, aloitetaan tarkastelemalla työn taustaa tietoverkkojen ja niiden mittauksen kehittymisen näkökulmasta. Luvussa käydään myös läpi protokollien viitemalleja sekä esitellään muutamia työn kannalta keskeisiä protokollia. Luvussa tutustutaan mittausjärjestelmään, sen teoreettiseen malliin, toteutuksen kriteereihin ja teknisiin toteutuksiin. Läpi käydään myös työn kannalta oleellisia tilastollisen analyysin muotoja.

Luku 3, Mittausjärjestelmä ja sen tekninen toteutus, käsittelee työssä luotua mittausjärjestelmää. Luku aloitetaan käymällä läpi suunnitteluperiaatteita joiden mukaan järjestelmä on luotu ja tutkitaan tehtyjen valintojen suhdetta edellisessä luvussa esitettyihin teorioihin ja todetaan mittausjärjestelmän mittauksen kohteet.

Mittausjärjestelmän arkkitehtuuri esitetään luvussa 3. Se käydään läpi yleisesti ja erikseen tarkemmin jokaisen komponentin osalta. Esitetään myös keskeiset työkalut, joita on käytetty mittausjärjestelmässä.

Luvussa 4 esitellään mittausjärjestelmän antamia tuloksia. Järjestelmää käytettiin liikennetallenteen analysointiin ja sen analysoinnin tulokset esitellään tässä luvussa.

Johtopäätökset ovat luvussa 5. Ne jakautuvat kahteen osaan: ensimmäisessä käsitellään mittausjärjestelmään liittyviä asioita suhteessa teoreettisen taustaan ja käytännön toteutukseen. Lisäksi pohditaan mittausjärjestelmän jatkokehitystä ja käyttömahdollisuuksia. Toisessa osassa keskitytään liikennetallenteen analyysin tuloksiin ja esitetään mahdollisia jatkotutkimuskohteita.

Kuudes ja viimeinen luku, yhteenveto, käsittelee tämän työn keskeisiä teemoja tiivistetyssä muodossa ja se soveltuu pikaiseen työhön tutustumiseen.

Kuvaus tämän diplomityön kirjoittamisen prosessista ja siinä käytetyistä työkaluista sekä menetelmistä on toteutettu VaTu (Valmistumisen Tuki) projektin yhteydessä kandidaatintyönä [1] ja on saatavina elektronisena Aalto-yliopiston kirjastosta.

2. Teoreettinen tausta

2.1. Historiallinen tausta

Tietoverkkojen, kuten me ne tunnemme, peruskivinä voidaan pitää Leonard Kleinrockin vuonna 1961 kirjoittamaa julkaisua [2] pakettikytkennän teoriasta ja sitä vuonna 1964 seurannutta kirjaa [3]. Tämän teoreettisen työn avulla Kleinrock pystyi vakuuttamaan kollegoitansa ja seuraajiansa ARPA (The Advanced Research Projects Agency) pakettikytkentäisen tiedonsiirron käyttökelpoisuudesta piirikytkentäiseen verrattuna. [4]

Tietokoneiden välistä viestintää käytännön näkökulmasta lähestyivät Lawrence Roberts ja Thomas Merrill. Vuonna 1965 he yhdistivät kaksi tietokonetta käyttäen puhelinverkkoyhteyttä ja siten loivat ensimmäisen laaja-alaisen tietokoneverkon. [5] Koe osoitti kaksi asiaa: ensinnäkin yhteiskäyttöiset tietokoneet toimivat hyvin yhteen liitettyinä - tietoa voidaan hakea ja ohjelmia suorittaa tarpeen mukaan toisella tietokoneella. Toinen ja tietoverkkojen kannalta oleellisempi löydös vahvisti sen, että piirikytkentäinen puhelinverkko on riittämätön tietokoneiden väliseen viestintään; tämä vahvisti Kleinrockin teoreettisen työn. [4]

Näiden töiden pohjalta ja osittain samojen tekijöiden toimesta aloitettiin ARPAn rahoittama tietokoneverkkojen tutkimus, joka johti nopeasti ARPANET (Advanced Research Projects Agency NETwork) nimisen projektin, joka tähtäsi tietokoneverkon luomiseen, syntyyn. Vastaavanlaista työtä tehtiin samaan aikaan myös muissa tutkimuslaitoksissa Yhdysvalloissa ja Iso-Britanniassa. Muiden tutkimusten merkittävimmät vaikutukset projektiin olivat termin "paketti" (*packet)* käyttö, linjanopeuden nostaminen 50kbps asti ja väärinymmärrys, joka on johtanut urbaaniin legendaan ARPANETin suunnittelemisesta ydinsodan kestäväksi. [4]

Elokuussa 1968 projekti oli saanut määritettyä rakenteen ja määreet ARPANET-verkkoa varten. Tällöin käynnistettiin hankintakilpailu pakettikytkinten, jotka ovat nykyisten verkkokorttien ja kytkinten välimuotoja, suunnittelusta. Sen voitti Frank Heartin johtama ryhmä, joka yhteistyössä Bob Kahnin kanssa oli merkittävässä roolissa verkkoarkkitehtuurin suunnittelussa. Samaan aikaan Roberts työskenteli yhdessä Network Analysis Corporation kanssa verkkotopologian suunnittelun ja optimoinnin parissa ja Kleinrock tutkimusryhmänsä kanssa keskittyi verkkomittausjärjestelmän valmistamiseen. [4]

Vuonna 1969 pystytettiin varsinainen ARPANET-verkko. Rakentaminen aloitettiin oikeutetusti Kleinrockin kotiyliopistosta UCLAsta ja vuoden loppuun mennessä oli verkkoon liitetty yhteensä neljä konetta. Jo näin pienessä verkossa ja historian valossa aikaisessa vaiheessa tietoverkkotutkimus keskittyi tietoverkkoon ja sen käyttötapoihin. Vastaavanlainen jako on edelleen olemassa. [4]

Loppuvuodesta 1970 NWG (Network Working Group) julkaisi S. Crockerin toimesta ensimmäisen päätelaitteiden välisen protokollan NCPn (Network Control Protocol).

Sen päälle käyttäjien oli mahdollista rakentaa omia sovelluksiaan, joista merkittävimmäksi muodostui sähköinen posti: electronic mail, email. Se tuotti merkittävimmän osan verkon ja sen seuraajien liikenteestä yli seuraavan kymmenen vuoden ajan. [4]

Idean avoimesta verkkoarkkitehtuurista esitti Bob Kahn tultuaan ARPAn töihin vuonna 1972. Tähän asti verkot oli liitetty toisiinsa piirikytkentäisesti. Pakettikytkentäisyyden myötä mahdollisuudet verkkojen yhdistämiseen kasvoivat. Tyypillinen ratkaisu oli, että verkko oli toiselle alisteinen ja toimi siinä kuin yksittäinen päätelaite. Avoimen arkkitehtuurin ajatus oli, että jokainen verkko voidaan suunnitella itsenäisesti ja se voi tarjota palveluita käyttäjilleen ja/tai muille verkoille, joihin se on liittynyt. [4]

Tämän ajatuksen pohjalta Kahn kehitti uuden protokollan avoimen verkkoarkkitehtuurin tarpeisiin. Keskeisinä ajatuksina arkkitehtuurin ja protokollan luonnissa oli, että [4]

- keskenään kytkettyihin verkkoihin ei tarvitse tehdä sisäisiä muutoksia Internetiin liittymisen yhteydessä.

- tietoliikenne tapahtuu parhaan yrityksen mukaan: mitään lupauksia palvelun tasosta ei anneta. Mikäli paketti katoaa matkalla se lähetetään uudelleen alkuperäisen lähettäjän toimesta.

- "mustat laatikot" yhdistävät verkkoja. Ne eivät ota kantaa liikennevoihin ja käsittelevät liikennettä paketti kerrallaan. Myöhemmin tulemme tuntemaan ne reitittiminä ja yhteyskäytävinä.

- verkossa ei ole keskitettyä hallintaa.

Näiden suuntaviivojen lisäksi protokollan tekemisessä tuli ottaa huomioon näkökulmia, jotka olivat tulleet tietoon aikaisemmin tai kehitystyön aikana: [4]

- Pakettien katoaminen ei saa aiheuttaa tietoliikenteen katkeamista. Algoritmit suunnitellaan s.e. puuttuvat paketit lähetetään uudelleen alkuperäisen lähteen toimesta.

- Päätelaitteet voivat päättää pakettia lähettäessään reitin, mikäli välissä oleva verkko sitä tukee.

- Yhdyskäytävät osaavat välittää paketteja eteenpäin paketissa itsessään olevien tietojen ja yhdyskäytävän itsensä asetusten mukaisesti.

- Tarve julkisille ja universaaleille osoitteille.

- Yhteensopivuus eri käyttöjärjestelmien välillä.

Työn tulos oli jotain, mitä sittemmin kutsuttiin TCP:ksi (Transmission Control Protocol). Vaikka tarkoitus oli tehdä yksi protokolla, joka tarjoaa useita erilaisia siirtopalveluita verkon yli, käytännön toteutus mahdollisti vain virtuaaliset piirikytkennät, päätelaitteiden välisen luotettavan bittiputken. [4]

Käytäntö kuitenkin osoitti, että on olemassa sovelluksia, joille TCP:n tarjoama palvelu ei ollut riittävää. Protokolla jaettiin kahtia: IP (Internet Protocol) vastaa pakettien osoittamisesta ja reitityksestä ja erillinen TCP, joka huolehtii vuonkäsittelystä, virheenkorjaamisesta ja virtuaalisista piireistä. Tämä on myös tuntemamme TCP/IP. [4]

TCP:n rinnalle kehitettiin myös vaihtoehtoinen uusi protokolla UDP (Universal Datagram Protocol), jota käyttämällä on mahdollista käyttää vain IP:n tarjoamia palveluita ja ohittaa TCP:n vikasietoisuuden tarjoavat mekanismit. [4]

Alkuperäisestä NCP:stä siirryttiin hallitusti TCP/IP:n käyttöön ARPANET:issä 1.1.1983. Siirtymä oli suhteellisen kivuton, koska verkkoyhteisö oli ehtinyt suunnitella sitä usean vuoden ajan. [4]

Ajan myötä yhä enemmän verkkoja liittyi toisiinsa ja ARPANETistä tuli vain osa Internetiä. Koon kasvaessa huomattiin ongelmia teknisten ratkaisujen skaalautuvuudessa. Alkuperäisessä suunnitelmassa IP-osoitteet oli jaettu erikokoisiin verkkoihin. Verkkoja oli kolmea tasoa sen mukaan kuinka monta ensimmäistä oktettia IP-osoitteesta on varattu verkolle. Osoiteavaruus oli jaettu näille erikokoisille verkoille valistuneen arvauksen mukaan. [4]

Arvaus osui väärään ja etenkin pienempien verkkojen tarve ylitti niille varatun osoiteavaruuden. Tämän ongelman ratkaisuun kehitettiin CIDR (Classless Inter-Domain Routing), jonka ideana on se, että verkon ositteelle voi varata haluamansa määrän bittejä IP-osoitteen alusta aliverkkopeitteellä. Vastaavanlaisia parannuksia on tehty myös nimenselvitykseen, jossa paikallisesti ylläpidetyn listan korvasi nimipalvelu (DNS, Domain Name Server), ja reititykseen, jossa jokainen reititin ei enää tiennyt verkon kokonaiskuvaa vaan selvitti paketin seuraavan yhteyskäytävän naapureiltansa reititysprotokollia käyttäen. [4]

2.2. Liikenne

2.2.1. Viitemallit

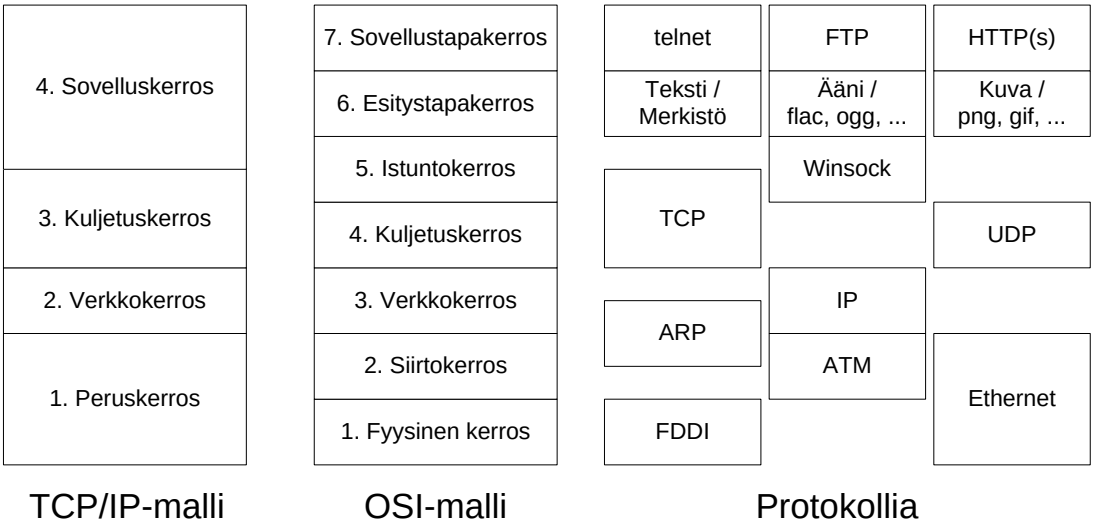
Viitemalleilla pyritään välittämään kuva Internetin teknisestä rakenteesta esittämällä eri protokollien roolit ja niiden väliset suhteet. Mallit voi myös mieltää verkon tekniseksi läpileikkaukseksi: mallit ovat kerroksittaisia alemman kerroksen aina tarjotessa palveluita ylemmälle kerrokselle.

Yleisesti käytössä on viitemalleista kaksi: TCP/IP-malli [6], jota kutsutaan myös DoD-malliksi (Department of Defence) taustansa vuoksi, ja OSI-malli [7] (Open Systems Interconnection). Ensimmäisen kuvaa protokollapinon osien välistä suhdetta sekä sen liittymistä järjestelmiin ja jälkimmäinen on tarkoituksenmukaisesti tehty eheäksi standardiksi.

Nämä aiheen erilaiset lähestymistavat ovat johtaneet mielenkiintoiseen tilanteeseen: OSI-mallia käytetään yleisesti teoreettisessa tarkastelussa ja koulussa opetusmallina, mutta sitä vastaavaa protokollapinoa ei ole tehty. TCP/IP-mallilla on päinvastainen tilanne: Protokollat, joita mallilla kuvataan, ovat Internetin peruskivi, mutta mallia ei

juuri koskaan käytetä muussa kontekstissa kuin kuvaamaan näitä protokollia ja niiden suhdetta alla olevaan verkkoon ja protokollien varaan rakennettuihin palveluihin.

Kuvassa 1 on esitetty molemmat viitemallit suhteessa toisiinsa sekä joitain protokollia ja kuinka ne sijoittuvat malleihin. Viitemallit on esitetty siten, että ylimpänä on abstraktein kerros. Tietoa verkkoon lähetettäessä käydään malli ylhäältä alas ja vastaavasti vastaanotettaessa alhaalta ylös.



Kuva 1: TCP/IP- ja OSI-viitemallit sekä joitain protokollia suhteessa niihin.

TCP/IP-mallissa on neljä kerrosta. Alin, peruskerros, on epätarkka ja kuvaa pääasiallisesti erilaisia verkkoteknologioita, jotka pystyvät välittämään tietoliikennepaketin linkin yli. Toinen, verkkokerros, vastaa siitä, että viesti välitetään verkossa kahden päätelaitteen välillä. Tämä vastaa tietoliikennepaketin välittämisestä s.e. se toimitetaan verkossa olevalta päätelaitteelta toiselle, mahdollisesti yhden tai useamman välivaiheen yli parhaan kyvyn mukaisesti. Käytännössä tämä kerros kuvaa IP-protokollan tehtävän. [6]

Vastaavasti TCP/IP-mallissa kuljetuskerroksen tehtävä on muodostaa päästä päähän kulkeva yhteys. Karuimmillaan tämän tekee UDP (Universal Datagram Protocol), joka käytännössä tarjoaa kauttansa ylemmille tasoille lähes suoran IP:n palveluiden käytön. Tämän lisäksi se tarjoaa ainoastaan porttien avulla tapahtuvan multipleksoinnin ja tarkistussumman datan oikeuden takaamiseen. Täydellisemmän implementaation tästä kerroksesta tarjoaa TCP. [6]

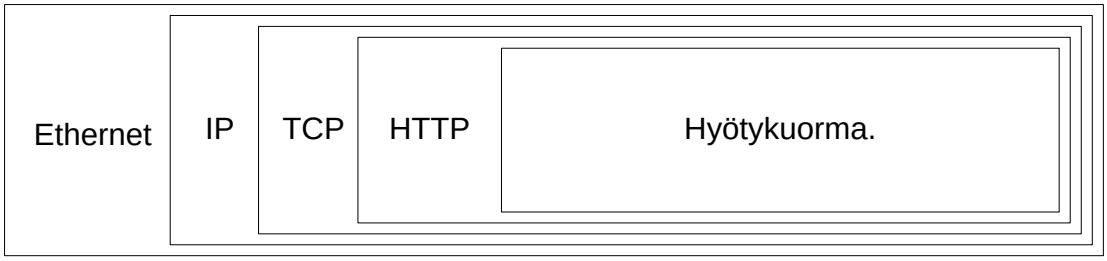
Ylin, sovelluskerros, on peruskerroksen kaltainen. Se kuvaa keskeisien protokollien päälle rakennettuja sovelluksia sen tarkemmin ottamatta kantaa minkälaisia nämä sovellukset ovat. Se, että ensimmäiselle ja neljännelle kerrokselle sijoittuu lukuisia protokollia ja tekniikoita ja toiselle ja kolmannelle vain muutama, on antanut viitemallille lempinimeksi tiimalasimalli.

OSI-mallin ensimmäinen kerros, fyysinen kerros, on niitä keinoja, joilla voidaan kuljettaa bittejä siirtotien yli. Tällaisia keinoja ovat muun muassa valokuitukaapeli ja radiotie lähettimineen ja vastaanottimineen. Siirtokerros sen yläpuolella vastaa tiedonsiirrosta kahden noodin välillä eli käyttää alemman kerroksen bittiputkea

sanomien lähettämiseen ja vastaa niiden perillemenosta. Nämä kaksi alinta kerrosta yhdessä vastaavat TCP/IP-mallin peruskerrosta. [7]

Mallissa kolmantena on verkkokerros, joka on vastaava TCP/IP-mallin kanssa. Sen kuljetuskerros on kuitenkin OSI-mallissa jakautunut käytännössä kahdeksi. OSI-mallin kuljetuskerros vastaa vain luotettavasta päästä päähän tiedonsiirrosta. Luotettava tässä tapauksessa ei välttämättä tarkoita sitä, että tämän kerroksen protokolla korjaisi verkkokerroksen virheitä, kuten TCP tekee vaan, että siirtyneen datan oikeellisuudesta pidetään huolta ja vähintään virheet ilmoitetaan, kuten UDP tekee. Istuntokerroksessa, joka on OSI-mallin viides kerros, osallistuvat tahot organisoivat ja synkronisoivat tiedonsiirtonsa. Käytännössä siis ne TCP:n toiminnallisuudet, jotka ylittävät UDP:n, kuuluvat tähän kerrokseen. [7]

OSI-viitemallin kuudes ja seitsemäs kerros vastaavat TCP/IP-mallin sovelluskerrosta. Esityskerros pitää sisällään yleiset tiedon esitystavat ja menetelmät sovelluksen sisäisten viestien tallentamiseen. Sovelluskerroksessa ovat ne protokollat, joita päätelaitteissa ajettavat ohjelmat käyttävät toisilleen viestintään, esimerkiksi selaimen ja www-palvelimen välinen HTTP (HyperText Transfer Protocol).



Kuva 2: Esimerkki tyypillisestä tietoliikennepaketista

Viitemallien kerroksittaisen rakenteen suhde tosimaailmaan selviää, kun tarkastellaan tyypillistä tietoliikennepakettia. Sellainen on esitetty kuvassa 2. Esimerkkinä on tyypillisessä työasemaympäristössä www-selailun yhteydessä liikkuva tietoliikenne paketti. Alimman tason protokollan, Ethernet; kerrokset 1 ja 2, kehyksen sisään on asetettu välitettäväksi seuraavan tason protokollan, IP; kerros 3, kehys, jonka hyötykuormassa on TCP:n, kerros 4 ja 5, kehys, joka puolestaan kantaa ylemmän tason, HTTP; kerrokset 6 ja 7, kuormaa. Lopulta HTTP kuljettaa lopullista hyötykuormaa, esimerkissä verkkosivua tai sen osaa.

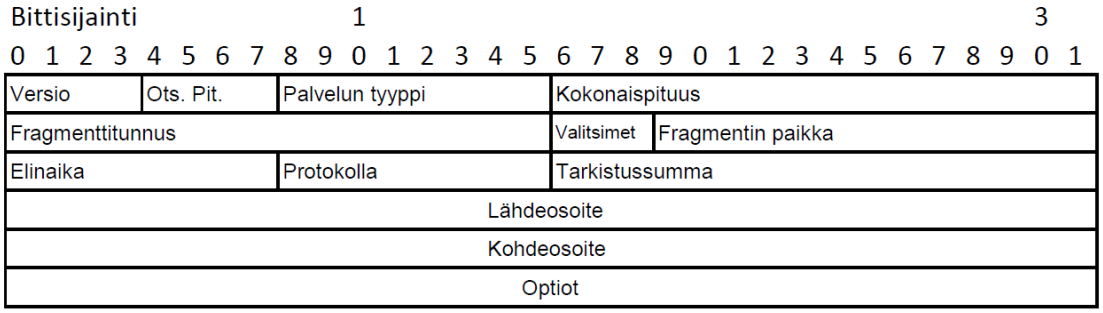
Rakenne myös liittyy em. tapaan käydä läpi mallia ylhäältä alas lähetettäessä ja alhaalta ylös vastaanotettaessa: lähettäjä luo ensin ylimmän tason kehyksen, josta tulee alemman tason hyötykuorma. Vastaavasti vastaanottaja aloittaa purkamisen ulommaisesta, alimmasta, kerroksesta.

Mallit eivät kuitenkaan täydellisesti kuvaa todellista tilannetta ja käytössä olevat protokollat eivät vastaa täysin tiettyjä kerroksia. Esimerkiksi ARP:n (Address Resolution Protocol) avulla voidaan hahmottaa siirtokerroksen (tai peruskerroksen) ja verkkokerroksen osoitteiden välisiä suhteita. Se ei varsinaisesti kuulu kumpaankaan kerrokseen vaikka sisältää toiminnallisuutta molemmista. Muut protokollat saattavat kuulua useampaan kerrokseen, kuten Ethernet, joka kattaa OSI-mallin fyysisen ja siirtokerroksen.

2.2.2. IP - Verkkokerros

IP (Internet protocol) on suunniteltu tarjoamaan tavan lähettää viestejä, datagrammeja, kahden (lähde ja kohde) päätelaitteen välillä, joita määrittää kiinteän mittainen osoitekenttä. IP myös mahdollistaa sen, että suurimman mahdollisen datagrammin koko voi vaihdella reitin varrella. Alkuperäinen paketti voidaan sirpaloida ja eheyttää tarvittaessa. [8] Kuten aikaisemminkin on mainittu, toteuttaa IP itseoikeutetusti TCP/IP viitemallin toisen kerroksen (verkko) ja mallien yhteneväisyyden mukaan samoin myös OSI-mallin verkkokerroksen.

IP kehyksen, paketin, aloittaa otsikko, joka sisältää protokollan toimintaan liittyvät tiedot. Versiossa neljä se on määrämuotoinen ja määrätyn mittainen. Otsikon muoto on esitetty kuvassa 3. Siinä otsikko on jaettu 4 tavun, 32 bitin, riveihin. Tämä on myös käytännön syistä, mm. 8, 16 ja 32 bittiset datarakenteet C-ohjelmointikielessä, tyypillisin tapa esittää otsikko.



Kuva 3: IP otsikko

Otsikko alkaa **versionumerolla**. Tässä tapauksessa se on 4. Kentän mukaan voidaan paketti olla käsittelemättä, jos versio on sellainen, jota laite ei tue. [8]

Otsikon pituus kertoo otsikon pituuden 32-bitin kerrannaisena. Pakolliset kentät vievät 160-bittiä, joten arvon täytyy olla vähintään viisi. Siten kenttä määrää myös viimeisen kentän (optiot) koon. [8]

Palvelun tyyppi sisältää useita erilaisia valitsimia, jotka vaikuttavat verkon tarjoaman palvelun tyyppiin. Oleellisesti kenttä jakautuu kolmen bitin kokoiseen palvelun tärkeysjärjestyksen määräävään kenttään ja kolmeen bitin kokoiseen valitsimeen, joilla voidaan määrätä kolmikantainen valinta viiveen, luotettavuuden ja välityskyvyn kanssa. Nämä valinnat opastavat muita verkkolaitteita paketin käsittelyssä. On suositeltavaa, ettei yli 576 tavun paketteja käytettäisi. Tällöin on ajateltu käytettävän 512 tavua hyötykuormalle ja 64 tavua otsikoille. [8]

Kokonaispituus on ilmaistu tavujen kerrannaisena ja siihen lasketaan myös tämän otsikon pituus. [8]

Mikäli paketti sirpaloidaan käytettään **fragmenttitunnusta** tunnisteena merkitsemään se ryhmä (alkuperäinen paketti), jonka sirpale tämä paketti on. Vastaavasti **fragmentin paikka** osoittaa tämän sirpaleen paikan fragmenttitunnuksen nimeämässä ryhmässä. **Valitsimilla** voi määrätä, ettei tätä pakettia ei saa sirpaloida tai että tämä paketti on viimeinen sirpale. [8]

Elinaika kertoo kuinka kauan tämä paketti voi olla Internetissä. Aina kun paketti välitetään edelleen kentän arvoa vähennetään yhdellä. Kun se kohtaa nollan paketti tuhotaan. [8]

Protokolla määrää seuraavan tason otsikon protokollan. Protkollien vastaavuuttaa eri numeroihin pitää yllä Internet Assigned Numbers Authority (IANA). [8] [9]

Tarkistussumma lasketaan otsikosta takaamaan sen eheys. Jos otsikon tiedot, kuten elinaika, muuttuvat, pitää tarkistussumma laskea uudelleen.

Kohde- ja lähdeosoitteet ovat versio neljän mukaisia osoitteita. Pääasiallisesti näiden tietojen mukaan tehdään reitityspäätökset. [8]

Optiot on varattu protokollamäärityksen ulkopuolisten asioiden määrittämiseen, eikä kenttää ole pakko käyttää. [8]

Tässä esitetyn ja yleisesti käytössä olevan version neljä (IPv4) seuraaja on versio kuusi (IPv6). Siinä on otettu huomioon suurimmat ongelmat, joihin on törmätty IPv4:n kanssa. Merkittäviä uudistuksia on mm. osoitteen koon kasvattaminen 128 bittiin sekä otsikoiden muuttamisen. Perusotsikko on hyvin yksinkertainen, mutta tarvittavaa lisätoiminnallisuutta löytyy laajennusotsikoista, joita lisätään tarvittaessa useitakin erilaisia. Tämä antaa enemmän joustavuutta IPv4:n jäykkään malliin verrattuna ja myös vähentää liikenteen määrää. [10]

Siirtyminen IPv6 on tapahtunut hyvin hiljalleen verrattuna historialliseen yhden yön siirtymään IPv4:n ja TCP:n kanssa. Tosin tätä työtä kirjoitettaessa vapaat IPv4 osoitteet ovat loppuneet. [11] Tämä todennäköisesti tulee nopeuttamaan siirtymistä, kun ongelmat osoitteiden loppumisesta yleistyvät.

IPv6:ta ei käsitellä laajemmin tässä työssä.

2.2.3. TCP - Kuljetuskerros

TCP on suunniteltu takaamaan luotettava kaksisuuntainen tavuvuo, virtuaalinen piiri-kytkentä. TCP/IP-viitemallissa TCP sijoittuu määritelmällisesti kuljetuskerrokseen. OSI-mallin kanssa tilanne ei ole yhtä yksinkertainen. TCP vastaa kaikista kuljetuskerroksen tehtävistä, mutta, mm. yhteydenhallintansa vuoksi, toteuttaa joitain istuntokerrokselle kuuluvia tehtäviä.

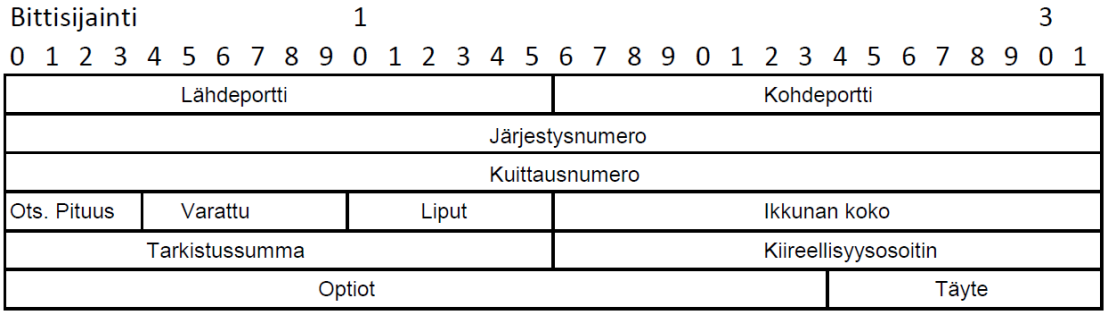
Yhteys luodaan isäntäkoneiden välisellä kolmivaiheisella kättelyllä (SYN – SYN/ACK – ACK). Tiedonsiirron aikana lähettäjä numeroi lähettämänsä paketit ja vastaanottaja kuittaa vastaanottamansa paketit ACK paketilla, jossa on vastaanotetun paketin sekvenssinumero. Yhteys suljetaan oikeaoppisesti FIN tai RST paketilla. Tämän perusmekaniikan avulla saavutetaan luetettava tiedonsiirto. Vaikka siirtotiessä olisikin virheitä, saadaan tavuvuo regeneroitua vastaanottavassa päässä oikein. Virhe voi johtua mm. väärästä saapumisjärjestyksestä, jota voidaan korjata sekvenssinumeroin ja saapumatta jääneet paketit lähetettää uudelleen. [12]

Luotettavien yhteyksien, virtuaalisten piirikytkentöjen, ohella TCP tarjoaa muitakin merkittäviä palveluita. Sekvenssinumeroiden käsittely mahdollistaa vuonkäsittelyn: uusi paketti lähetetään vasta, kun vastaanottaja on lähettänyt ACK-viestissään edellisen paketin sekvenssinumeron. Samanaikaisesti käytössä olevien sekvenssinumeroiden lukumäärää, ikkunaa, voidaan kasvattaa ja pienentää verkon tilan mukaan. [12]

Kahden isännän välinen tiedonsiirto ei rajoitu yhteen TCP yhteyteen - protokolla mahdollistaa multipleksoinnin. TCP:n osoitteet ovat 16 bitin kokoisia ja ne esitetään kymmenkantaisena kokonaislukuna. Niitä kutsutaan, erona IP osoitteisiin, porteiksi. Yhteydet muodostuvat porttiparien välille. Tällöin voi kahden isännän välillä olla jopa $2^{16} \times 2^{16}$ (noin 4,3 miljardia) samanaikaista TCP yhteyttä - käytännössä muut asiat rajoittavat yhteyksien maksimimäärän paljon pienemmäksi. [12]

Periaatteessa portit ovat keskenään tasa-arvoisia ja niiden käyttö riippuu implementaatiosta. Käytännössä portit on jaettu kolmeen osaan: tunnettuihin (1-1023), rekisteröityihin (1024-49151) ja dynaamisiin (49152-65535). Kahden ensimmäisen kategorian portit assosioidaan protokolliin. IANA on auktoriteetti ja pitää listaa assosiaatioista. Määrätyt porttinumerot ovat palvelinportteja, joissa kuuntelee tunnettuja palveluita ja joihin asiakas ottaa yhteyttä. Asiakasportti, josta otetaan yhteyttä, valitaan ideaalisesti dynaamisista porteista, mutta voi olla mielivaltainen asiakaskoneen toteutuksen mukaan. [13]

TCP otsikko seuraa paketissa IP:n otsikkoa. Kuvassa 4 on esitetty TCP:n otsikko samoin kuin IP:n aiemmin.



Kuva 4: TCP otsikko

Lähde- ja kohdeportti ovat otsikon 32 ensimmäistä bittiä. Lähdeportti on siinä mielessä merkittävä, että se osoittaa myös vastaanottajalle, minne vastaukset tulee lähettää. [12]

Järjestysnumero kertoo tämän paketin ensimmäisen tavun sijainnin tavuvuossa. Jos kyseessä on synkronisointiviesti tulee ensimmäisen varsinaisen dataoktetin sijainti olemaan annettu arvo +1. [12]

Jos kyseessä on kuittausviesti **kuittausnumero** ilmoittaa lähettäjälle seuraavan dataoktetin sijainnin, jonka vastaanottaja on valmis ottamaan vastaan. [12]

Otsikon pituus kertoo otsikon pituuden 32-bitin kerrannaisina. [12]

Liput määräävät minkälainen paketti on kyseessä. Synkronisointi- (SYN) ja lopetuslippuja (FIN) käytetään yhteyden oikeaoppiseen avaamiseen ja lopettamiseen. Uudelleenalustusta (RESET) käytetään ilmoittamaan yhteyden toiselle osapuolelle, että yhteyden tila on epämääräinen ja se halutaan aloittaa uudelleen. Lipuilla voi ilmoittaa myös kuorman olevan kiireellinen (URG) tai että tiedon siirto aloitetaan heti (PUSH). [12]

Ikkunan koko määrittää kuinka monta tavua vastaanottaja on valmis ottamaan vastaan kuittausnumerosta lähtien. [12]

Tarkistussumma lasketaan otsikosta, hyötykuormasta ja näennäisotsikosta, joka on koostettu edeltävän IP otsikon tiedoista. Laskennan aikana tarkistussummakenttä katsotaan sisältävän nollia. [12]

Jos aikaisemmin mainittu kiireellisyysvalitsin on käytössä, **kiireellisyysosoitin** osoittaa sen tavun sijainnin tavuvirrassa, joka on ensimmäinen kiireellisen datan jälkeen.

Optiot ovat vapaavalintaisia, mutta jos niiden pituus on alle 32-bittiä, pitää kenttää laventaa **täytteellä** lisäämällä nollia loppuun s.e. otsikon koko pysyy 32-bitillä jaollisena.

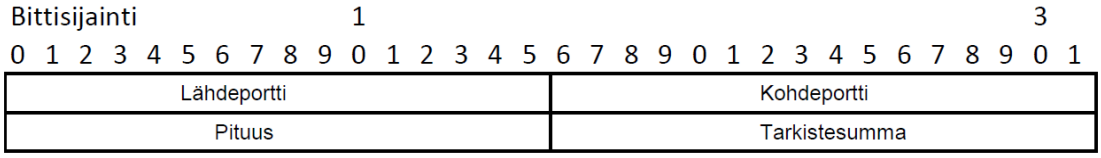
2.2.4. UDP – Yhteydetön vaihtoehto

TCP on erinomainen vaihtoehto luotettavaan tiedonsiirtoon. Ne mekanismit, joilla luotettavuus saavutetaan, tuottavat samalla ongelmia mm. reaaliaikaisuuden suhteen. Sen vuoksi TCP:n rinnalle luotiin toinen protokolla, jossa ei olisi näitä mekanismeja. Tällöin sallitaan mahdollisimman suora IP protokollan käyttö, viitemallin mukaisesti.

UDP, TCP:n rinnakkaisprotokollana, sijoittuu TCP/IP-mallissa kuljetuskerrokseen. Koska UDP ei tarjoa samalla lailla palveluita, sijoittuu se myös OSI-mallissa suoraan kuljetuskerrokseen.

Ainut palvelu, jonka UDP:ssä on kopioitu TCP:ltä on multipleksointi. Osoitteistus on vastaava kuin TCP:llä, mutta portit eivät kuitenkaan ole samoja. Tyypillisesti porttinumerot erotellaan toisistaan protokollan mukaan, esim. 80/tcp. Muut toiminnot, kuten virheenkorjaus ja vuonhallinta, jätetään tarkoituksella ylempien tasojen protokollien hoidettavaksi tarpeittensa mukaan. [14]

Näistä syistä UDP:n otsikko on varsin yksinkertainen. Se on esitetty IP:n ja TCP:n otsikoiden tavoin kuvassa 5. [14]



Kuva 5: UDP otsikko

Lähde- ja kohdeportit ovat multipleksointia ja palveluiden erittelyä varten, samoin kuin TCP:ssä. Erona on se, että lähdeportti ei ole pakollinen. Jos sen arvo on muu kuin nolla, voidaan olettaa, että lähettäjä odottaa vastausta siihen porttiin. [14]

Pituus on tavuina sekä otsikon (8 tavua) että kuorman koko. [14]

Tarkistesumma lasketaan hyötykuormasta, otsikosta ja näennäisotsikosta, joka on koostettu edeltävän IP otsikon tiedoista. Kuormaa laajennetaan laskennan ajaksi tarvittaessa nollilla, jotta sen koko tavuissa on parillinen kokonaisluku. [14]

2.2.5. HTTP - Sovelluskerros

HTTP on tarkoitettu sovellustason protokollaksi hajautetun, yhteiskäytöllisen hypermedian siirtoon. Täten se myös sijoittautuu molemmissa viitemalleissa ylimpään, sovelluskerrokseen. Se on myös protokolla WWW (World Wide Web) takana. [15]

Kun tavoitteena on datan siirto, on luontevaa, että HTTP käyttää TCP:tä kuljetuskerroksessa. OSI-mallin esitystapakerroksessa voidaan käyttää useita tapoja, kuten tekstiä ja erilaisia pakkausvaihtoehtoja. Asiakas ja palvelin sopivat käytettävästä esitystavasta.

HTTP perustuu pyyntö-vastaus-periaatteeseen: asiakas lähettää kyselyn palvelimelle, johon se saa vastauksen. TCP:n yhteydellisyys mahdollistaa toki viestien jakamisen useamman IP paketin välille, mikäli se on tarpeen, mutta itsessään protokolla on yhteydetön. Jokainen kysely käsitellään muista erillisenä. Sittemmin protokollaa on laajennettu tilalliseen suuntaan lisäämällä pieniä sanomia, evästeitä, joita asiakkaat ja palvelimet lähettävät toisilleen pyynnöissä ja vastauksissa. Siirto tapahtuu protokollamäärityksen puitteissa ja evästeiden hallinta jätetään ohjelmistojen käsiteltäväksi. [15] [16]

Protokolla on tekstipohjainen ja viestejä jäsennetään rivinvaihdoilla, joilla on merkitystä. Sekä pyynnön, että vastauksen yleinen rakenne on samanlainen: ensimmäisellä rivillä on varsinainen pyyntö tai vastaus. Sitä seuraa otsikkorivit. Viimeisenä on viestin runko, jos sellainen on.

```
GET /index.html HTTP/1.1
Host: www.google.fi
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0)
Gecko/20100101 Firefox/4.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fi-fi,fi;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Proxy-Connection: keep-alive
Cookie:
PREF=ID=696b2f1d84e64f49:U=549ac34669507c02:FF=0:TM=1263678084:LM=129
6131350:S=-weJ3UTGRZ5pwHjs;
```

Kuva 6: HTTP pyyntö

Kuvassa kuusi on esitetty esimerkki pyynnöstä. Ensimmäisenä ensimmäisellä rivillä on pyynnön metodi. Sitä seuraa kohteen URI (Uniform Resource Identifier) ja

käytettävä protokollan versio. Kentät erotellaan tyhjein, kuten välilyönnein ja sarkaimin. [15]

Yleisimmät metodit ovat tiedon noutoon tarkoitettu GET ja POST, jonka käyttötarkoitus on siirtää tietoa palvelimelle käsiteltäväksi. Molemmilla metodeilla on mahdollista välittää ylimääräistä informaatiota palvelimelle. GET:n tapauksessa se tehdään muuttamalla kohteen URI:a - ns. GET-parametrit. POST-metodilla siirretyt tiedot ovat viestin rungossa. Protokollamäärittelyssä on esitetty myös monia muita metodeja, mutta ne ovat harvemmin käytössä. [15]

Pyynnön jälkeen seuraa otsikot. Ne ovat tyypillisesti muotoa "otsikon nimi: arvo" rivin päättyessä oikeaoppiseen rivinvaihtoon. Asettelu on tarkemmin määritelty standardissa RFC 822 [17]. Host otsikkoa käytetään kertomaan palvelimelle, minkä palvelinnimen mukaan sivua on haettu. Tämä mahdollistaa useita eri nimillä toimivia palvelimia samalla IP osoitteella. User-agent kertoo käytetyn asiakasohjelmiston. Accept-alkuiset otsikot kertovat palvelimelle missä muodoissa asiakas pystyy käsittelemään vastauksen. Cookie-otsikon alla ovat evästeet, joista keskusteltiin aikaisemmin. Keep-alive ja proxy-connection ovat välimuistin hallintaan liittyviä ehdotuksia. [15]

Runkoa tässä viestissä ei ole.

```
HTTP/1.1 200 OK
Date: Sun, 24 Apr 2011 22:56:01 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Server: gws
X-XSS-Protection: 1; mode=block
Content-Length: 15609

<!doctype html><html><head><meta http-equiv="content-type"
content="text/html; charset=UTF-
8"><title>Google</title><script>window.google={kEI:"gaq0TZvFOI7GtAatg
NnuCw",kEXPI:"17259,17311,17315,23628,26849,27084,29050,29403,29418,2
9477,29681,29685,29715,29782,29813,29936",kCSI:{e:"17259,17311,17315
```

Kuva 7: HTTP vastaus

Kuvassa seitsemän on edellä kuvatun pyynnön vastaus. Se alkaa vastausrivillä. Ensimmäisenä on käytetty protokolla ja sen versio. Sitä seuraa tilakoodi ja sen englanninkielinen selite. Tilakoodi kertoo asiakkaalle pyynnön suorituksen lopputuloksen ja mahdollisesti ohjaa jatkotoimenpiteisiin. Koodin ensimmäinen numero kertoo koodin luokan. Tyypillisiä koodeja ovat 200 (OK), 302 (uudelleenohjaus) ja 404 (ei löytynyt). Protokollamäärittelyssä olevien koodien lisäksi eri valmistajat ovat tehneet laajennuksia, esimerkiksi IIS (Internet Information Services, Microsoftin HTTP palvelin) lisää joihinkin koodeihin desimaaleja. [15]

Vastausriviä jälkeen tulee otsikot. Niissä on tietoja palvelimesta ja ohjeita vastauksen tallentamiseen välimuistiin. Työn kannalta oleellisimpia otsikoita ovat Content-Type, joka ilmoittaa rungon sisällön MIME [18] (Multimedia Internet Message Extensions) tyypin, ja Content-Length, joka kertoo rungon pituuden tavuina. [15]

Otsikoiden jälkeen tulee viestin runko. Tässä tapauksessa se on google.fi:n etusivu html-muodossa.

2.2.6. PDU, Protocol Data Unit

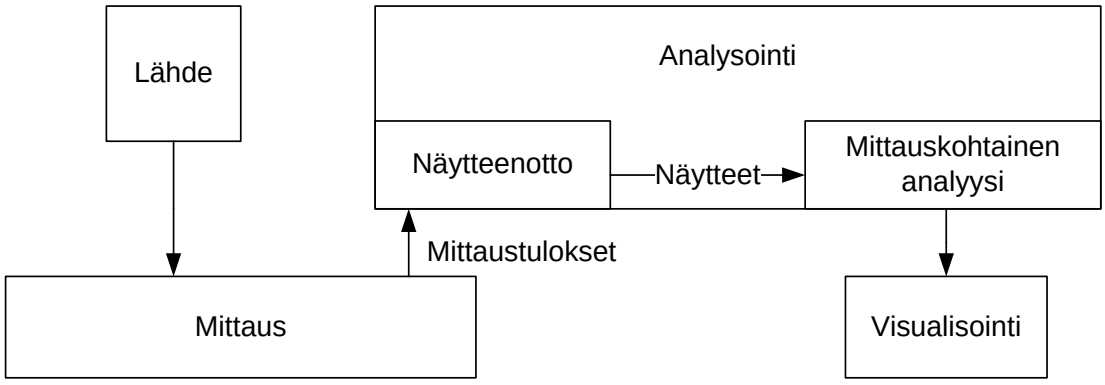
Eri protokollat käyttävät erilaisia nimityksiä protokollalla kahden osallistujan välillä välitettävästä viestistä. Tyypillisesti nimitykset ovat samoja tai samankaltaisia tietyllä viitemallin kerroksella. Tällaisia nimityksiä ovat mm. kehys (siirtoyhteyskerros, Ethernet), paketti (verkkokerros, IP) ja sanoma (sovelluskerros).

Nimitysten yleisnimi on Protocol Data Unit (PDU, protokollatietoyksikkö). Se on määritelmällisesti tietoyksikkö, joka on protokollassa määritelty ja joka sisältää protokollan tietokenttiä sekä mahdollisesti käyttäjän dataa. [19]

2.3. Mittausjärjestelmä

2.3.1. Mittausjärjestelmän malli

Väitöskirjassaan Arlos [20] esittää mallin, jossa mittausjärjestelmä koostuu neljästä osasta: Liikenteestä, mittauksesta, analyysistä ja visualisoinnista.



Kuva 8: Mittausjärjestelmän malli

Mittausjärjestelmämallin lähtökohta on lähde, joka luo mitattavan liikenteen. Aktiivisissa mittauksissa lähde on liikennegeneraattori. Sen tehtävä on luoda liikennettä annettujen parametrien mukaan. Liikennegeneraattori on mittapisteen kaltainen, mutta tapahtumien tallentamisen sijasta se luo niitä. Passiivisissa mittauksissa ei yleensä käytetä liikennegeneraattoreita. Sen sijaan lähteeksi voidaan mieltää joko verkko, jota mitataan ja jossa liikenne liikkuu, tai jokainen siihen liittynyt liikennettä tuottava päätelaite. [20]

Mittausosan tehtävä on vain ja ainoastaan mittausdatan kerääminen. Verkosta kerätään liikennettä PDU kerrallaan. Sitä voidaan suodattaa eli valita vain tiettyihin kriteereihin sopivat kohteet. Mittaosassa ei kuitenkaan tehdä minkään tason yhteen keräämistä tai parametrien eristämistä ja tallentamista. Mittaus voidaan tehdä verkkomallin kaikilla tasoilla aina fyysiseltä tasolta sovellustasolle. Jos mittaosaa ajatellaan mustana laatikkona, ottaa se vastaan verkkoliikennettä ja muuntaa ne mittaustuloksiksi välitettäväksi eteenpäin. Ne voivat olla vain abstrakti osa ohjelman prosessissa tai aivan konkreettisia, kuten tiedostoja tai tietokantoja. [20]

Mittausosa on myös ainut yhteinen osa aktiivisilla ja passiivisilla mittaustavoilla: molemmat keräävät verkosta PDU:ta. Jopa tekninen toteutus voi olla sama mittaustavasta riippumatta. [20]

Analysointi käsittää kaiken sen, mitä tehdään mittaustuloksille ennen niiden visualisointia. Analysointi voidaan jakaa kahteen osaan, näytteenottoon ja mittauskohtaiseen analyysiin. Näytteenotossa valitaan pienempi joukko edustamaan koko mittaustuloksia. Näytteet tai mittaustulokset analysoidaan lopulta mittauskohtaisella menetelmällä. Se voi olla mitä tahansa yksinkertaisesta (liikenteen kokonaismäärän laskeminen) hyvin monimutkaiseen (hyökkäysyritysten etsiminen HTTP pyynnöistä). Analysoinnin jälkeen tulokset ohjataan edelleen käytettäväksi visualisointiin. Analysoinnista keskustellaan enemmän luvussa 2.6. [20]

Visualisoinnin tehtävänä on esittää analysoinnin tulokset käyttäjälle. Tämä osa mittausjärjestelmän mallista on myös ainut, jonka kanssa loppukäyttäjä välittömästi on tekemisissä. Tuloksiin ja niistä tehtäviin johtopäätöksiin on mahdollista vaikuttaa visualisointitavoilla ja -menetelmillä. Sen vuoksi tämän osan menetelmien valintaan ja toteutukseen pitää kiinnittää erityistä huomiota. [20]

Visualisointi ei ole ainoastaan kuvaajia ja animaatioita. Graafista esitystä käytetään, kun esiteltävänä on suuri määrä tuloksia. Numeerinen esitystapa on sopiva muutamille merkitseville luvuille.

2.3.2. Mittausjärjestelmän toteutuksen kriteerit

Arlos mainitsee työssään [20] myös muutamia keskeisiä periaatteita, joita on hyvä noudattaa mittausjärjestelmää toteutettaessa. Ne ovat kustannustehokkuus, käytettävyys, tietoturvallisuus ja modulaarisuus.

Kustannustehokkuus on merkittävä tekijä reaalimaailmassa. Mittalaitteisto on kallista verrattuna normaaliin, työpöytäkäyttöön tarkoitettuun kuluttajatason tietotekniikkaan. Kalliin laitteiston käyttöaste tulee pitää korkealla ja silti välttää tilanne, jossa laitteisto on varattuna liian pitkään yksittäiselle käyttäjälle.

Käytettävyys on tärkeä tekijä mitä tahansa järjestelmää suunniteltaessa. Mittausjärjestelmässä pitää mittausten määrittäminen ja tulosten saaminen olla käyttäjän kannalta helppoa, huolimatta varsinaisen mittausprosessin kompleksisuudesta.

Tietoturvallisuus mittausjärjestelmässä kohdistuu etenkin näytteisiin ja mittaustuloksiin. On tietenkin selvää, että arkaluontoinen materiaali ei saa päätyä vääriin käsiin. Tässä tapauksella tietoturva keskittyy enemmän tietojen eheyteen. Prosessin lopputuloksen kannalta on oleellista, ettei mitattu tai analysoitava data ole muuttunut.

Modulaarisuus on ehkä tärkein suunnitteluparadigma mittausjärjestelmää tehdessä etenkin, jos kyseessä on hajautettu järjestelmä. Jokainen komponentti on siinä määrin itsenäinen, että niitä voidaan kehittää toisistaan riippumatta. Tämä myös tarkoittaa sitä, että komponentit ovat myös vaihdettavissa, jos esimerkiksi vaihdetaan linkkinopeutta suuremmaksi.

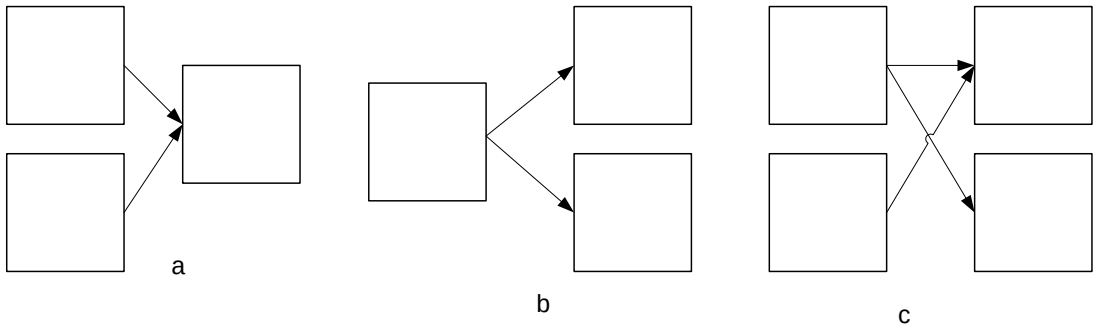
2.3.3. Hajautettu mittausjärjestelmä

Aiemmin esitetyn mittajärjestelmän mallin helposti mieltää olevan prosessi, joka pyörii vain yhdessä laitteessa etenkin, kun osa yleisesti käytössä olevista menetelmistä on sellaisia. Järjestelmää voidaan kuitenkin hajauttaa. Kaikki mallin yksittäiset osat (lähde, mittaus, analyysi ja visualisointi) voidaan hajauttaa. Tällöin yhdistävänä tekijänä ovat vuot osien välissä, joiden voi mieltää olevan tosiasiallisesti erilaisia jaettuja medioita.

Yksinkertaisin tapa hajauttaa järjestelmä on se, että jokainen eri osa on yksittäinen laite. Tässä lähestymistavassa prosessi on edelleen lineaarinen eli laitteet ovat liitetty toisiinsa peräjälkeen, mutta yksi osa ei enää varaa kaikkia resursseja yhteiseltä alustalta. Sen lisäksi resursseja vapautuu osa kerrallaan koko järjestelmän sijaan.

Toinen tapa lähestyä asiaa on järjestelmän osien järjestäminen rinnakkain. Mallissa olevia osia voi olla jokaista eri tyyppiä mielivaltainen määrä. Tietyn tyyppisten osien pitää olla aina toisistaan riippumattomia, jotta rinnakkaistaminen on mahdollista.

Kuvassa 9 on esitetty miten rinnakkaiset osat voivat liittyä seuraavaan osaan. Monta yhteen (a) tapauksessa useamman osan tietovuo tulee käsiteltäväksi samalle osalle. Esimerkkinä mittaus, jossa on kaksi mittapistettä ja jossa tutkitaan kulkeeko vai katoaako paketti matkalla tai kuinka paketin viive käyttäytyy kahden mittapisteen välillä. Yhdestä moneen (b) tilanteessa tietovuo toimitetaan useammalle seuraavan tason osalle. Mittausjärjestelmässä voi olla esimerkiksi useita erilaisia analysaattoreita, joille tulee sama mitattu liikenne käsiteltäväksi; jokaiselle eri tavalla suodatettuna. Järjestelmässä voi olla useita viittauksia (c), jotka ovat tyypiltään yhdestä moneen tai monesta yhteen ja jotka ovat osittain päällekkäisiä: Sama tietovirta voidaan toimittaa useammalle seuraavan tason osalle ja osa voi ottaa vastaan useita tietovirtoja.



Kuva 9:Rinnakkaisliitoksia

Hajauttaminen poistaa tiettyjä pullonkauloja mittausjärjestelmässä. Merkittävin on resurssien määrän kasvaminen ja niiden jakautuminen mallin eri osien kesken. Yksinkertaisessa tapauksessa mittalaitteen malli kuvaa yhden laitteen. Jos järjestelmä hajautetaan pienempiin alajärjestelmiin, on yhden osan käyttöaika pienempi, joten sen käyttöasetetta voidaan nostaa. Toisaalta samanaikaiset ja toisistaan riippumattomat mittausprosessit voivat olla rinnakkaistettavissa. Esimerkiksi sama mittapiste voi tuottaa mittaustuloksia useammalle analysaattorille.

Hajautetusta järjestelmästä ei ole pelkästään hyötyä. Hajautetussa järjestelmässä kompleksisuus kasvaa. Järjestelmän rakenteesta riippuen voi vikaantuminen estää

kaiken käytön tai vaikuttaa vain yhteen tai useampaan prosessiin. Myös hallinnointi- ja ylläpitotoimet on monimutkaisempia verrattuna järjestelmään, jonka kaikki osat ovat samassa laitteistossa.

Konkreettisesti hajautetun järjestelmän ensimmäinen vastaan tuleva ongelma on se, että mallin osien väliset tietovoiden media vaihtuu hitaampaan ja epäluotettavampaan, esimerkiksi jaetusta massamuistista verkkoyhteyteen.

2.4. Mittaaminen

2.4.1. Aktiivinen mittaaminen

Aktiivinen mittaus perustuu toimenpiteen vastareaktion tarkkailuun: verkkoon tuotetaan liikennettä (toimenpide), joka käyttäytymistä tutkitaan (vaste). Klassinen käyttökohde on mitata kahden päätepisteen välistä paketin kiertoaikaa lähettämällä ICMP (Internet Control Message Protocol) kaiutuspyyntöjä ja analysoimalla vastauspakettien saapumiseen kuluva aika. Menetelmä tunnetaan yleisemmin käytetyn komennon nimellä, *ping*.

Aktiivisessa mittauksessa vasteen käyttäytymistä seurataan eri ympäristöissä tai eri tilanteissa. Aikaisempi esimerkki voitaisiin toistaa tasatunnein, jolloin saisi selville verkon siirtokykyä eri vuorokauden aikoina. Koska tutkimuksen kohteena on vaste, on aktiivinen mittaus verkon tarkasteluun. Verkossa kulkevasta liikenteestä se antaa tietoa vasta toissijaisesti: aktiivinen mittaus kertoo miten verkossa oleva liikenne vaikuttaa verkon tilaan. [20]

Varsinainen mittaaminen verkosta tapahtuu kytkemällä mittalaite kuten normaali päätelaite tai käyttämällä passiivisen mittaamisen menetelmiä. [20]

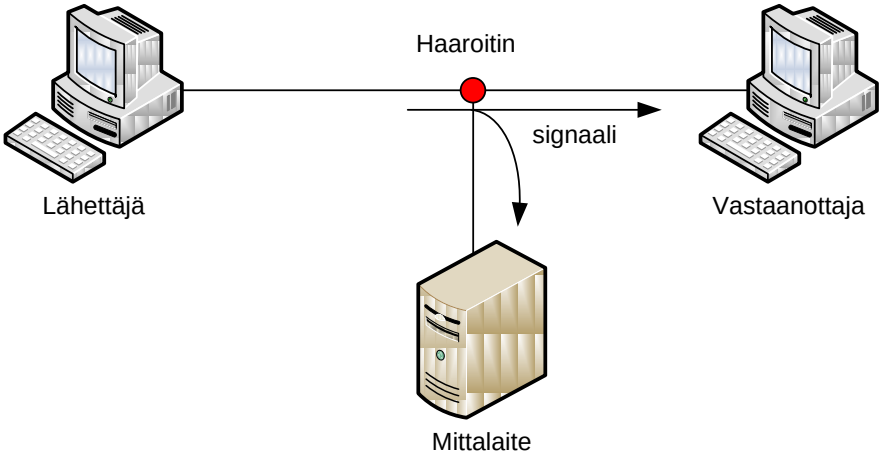
2.4.2. Passiivinen mittaaminen

Passiivisessa mittaamisessa seurataan verkossa kulkevaa liikennettä sitä muuttamatta. Siinä missä aktiivinen mittaaminen muuttaa verkon tilaa, muuttaa passiivinen mittaaminen verkkotopologiaa ainakin yhdellä viitemallin tasolla. Mittalaite tulee liittää verkkoon korkeintaan sillä viitemallin tasolla, jossa sijaitsee alimmat tutkivat kohteet. Liitos verkkoon voidaan tehdä mm. seuraavilla tavoilla:

Passiivinen kuuntelu tapahtuu verkkomallin fyysisessä kerroksessa. Se on luonteeltaan samanlaista kuin perinteinen analogisten puhelinlinjojen salakuuntelu. Linkki haaroitetaan s.e. signaali ohjataan haaroittimella myös mittalaitteelle.

Haaroittimet ovat passiivisia komponentteja, joten sellaisen käyttäminen ei muuta linkin viestiä, mutta aiheuttaa havaittavissa olevaa häiriötä ja vaimentumista. Passiivilaitteena haaroittimen luotettavuus on hyvä. Passiivisessa kuuntelussa, nimensä mukaisesti, ei ole mahdollisuutta vastavuoroiseen toimintaan verkon kanssa.

Kytkentäperiaate on esitetty kuvassa 10. Lähettäjän ja vastaanottajan (A ja B) välillä on fyysinen yhteys, jotka pitkin signaali kulkee. Haaroitin siirtää saman signaalin samanlaista fyysistä yhteyttä käyttäen myös mittalaitteelle. [20]

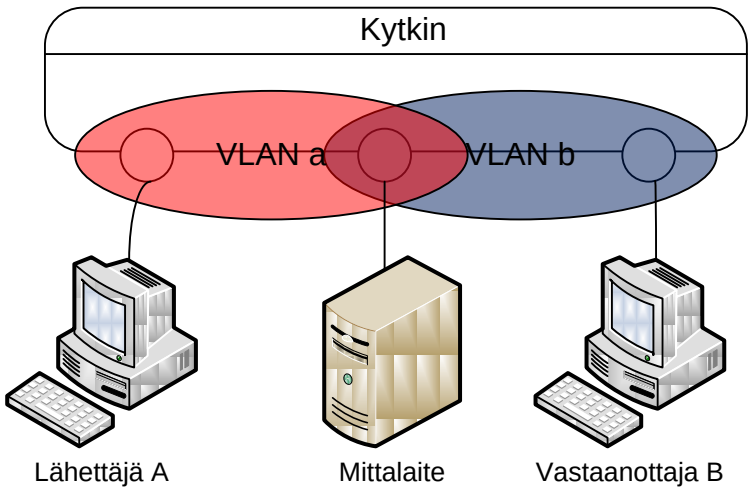


Kuva 10: Mittalaitteen kytkeytyminen verkkoon passiivisessa kuuntelussa

Liikenteen kopioiminen tapahtuu OSI-viitemallin siirtoyhteyskerroksessa tai verkkokerroksessa. Idea on se, että valitussa kerroksessa liikenteen reititystä muutetaan s.e. jossain verkon pisteessä liikenne kahdennetaan ja kopio ohjataan mittalaitteelle.

Siirtoyhteyskerroksessa tämä voidaan tehdä kytkimen avulla. VLAN (Virtual Local Area Networks) [21] avulla on mahdollista ohjata yhden tai useamman portin liikenne myös toiselle portille, johon mittalaite on liitetty. Ongelmana on se, että laitteen suorituskyky ei välttämättä riitä useammasta lähteestä koostettuun liikenteeseen; portti, johon mittalaite on liitetty, on maksiminopeudeltaan pienempi kuin tarkasteltavien liikennevirtojen yhteenlaskettu nopeus.

Mittalaitteen liittäminen verkkoon liikennettä kopioimalla on esitetty kuvassa 11. Kytkimeen on liittynyt useita erilaisia verkkoja ja päätelaitteita. Kahden eri päätelaitteen (A ja B) liikennettä mitataan. Kummankin kytkinportti kuuluu omaan VLAN joukkoon (a ja b). Mittalaitteelle toimitetaan molempien päätelaitteiden liikenne, joten sen kytkinportti kuuluu molempiin. [20]



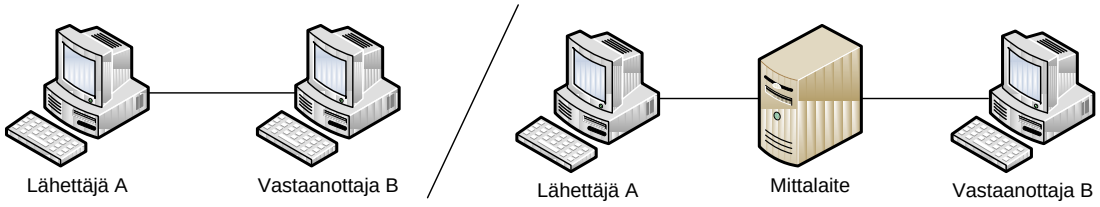
Kuva 11: Mittalaitteen kytkeytyminen liikenteen kopionnissa

Läpiviennissä liikenne kulkee mittalaitteen läpi. Verkkotopologian kannalta mittalaite itsessään on läpinäkyvä. Liikennettä voidaan analysoida nousemalla viitemallissa haluttuun kerrokseen tai kerroksiin.

Reaaliaikaisten analysointien perusteella liikennettä voitaisiin estää tai muuttaa. Tällöin puhutaan palomuureista ja keskeyttävistä välimuistipalvelimista. Kummatkaan eivät kuulu tämän työn piiriin.

Useat erilaiset verkkolaitteet omaavat jonkinasteista toiminnallisuutta verkkoliikenteen mittaamiseen varsinaisen toimintansa ohella. Tyypillisesti tällaisten verkkolaitteiden hallinta suoritetaan SNMP:n (Simple Network Management Protocol) [22] avulla. Tällöin verkkohallintajärjestelmä (Network Management System, NMS) noutaa kerätyt tiedot. Muuttujat ovat ennalta määrättyjä. Tämä tarkoittaa, että voidaan seurata liikenteen sijasta vain liikenteen tunnuslukuja ja siten menetelmä ei sovellu täysveriseen liikenteen mittaamiseen.

Osaa laitteista on mahdollista käyttää muun toiminnan ohessa mittapisteenä, jolloin se lähettää määritellyn osan liikenteestä ja sen tunnusluvuista edelleen analysoitavaksi.



Kuva 12: Mittalaitteen kytkeytyminen läpiviennissä

Kuvassa 12 on esitetty mittalaitteen kytkeytyminen verkkoon läpiviennin avulla. Lähettäjän ja vastaanottajan (A ja B) näkökulmasta mittalaitetta ei ole kytketty. Tosiasiallisesti se on liitetty verkkoon niiden väliin. [20]

2.4.3. Paketinkaappaus

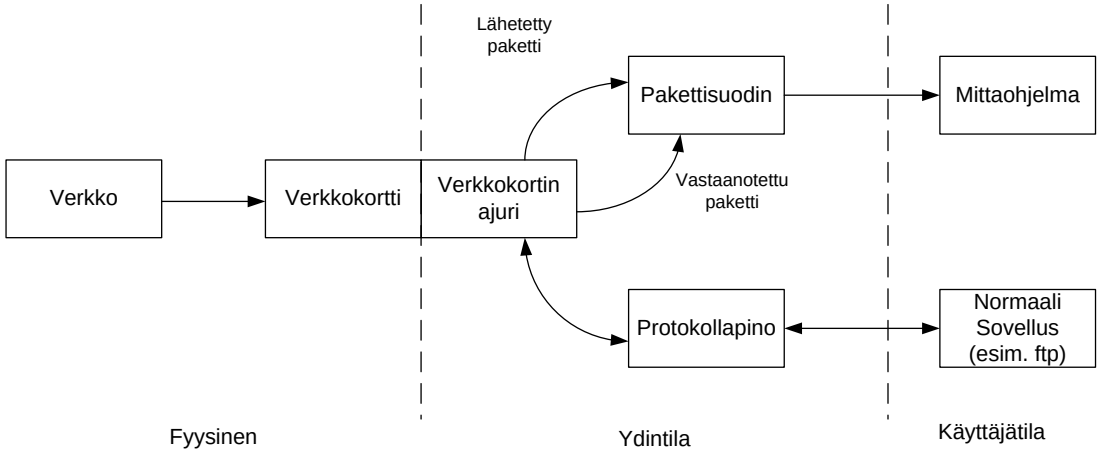
Paketin kaappaamisen prosessi on se toimi, jolla kerätään dataa sen kulkiessa verkossa. Paketinkaappausta käyttää toiminnassaan monenlaiset sovellukset, kuten esimerkiksi tietomurtohälyttimet (IDS, intrusion detection system) ja verkkoanalysaattorit. Paketin kaappaamisen prosessi on riippuvainen fyysisen- ja sovelluskerroksen toteutuksista järjestelmässä. Pääpiirteiltään se on kuitenkin aina samanlainen. Ohessa esitellään paketin kaappaaminen Ethernet-verkosta. [23]

Normaalissa liikennöinnissä verkkokortti ottaa vastaan kehyksiä verkosta ja vertaa sen kohdeosoitetta omaan siirtokerroksen osoitteeseensa. Jos se täsmää, luo verkkokortti keskeytyspyynnön. Keskeytyksen käsittelee verkkokortin oma ajuri, joka on asennettu käyttöjärjestelmään. Ajuri lisää vastaanotettuun dataan aikaleiman ja kopioi sen verkkokortin muistista käyttöjärjestelmän ytimen muistiin. Datasta tarkistetaan minkä tyyppinen PDU on kehyksen sisällä. Sen mukaan data välitetään edelleen vastaavalle protokollan käsittelijälle. Lopulta, protokollapinon läpikäytyään, data saavuttaa kohdesovelluksen, joka sijaitsee käyttäjätilassa. [23]

Pakettia kaapatessa käydään läpi sama prosessi, mutta verkkokortin ajuri lähettää kopion jokaisesta paketista, myös niistä, joiden osoite ei vastannut verkkokortin

omaa, pakettisuodattimelle, joka sijaitsee - protokollapinon tavoin - ydintilassa. Se välittää saamansa paketit nimetylle sovellukselle käyttäjätilaan. Oletusarvoisesti pakettisuodatin välittää kaikki paketit, mutta sen toimintaa voi hallita. [23]

Molemmat prosessit on esitetty kuvassa 13. Kuva on jaettu kolmeen osaan: fyysiseen, joka vastaa laitteistoa ja käyttöjärjestelmän kahteen eri muistiavaruuteen, ydintilaan ja käyttäjätilaan.



Kuva 13: Prosessi paketin vastaanottamiseen ja kaappaamiseen

2.4.4. Mittaustapojen ongelmia

Perimmäinen ongelma aktiivisissa mittauksissa on se, että verkon tilaa muutetaan mittauksen aikana. Impulssi voi vaikuttaa verkkoon niiltä osin kuin se mittauksen kannalta oleellista ja siten vääristää tuloksia. Aktiivinen mittaaminen ei ole luotettavaa myöskään siinä mielessä, että verkon ylläpito voi antaa mittauksille normaaliin liikenteeseen verrattuna erilaisen palvelun laadun ja siten manipuloida tuloksia. [20]

Asian voi järkeistää sillä, että minkälaisia eroja tuloksiin tulee, jos samanaikaisesti tarkistetaan kymmenen tai kymmenentuhannen kohteen tilaa ping-ohjelmalla. Tämän vuoksi aktiiviset mittaukset tulee mitoittaa ja ajastaa tarkasti.

Karkeasti ottaen aktiivinen mittaus mittaa verkkoa ja on vaara, että se muuttaa siinä olevaa liikennettä. Käänteisesti passiivinen mittaus tarkastelee verkossa olevaa liikennettä, mutta saattaa vaikuttaa verkon tilaan. Mittalaite kytketään verkkoon lisäämällä uusi verkkoelementti tai muuntamalla jo olemassa olevaa verkkoelementtiä. Tämä alentaa verkon toimintavarmuutta.

Verkon toimintavarmuus, kuinka suuren osan ajasta se on käytettävissä, ilmaistaan tyypillisesti siten, että kuinka monta ensimmäistä desimaalia ovat yhdeksikköjä. Ihannetilanteena on "viisi yhdeksikköä" eli 99.999% toimintavarmuus, vaikka sitä ei yleensä tavoiteta. [24]

Kahden toisistaan riippumattoman todennäköisyyden leikkaus on niiden todennäköisyyksien tulo. [25] Se on esitetty kaavassa 1. Taulukossa 1 on laskettu

mittalaitteen ja verkon toimintalaitteen yhdistetty toimintavarmuus, kun kummankin oma toimintavarmuus vaihtelee kahdesta viiteen yhdeksikköön.

$$P(A \cap B) = P(A)P(B) \quad (1)$$

Taulukosta 1 nähdään, että kun mittalaite ja verkko ovat yhtä toimintavarmoja, viimeinen dekadi muuttuu kahdeksaksi. Käytännössä tämä tarkoittaa, että aika, jonka yhdistetty systeemi on poissa toiminnasta, kaksinkertaistuu.

Taulukko 1: Mittalaitteen ja verkon yhdistetty toimintavarmuus

		Verkon toimintavarmuus			
		99 %	99,90 %	99,99 %	99,999 %
Mittalaitteen toimintavarmuus	99 %	98,010 %	98,901 %	98,990 %	98,999 %
	99,90 %	98,901 %	99,800 %	99,890 %	99,899 %
	99,99 %	98,990 %	99,890 %	99,980 %	99,989 %
	99,999 %	98,999 %	99,899 %	99,989 %	99,998 %

Toinen merkittävä ongelma passiivisissa mittauksissa on mitattava liikenne, tarkemmin sen määrä. Mittausjärjestelmään voi syntyä pullonkauloja. Aikaisemmin on mainittu, että liikennelähteitä yhdistämällä voidaan joutua tilanteeseen, jossa mitattava liikennevuo on suurempi kuin mittalaitteen liitännän maksimikapasiteetti. Sen lisäksi liikenteen määrä vaikuttaa mittausjärjestelyn analysointiin. [20]

Reaaliaikaisen analysoinnin ongelmana on laskentakapasiteetti: mittajärjestelyn tulee pystyä käsittelemään liikennettä myös sen maksimikapasiteetilla. Kun mittausdata kerätään analysoitavaksi myöhemmäksi ongelmaksi nousee kertyvän datan määrä ja sen tallennus ja kuljetus. Nykyisin yleisesti lähiverkoissa käytössä oleva 100Mbps tuottaa täydellä kapasiteetillaan lähes kaksi teratavua dataa vuorokaudessa. Vastaavasti kertyisi mittausdataa nopeudeltaan 10 Gbps olevasta runkoverkkolinkistä, jonka käyttöaste on 50%, noin 98Tt eli noin 1,2Gt sekunnissa. Suuren datamäärän lisäksi vain harvat tallennusjärjestelmät pystyvät yhtä nopeasti jatkuvaan kirjoittamiseen. [20]

2.5. Analysointi ja visualisointi

Tässä luvussa käsitellään pääsääntöisesti analysointia ja visualisointia, mutta osan käsiteltävistä asioista voisi mieltää kuuluvan mittauksen piiriin.

2.5.1. Analysoinnin kannalta tärkeitä tietorakenteita

2.5.1.1. Viisikko

Kun yhden protokollatietoyksikön sisältöä yritetään kiteyttää jää jäljelle se, että tärkeintä on tietää mistä tietoyksikkö tulee, minne se on menee ja mitä se sisältää.

Verkkokerroksen protokollan (IP) otsikkotiedoista saa selville verkkotason kohde- ja lähdeosoitteen. Kuljetuskerroksella on käytössä vastaavasti lähde- ja kohdeportit,

mikäli käytetty protokolla niitä tukee. Tämä vastaa kahteen ensimmäiseen kysymykseen. Viimeiseen kysymykseen ei ole yksiselitteistä vastausta. Tarkastelemalla jo viisikkoon valittuja portteja ja tekemällä ero osoitteiltaan samanlaisten TCP ja UDP välillä IP otsikon protokollakentän avulla. Vertaamalla käytettyjä portteja varattujen porttien listaan [13] voi tehdä valistuneen arvauksen siitä, mitä hyötykuormassa on.

Tämä viisikko (lähdeosoite, kohdeosoite, lähtöportti, kohdeportti ja kuljetuskerroksen protokolla) on yleisessä käytössä Internet-liikenteen kuvaamisessa. Tyypillisesti jokaisen talletetun viisikon yhteyteen tallennetaan aikaleima, jolloin ne voidaan sijoittaa aikajanalle.

2.5.1.2. Vuo

Karkeimmillaan vuo on joukko paketteja, jotka jakavat yhden tai useamman ominaisuuden, jotka liittyvät sen päätepisteisiin. Ne voivat perustua edellä esitettyyn viisikkoon, parista verkkoavaruuksia (esim. 192.168.1.0/24 ja 10.0.0.0/8) tai kahdesta listasta verkkoelementtejä (IP osoitteita). [26]

Jos vuon jaetut ominaisuudet ovat vain tilallisia, on se ajallisesti ääretön eli ei voida määrittää sen alku ja loppuaikaa. Vuolla kuitenkin on elinikä: se alkaa ensimmäisestä havaitusta PDU:sta ja päättyy viimeiseen havaintoon. Koska käytännössä paketin ei voi todeta olevan viimeinen sen havaitsemishetkellä, käytetään aikakatkaisua. Kun viimeisen havaitun paketin jälkeen on kulunut ennalta määrätty aika, katsotaan vuo päättyneeksi. Mikäli samoilla määreillä alkaa uudelleen vuo, katsotaan sen olevan uusi ja edellisestä riippumaton. [26]

Vuolla on useita erilaisia määritelmiä. Se voi olla yksisuuntainen tai kaksisuuntainen. Jälkimmäisessä tapauksessa samaan vuohon katsotaan kuuluvan myös ne PDU:t, jotka vastaavat, kun lähde- ja kohdeosoitteiden ja -porttien paikkaa vaihdetaan keskenään. [26]

2.5.2. Mittaustulosten määrän vähentäminen

Kuten aikaisemmin on todettu, kertyy passiivisista mittauksista merkittävä määrä mittaustuloksia. Ennen analysointia ja taltiointia on mittaustulosten määrää perusteltua vähentää. Sen voi toteuttaa muun muassa alla kuvatuilla tavoilla.

2.5.2.1. Suodattaminen

Suodattaminen tarkoittaa tässä merkityksessä yksinkertaisesti sitä, että joukosta valitaan alijoukko jonkin säännön tai jaetun ominaisuuden mukaan. Käytännössä liikenteestä valitaan vain tietyn kriteerin tai kriteerit täyttävät PDU:t mittaustulokseen.

Arloksen mallissa suodatus on mittausosan tehtävä. Tämä tekee suodattamisesta hyvän tavan vähentää mittausdataa, sillä vähentäminen tapahtuu varsin varhaisessa vaiheessa ja siten poistaa kuormaa useammalta myöhemmiltä osilta.

Suodatin pitää asettaa erikseen jokaiselle mittauskohtaiselle analyysille. Esimerkiksi tarkasteltaessa ilmiötä tietyn protokollan sisällä, voidaan suodattaa vain kyseisen protokollan paketit kuljetustason protokollan ja porttien mukaan.

2.5.2.2. Näytteenotto

Näytteenoton idea on yksinkertainen, otetaan pienempi osajoukko edustamaan koko joukkoa. Esimerkiksi tunnin aikana tapahtuvaa liikennettä tutkittaessa mitattaisiin 15 minuutin otos, joka analysoitaisiin koko liikenteen sijasta. Esimerkissä esitetyn ei-tilastollisen näytteenoton lisäksi voidaan näytteet valita mm. seuraavilla tavoilla:

Satunnaisessa näytteenotossa nimensä mukaisesti näytteet valitaan täysin satunnaisesti. Satunnaisessa näytteenotossa hyvänä puolena, että siinä näytteenotosta johtuva vääristymä on pienempi. [27]

Tilastollisessa näytteenotossa valitaan tutkittavan joukon joka N:näs jäsen. Tämän menetelmän etu on lähinnä helpossa implementaatiossa. [27]

Kerrostuneessa näytteenotossa valitaan tutkittavasta joukosta jokin määräävä tekijä, esimerkiksi kohdeportti, ja valitaan näytteet siten, että sekä näytteessä että tutkittavassa joukossa kyseisen tekijän jakauma on vastaava. [27]

Verkkoilmiöt ovat käytännössä äärettömiä ja sen takia niitä tarkasteltaessa näytteenotto on lähes mahdotonta suorittaa onnistuneesti. Näytteenotto soveltuukin parhaiten tilanteisiin, kun halutaan testata järjestelmää tai analysointimenetelmää. [27]

2.5.2.3. Otsikkotiedot vs. protokollatietoyksikkö

Yksi suoraviivainen tapa vähentää kertyvää dataa on olla käsittelemättä protokollatietoyksiköitä täydellisesti. Sovelluskohtaisesta analysointimenetelmästä pystyy arvioimaan, mitä osaa tai osia protkollatietoyksiköstä se käsittelee. Tyypillisesti valinta on joko-tai eri protokollien otsikkotietojen ja hyötykuorman välillä. Kompaktimpaa lähestymistapaa edustaa edellä mainittu viisikko. Tämän ansiosta voidaan, menetelmä tuntemalla, valita vain ne osat protokollatietoyksiköstä, jotka ovat merkityksellisiä analysointia varten.

Ero tämän tavan ja suodattamisen välillä on se, että suodatettaessa jätetään huomiotta osa tapahtumista (PDU:sta) ja tässä tapauksessa kaikki tapahtumat käsitellään, mutta jokaisesta jätetään osa pois.

2.5.3. Aggregointi ja disaggregointi

Aggregoinnissa ja disaggregoinnissa on kyse siitä, että mittaustuloksista otetaan tarkasteluun yksittäisiä parametreja, joita koostetaan tai jaotellaan. Aggregoinnissa uusi sarja konstruoidaan yhdistämällä kaksi tai useampia kohteena olevaa sarjaa yhdeksi uudeksi sarjaksi. Disaggregoinnissa uusi sarja konstruoidaan jakamalla dissaggregoinnin kohteena oleva sarja osiinsa. [28]

Verkkoliikenteen analysoinnissa aggregoinnin tavoitteena on kerätä valitut parametri tai parametrit määrättyä muuttujaa vastaan. Disaggregoinnissa mittaustuloksista valitaan alijoukkoja s.e. muut samoihin tapahtumiin liittyvät tiedot säilyvät. Tyypillisesti koostaminen liittyy aikaan ja hajauttaminen tilaan. [27]

Aggregointi ja disaggregointi ovat suhteellisia toimenpiteitä. Ne riippuvat aina siitä minkä muuttujan tai muuttujien suhteen operaatio tehdään. Saman muuttujan suhteen

tehtynä aggregaatio ja disaggregaatio ovat toistensa käänteisoperaatioita. Voidaan olettaa yleensä, että aggregaatio on helpompi ja mahdollinen toimenpide. Sitä vastoin disaggregaatiota voi olla mahdotonta toteuttaa. [27]

Tila-aggregaatio kerää yksittäisiä havaintoja aggregaateiksi. Se tuottaa sarjan, joka kuvaa mitatun liikenteen ominaisuuksia valitun aggregointipisteen suhteen. Yksinkertainen esimerkki olisi valita pisteeksi paketin koko, jolloin tulokseksi saadaan sen jakauma mittaustuloksissa. [27]

Disaggregaatio tilan suhteen ottaa esille tietyn osa mittaustuloksista. Edellä kuvattu suodattaminen on osa tämän tyyppistä disaggregaatiota. Aikaa ei oteta huomioon tila-disagregaatiossa: suodattamisen lisäksi eri valitut muuttujat mitatuista tapahtumista summattaisiin yhteen. Esimerkki tästä olisi selvittää tietyn verkkoavaruuden jokaisen IP-osoitteen liikenteen määrä mittausjaksolla. [27]

Aika-aggregaatiossa muodostetaan valituista muuttujista aikasarja. Esimerkiksi PDU:n kokoja aika-aggregoimalla saadaan sarja verkkoliikenteen nopeudesta. Tarkasteltavasta aikaskaalasta riippuu minkä kokoisena havaintoaikaa pitää. Jos tarkastellaan välitöntä siirtonopeutta, on sekunti hyvä valinta. Vuotuisia trendejä seuratessa taas vuorokausi on parempi valinta. [27]

Käytännössä analysoitavia muuttujia aggregoidaan ja disaggregoidaan eri pisteiden suhteen ennen lopullista tulosta. Edellisen esimerkissä päiväkohtaiset liikenteet disaggregoidaan vuorokauden jokaiselle sekunnille, jolloin saadaan suureeksi vertailukelpoinen vuorokauden keskimääräinen siirtonopeus. [27]

2.5.4. Tilastollinen analyysi

Kuten on todettu, on aggregaatio ja disaggrekaatio merkittäviä tapoja jäsentää mittaustuloksia. Ne muodostavat sarjoja, joita on hyvä analysoida tilastollisin menetelmin.

2.5.4.1. Tilastollisia muuttujia

Vaihteluväli kertoo suurimman ja pienimmän arvon välisen eron. Sillä on merkitystä erityisesti välivaiheena mittaustuloksia normalisoitaessa. Kaavassa 2 on vaihteluvälin formaali määritelmä. [27]

$$R = \max_i(x_i) - \min_i(x_i) \tag{2}$$

Aritmeettinen keskiarvo on yksi yleisimmin käytetyistä tilastollisista suureista. Se käyttää koko aineiston keskimääräisen arvon laskemiseen. Aritmeettisen keskiarvon laskemista varten lasketaan aineistosta summa, joka jaetaan yksittäisten alkioiden lukumäärällä. [27]

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{3}$$

Suuret ääriarvot vaikuttavat merkittävästi aritmeettiseen keskiarvoon. Sen vuoksi toisinaan käytetään mediaania keskiarvon sijasta. Mediaani on järjestetyn sarjan

keskimmäinen arvo parittomille sarjoille ja kahden keskimmäisen arvon aritmeettinen keskiarvo (parilliset sarjat, kaava 4) [27]

$$\frac{x_{n/2} + x_{n/2+1}}{2} \tag{4}$$

Varianssi mittaa arvojen vaihtelun laajuutta odotusarvon ympärillä. Se riippuu käsiteltävistä arvoista ja asteikosta. Kun varianssi lasketaan äärelliselle populaatiolle käytetään odotusarvon sijasta keskiarvoa. [27] Tällöin saatetaan käyttää nimikkeenä populaatio- tai otosvarianssia. Varianssi on esitetty seuraavasti

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1} \tag{5}$$

Tässä mallissa pitää kuitenkin tietää etukäteen keskiarvo kuinka monta jäsentä sarjassa on. Kun varianssia lavennetaan keskiarvon kaavalla, saadaan kaava, jonka laskeminen ohjelmallisesti on helpompaa, koska sarjan tarvitsee käydä läpi vain kerran. Laajennettu kaava on seuraava [27]

$$s^2 = \frac{n\sum_{i=1}^{n} {x_i}^2 - (\sum_{i=1}^{n} x_i)^2}{n(n-1)} \tag{6}$$

Keskihajonta on varianssin positiivinen neliöjuuri. Keskihajonta kertoo kuinka paljon tarkasteltavat arvot poikkeavat keskimäärin odotusarvosta, otoskeskiarvon tapauksessa keskiarvosta. Laadultaan se on sama kuin tutkittava muuttuja. [25]

2.5.4.2. Normalisointi

Normalisointia käytetään mittausdataan, jotta yhteensopivuus eri mittaussarjojen kesken. Sen avulla on myös mahdollista paremmin havaita ilmiöitä, joissa vaihteluväli on pieni. Normalisoinnin perusideana on siirtää mittaustulosten vaihteluväli toiselle skaalalle. Samalla tavalla normalisoidut mittaustulokset ovat keskenään vertailukelpoisia. Yleisesti käytetty normalisoinnin tapa on käyttää lukumääräisen jakauman sijasta prosenttiosuuksia. Muita tyypillisiä vaihteluvälejä, joille normalisointi tehdään ovat [0,1] ja [-1,1]. Sopivan normalisointimenetelmän valitseminen riippuu tilanteesta. [27]

Luoma & kumpp. esittävät oppikirjaluonnoksessaan [27] harkittavaksi seuraavia yksinkertaisia menetelmiä normalisointiin:

Minimi-maksimi-normalisoinnissa mitattu vaihteluväli skaalataan lineaarisesti uudelle vaihteluvälille. Menetelmän uuden arvon laskeminen on seuraava:

$$x_{uusi} = \frac{x_{vanha} - min_{vanha}}{max_{vanha} - min_{vanha}} R_{uusi} + min_{uusi} \tag{7}$$

Z-piste-normalisointi on käytännöllinen silloin, kun ei ole tiedossa alkuperäisen sarjan minimi- ja maksimiarvoja tai jos niiden ulkopuoliset arvot sotkevat edellisessä kohdassa esitetyn minimi-maksimi-normalisoinnin tuloksen. Tarkoitus on, että suurin osa uuden sarjan arvoista on korkeintaan keskihajonnan päässä keskiarvosta. Uusi

arvo lasketaan vähentämällä vanhasta arvosta vanhan sarjan keskiarvo ja jakamalla se keskihajonnalla.

$$x_{uusi} = \frac{x_{vanha} - \bar{x}_{vanha}}{s} \quad (8)$$

Desimaaliasteikolla normalisointi tarkoittaa, että tulosten desimaaliasteikkoa siirretään siten, että maksimiarvo jää alle halutun eli n valitaan kaavassa 9 siten, että se on pienin kokonaisluku, jolla toteutuu ehto: $max(|x_{uusi}|) \leq haluttu\ maksimi$.

$$x_{uusi} = \frac{x_{vanha}}{10^n} \quad (9)$$

Lisäksi joitain tilastollisia suureita on hyvä valita erityisesti vertailuun. Keskihajonnan, jonka suuruus riippuu aineistosta, josta se on laskettu, sijaan on hyvä käyttää vaihtelukerrointa (v), joka kertoo keskihajonnan suhteessa keskiarvoon:

$$v = \frac{s}{\bar{x}} \quad (10)$$

2.5.4.3. Kvantiilit

Kvantiili on järjestettyjen tulosten kertymäfunktio jaettuna n kappaleeseen, jolloin saadaan n-kvantiili. Silloin k:s kvantiili on sellainen arvo x, jolloin todennäköisyys pienempi x:n arvo on $\frac{k}{n}$. Joillain n:n arvoilla kvantiilille on oma nimitys: 4-kvantiiliä kutsutaan kvartaaliksi, 10-kvantiilia desiiliksi ja 100-kvantiilia persentiiliksi. [29] [27]

Ensimmäistä 4-kvantiiliä kutsutaan alakvartiiliksi, toista keskikvartiiliksi ja viimeistä, kolmatta, yläkvartiiliksi. Yleensä niitä merkitään vastaavasti Q_1, Q_2 ja Q_3. Keskikvartiili on myös samalla mediaani. [29]

Kvartiiliväli on se väli, jolle järjestetyn sarjan keskimmäiset 50% havainnoista sijoittuvat eli alakvartaalista yläkvartaaliin: [30]

$$Q_r = Q_3 - Q_1 \quad (11)$$

2.5.4.4. Histogrammi

Hyvä tapa sarjan karkeaan analysointiin on histogrammi. Sen avulla voi tarkastella visuaalisesti tuloksia, jolloin sitä voi esimerkiksi verrata silmämääräisesti tunnettuihin jakaumiin tai arvioida karkeasti tilastollisia muuttujia. [27]

Idea histogrammissa on se, että arvo-alue jaetaan samankokoisiin väleihin, laareihin. Jokainen havainto jaetaan omaan laariinsa, jolloin jokaista laaria vastaa siihen kuuluvien havaintojen lukumäärä. Kun ne kuvataan pylväsdiagrammilla, saadaan histogrammi. [27]

Osaan verkkoliikenteestä tehtävistä mittauksista tämä ei sovi sillä yksittäiset arvot ovat merkityksellisiä sinänsä, kuten porttinumerot, tai ne on parempi järjestellä muiden seikkojen mukaan, kuten IP-osoitteet verkkosegmenttien mukaan. [27]

Laarien lukumäärän ja koon valitseminen määrää myös histogrammin käyttökelpoisuuden. Siksi siihen pitää käyttää erityistä huomiota. Luoma & kumpp. esittävät liikenteen analyysissa sopivaksi tavaksi laskea laarin koko seuraavasti: [27]

$$W = 2 \cdot Q_r \cdot N^{-\frac{1}{3}} \quad (12)$$

Missä W on laarin koko, Q_r kvartaalivälin pituus ja N otoksen havaintojen lukumäärä.

Muodollisten kaavojen lisäksi esitetään seuraavaa proseduuria laarien koon ja lukumäärän määrittämiseksi: [27]

1. Laarien lukumäärä (ensimmäinen arvio): Lasketaan lukumäärä laareille käyttäen kaavaa 13, missä N on havaintojen lukumäärä.

$$n_1 = \sqrt{N-1} - 1 \quad (13)$$

2. Laarin koko (ensimmäinen arvio): Jaetaan koko alue n_1 kokoisiin paloihin kaavalla 14. n_1 on laarien lukumäärän arvio ja R vaihteluväli:

$$s_1 = \frac{R}{n_1} \quad (14)$$

3. Laarin koko (tarkennus): Pyöristetään laarin koon arvio ylöspäin vastaamaan havaintojen tarkkuutta. Esimerkiksi sen ollessa 0,4 ja havaintojen tarkkuus tasalukuja pyöristetään arvoksi s = 1.

4. Laarien rajat: Ensimmäisen laarin alaraja on havaintojen pienin arvo, josta on vähennetty puolet tarkkuudesta s: $min(x) - \frac{1}{2}s$. Yläraja, joka on samalla toisen laarin alaraja, on siitä laarin koon s päässä. Lopulta viimeisen laarin yläraja on $n_1 \cdot s$ etäisyydellä ensimmäisen laarin alarajasta.

2.5.4.5. Tulosten anonyymisointi

Internet-liikenteessä jokaisella lähettäjällä tai lähettäjäryhmällä on globaalisti uniikki IP-osoite. Sitä voidaan verrata palveluntarjoajan tietokantoihin, jolloin IP-osoitteet on mahdollista sitoa paikkaan ja henkilöön ja siten mieltää tunnistetiedoiksi. [31]

"Viesti ei ole luottamuksellinen, jos se on saatettu yleisesti vastaanotettavaksi. Viestiin liittyvät tunnistamistiedot ovat kuitenkin luottamuksellisia.". [32] Viestintä tietoverkoissa, kuten Internetissä voidaan mieltää koostuvaksi viesteistä. Laki kuitenkin viittaa *viestillä* tässä yhteydessä viitemallin ylempien kerrosten viesteihin, kuten sähköposteihin. [32]

Ongelma on siis kaksijakoinen: verkkotason osoitteiden voidaan mieltää olevan tunnistetietoja ja verkossa liikkuvat viestit ovat viestintäsalaisuuden alaisia. Molempiin on oma ratkaisunsa.

Verkkotason osoitteet voidaan näennäisesti anonyymisoida mittauksen tai analyysin aikana: osoitteet nimetään uudelleen siten, että säilytetään globaali uniikkius.

Toisaalta osoitteiden muuttaminen henkilötiedoiksi vaatii ristiintarkistusta muihin lähteisiin, jolloin ne eivät sellaisenaan ole tunnistetietoja. [31]

Viestintäsalaisuuden osalta tulee huomioida lain asettamat vaatimukset käsiteltäessä sellaisia protokollia, jotka lain mukaan määritellään viestintäsalaisuuden piiriin. Analysoinnin yhteydessä täytyy pitää huolta, että yksilöiviä tietoja ei välitetä eteenpäin. Tiedot sähköpostiliikenteen määrästä eivät tällaisia ole, mutta yksittäiset sähköpostit tai lainaukset niistä ovat.

2.5.4.6. Mitä mitata?

Mittauskohtainen analyysi määritellään erikseen jokaista mittausta varten. [20] Tässä työssä käytetyt mittauskohtaiset analyysitavat ja niihin liittyvät valinnat käsitellään luvussa 3. Sen lisäksi on esitetty koostamalla useasta lähteestä, että isoista liikenneotoksista seuraavien analyysien tekemistä suositellaan: [27]

- Liikennemäärät päivän, päivämäärän (viikonpäivien vaikutus, ym.) ja suunnan mukaan
- Liikennevirran suuruuden mittaus huippuarvojen selvittämiseksi
- Protokollien ja/tai sovellusten osuus liikenteestä
- Latenssin laskeminen käyttäen hyväksi liikenteessä esiintyvien TCP yhteyksiä.
- Hurst-parametrin estimaatin laskeminen eri menetelmiä käyttäen

3. Mittausjärjestelmä ja sen tekninen toteutus

3.1. Suunnittelun lähtökohdat

Mittausjärjestelmän suunnittelun ydinajatus on yksinkertainen: tarkoituksena on luoda yksinkertainen hajautettu reaaliaikainen tietoverkon liikenteen mittausjärjestelmä, joka toimii pohjana jatkokehitykselle. Lause on yksinkertainen, mutta pitää sisällään paljon. Hajottamalla se osiinsa on helppo mieltää keskeiset suunnitteluideat.

"mittausjärjestelmä" - Työn perimmäinen tulos on järjestelmä, joka tekee itsenäisiä mittauksia annettuun kohteeseen ja tuottaa niiden pohjalta visualisoituja tuloksia. Tämä antaa määreen sille mitä varsinaisesti tehdään, kun muut ovat lisämääreitä: miten se toteutetaan.

"tietoverkon liikenteen" - Mittauksen kohteena on moderni tietoverkko ja mittausjärjestelmän pitää pystyä käsittelemään niitä protokollia, jotka ovat käytössä mitattavassa verkossa ja liittyvät tarkasteltaviin ilmiöihin.

"reaaliaikainen" - Mittausjärjestelmä on kykeneväinen, ainakin tietyin rajauksin, käsittelemään mittauksia reaaliaikaisesti siitä verkosta, johon se on liitetty

"hajautettu" - Mittausjärjestelmä pyritään tekemään siten, että sen yksittäiset osat, tietyllä granulariteetilla, ovat toisistaan riippumattomia ja voidaan korvata rajapintojen pysyessä samanlaisina.

"yksinkertainen" ja "toimii pohjana jatkokehitykselle" - Järjestelmän ja sen yksittäisten osien rakenne pyritään pitämään mahdollisimman yksinkertaisena. Tavoitteena on, että tämän työn ja lähdekoodin avulla on mahdollista kehittää edelleen mittausjärjestelmää tai käyttää osia siitä muihin projekteihin.

3.1.1. Mittausjärjestelmän osat

Mittausjärjestelmän osiksi on valittu *lähde, mittapiste, analysaattori* ja *jatkokäsittely*. Suunnittelussa on pyritty siihen, että jokainen näistä osista on itsenäinen. Vaikka mallissa esitetyt nimetyt vaiheet, tehtävät, jakautuvat eri osien välille valitusta mittauskohteesta riippuen, on jokaisen toteutus itsenäinen moduuli.

3.1.1.1. Lähde

Mittausjärjestelmän lähde on se osa, joka luo mitattavan liikenteen. Lähde on osana lähinnä abstraktio niistä tavoista, joilla mitattava liikenne voidaan tuoda mittapisteelle. Aikaisemmin esitetyssä mittausjärjestelmän mallissa lähde mielletään olevan verkko passiivisessa mittauksessa tai verkkolaite aktiivimittauksessa. Tässä järjestelmässä lähde voi olla myös verkosta riippumaton, kuten syöte tiedostosta, joka sisältää aikaisemmin tallennettua liikennettä.

Lähteen ja mittapisteen rajapinta pääasiallisesti verkkoliikenteessä. Mikäli lähteenä on liikennetallenteen sisältävä tiedosto tai tietokanta, on rajapintana käyttöjärjestelmätason tietovuo.

3.1.1.2. Mittapiste

Mittapiste on järjestelmän ensimmäinen varsinainen osa. Se on myös ainut osa mittausjärjestelmästä, joka on yhteydessä lähteeseen. Näin pidetään kosketuspinta mittausjärjestelmän ja mitattavan verkon välillä mahdollisimman pienenä ja minimoidaan mittausjärjestelmän vaikutus verkon toimintavarmuuteen.

Mittapisteen tehtävä on ottaa vastaan liikenne lähteestä ja välittää se mittaustuloksina edelleen yhdelle tai useammalle analysaattorille. Mittauspiste voi myös suodattaa liikennettä sen ominaisuuksien mukaan ennen sen välittämistä tai tehdä näytteenottoa mitatusta liikenteestä ja lähettää näytteet edelleen.

Joissain analyysimenetelmissä on perusteltua, että mittapiste suorittaa mittauskohtaista analyysiä. Näin on etenkin yksinkertaisimmissa tapauksissa, kuten liikenteen määrän seuraamisessa suhteessa aikaan.

Mittapisteen ja analysaattorin rajapinta on yhteinen jaettu media. Erilaisia vaihtoehtoja on käsitelty myöhemmin tässä luvussa.

3.1.1.3. Analysaattori

Analysaattori ja mittapiste voidaan mieltää mustaksi laatikoksi, joka liitetään verkkoon. Sen tuottamat tulokset voidaan siirtää työasemaan jatkokäsittelyä varten. Tässä työssä analysaattori erotetaan mittapisteestä, jotta saavutetaan helpommin realisoitava ja modulaarinen malli.

Analysaattori ottaa vastaan mittapisteen keräämää mittausdataa. Se suorittaa näytteenoton annetusta mittausdatasta ja lopulta suorittaa mittauskohtaisen analyysin tai osan siitä. Lopputuloksena saadaan jäsenneltyjä tuloksia, tyypillisesti järjestettyjä havaintosarjoja, jotka on mahdollista helposti visualisoida.

Merkittävin ero analysaattorin ja jatkokäsittelyn välillä on se, että analysaattori on laite, jossa oleva prosessi ottaa vastaan mittauspisteen sille toimittamaa dataa ja muuntaa sen tulossarjoiksi automaattisesti.

Myös analysaattorin ja jatkokäsittelyn rajapinta on jaettu media. Edelliseen tapaukseen lisämääreenä on se, että media säilyttää siitä kautta siirretyn tiedon.

3.1.1.4. Jatkokäsittely

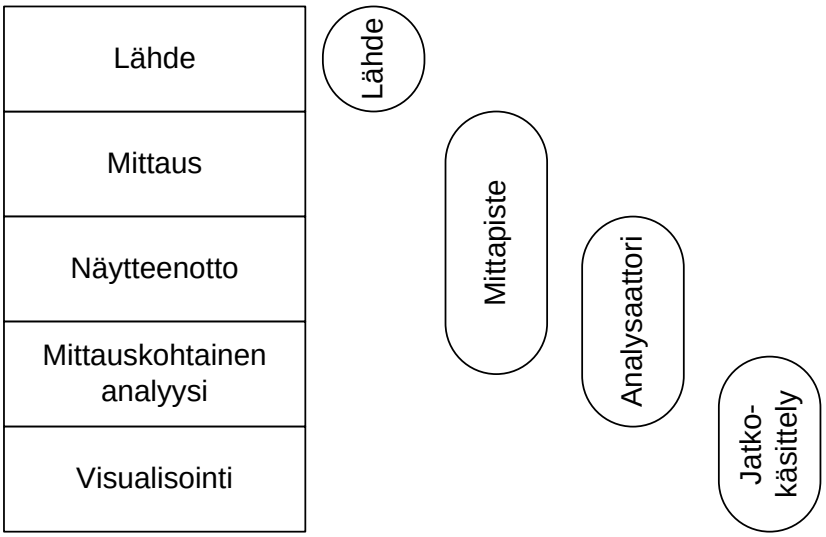
Jatkokäsittely, kuten lähde, on vähemmän konkreettinen osa mittausjärjestelmää. Se pitää sisällään kaikki ne menetelmät, joita käytetään analysaattorin tuottaman tiedon käsittelyyn ja visualisointiin.

Mittauskohtainen analyysi on analysaattorin ja jatkokäsittelyn välillä harmaata aluetta. Onkin menetelmäkohtaisesti arvioitava kuuluuko se analysaattorin vai jatkokäsittelyn vastuualueelle. Tässä työssä merkittävä semanttinen ero analysaattorin ja jatkokäsittelyn välillä on se, että analysaattori on automatisoitu ja jatkokäsittely on pääosin tehdään käyttäjälähtöisesti.

3.1.2. Suhde mittausjärjestelmän malliin

Luvussa 2.3.1. on esitetty mittausjärjestelmän malli. Sitä käytetään pohjana tämän mittausjärjestelmän jäsentämisessä. Arlos jakaa mittausjärjestelmän neljään osaan: lähteeseen, mittaukseen, analysointiin ja visualisointiin. Jokaisella näistä osista tarkasti määritellyt tehtävät, joita kukin toteuttaa.

Kuvassa 13 on esitetty osien suhde eri tehtäviin. Lähde on vastaava esitetyn mallin kanssa. Mittapiste voi mallissa olleen mittausosan tehtävien lisäksi tehdä suodatusta aggressiivisempaa liikenteen muokkaamista ennen analysaattoria. Lisäksi joissain analysointimenetelmissä se voi vastata myös mallin analyysiosaa. Analysaattori vastaa täysin mallin analyysiosaa. Jatkokäsittely vastaa visualisoinnista, mutta siihen voi sisältyä myös mittauskohtaista analyysiä.



Kuva 14: Mittausjärjestelmien osien suhde malliin

3.1.3. Toteutuksen kriteerit ja niiden täyttäminen

Luvussa 2.3.1. on esitetty kriteerejä mittausjärjestelmän toteutukselle. Ne otetaan huomioon jo järjestelmää suunniteltaessa. Jokainen kriteeri voidaan nähdä haasteena, johon on oma ratkaisunsa.

Modulaarisuus on jo muutenkin keskeinen kriteeri, kun suunnitellaan hajautettua järjestelmää. Tämän vuoksi kriteeri täytetään etuineen ja haittoineen jo ensiaskeleilla suunnittelussa.

Kustannustehokkuus saavutetaan käyttämällä vapaassa levityksessä olevia avoimen lähdekoodin ohjelmistoja. Järjestelmässä ei käytetä mitään erikoistunutta laitteistoa. Ratkaisut ovat ohjelmistopohjaisia ja siirrettävissä kohtuullisella vaivalla erilaisille alustoille. Hajautettu rakenne mahdollistaa resurssien mitoittamisen jokaiselle osalle erikseen ja siten vähentää yliresursointia.

Käytettävyys perinteisessä merkityksessä ei ole merkittävässä roolissa suunnittelussa; tavoitteena ei ole tehdä viimeisteltyä tuotetta. Käytettävyys mielletäänkin tässä yhteydessä helppoutena tutustua järjestelmään, sen toimintaan ja muokata sitä

toteutustasolla. Silloin se saavutetaan tällä kirjallisella työllä ja kooditasolla yksinkertaisella rakenteella.

Tietoturvallisuus on vähinten huomioitu näistä neljästä kriteeristä. Tiedon arkaluontoisuus ja lain asettamat vaatimukset tiedostetaan ja huomioidaan. Mihinkään erityisiin teknisiin toimenpiteisiin ei ryhdytä. Tiedon kulku järjestelmässä kuitenkin pyritään suunnittelemaan siten, ettei mitään arkaluontoisia tietoja pääse analysaattorin läpi.

3.2. Mittauskohteet

3.2.1. Liikenteen tunnusluvut otsikoista analysoimalla

Luvussa 2.5.4.6 on mainittu mittauksia, jotka olisi syytä toteuttaa aina isoja liikennemääriä analysoitaessa. Niistä ajastukseen liittyviä analysointimenetelmiä ei käytetä, koska käytössä olevat tavat simuloida verkkoliikennettä vääristävät alkuperäisen liikenteen aikaleimoja.

Eri protokollien osuus liikenteestä lasketaan. Sekä kuljetuskerroksessa, että joidenkin TCP:n ja UDP:n yläpuolisten protokollien osalta sekä osuutena kokonaisliikenteestä että pakettien lukumäärästä.

Pakettien kokojakaumasta lasketaan luvussa 2 esitettyjä liikenteen tunnuslukuja.

3.2.2. HTTP:llä siirretyt tiedostot

Tämän analyysimenetelmän on tarkoitus vastata kysymyksiin: Mitä ja minkälaisia tiedostoja siirretään HTTP:llä? Kuinka paljon näistä tiedostoista ovat samoja?

Ensimmäinen vaihe on eristää mitattavasta liikenteestä oikea osa: HTTP-liikenneosuus. IANA:n ylläpitämän porttilistan mukaan HTTP:lle allokoitu portti on 80/tcp. [13] Se osa liikenteestä, jonka lähde tai kohde kyseinen portti, voidaan olettaa olevan HTTP-liikennettä. Vaihtoehtoinen portti HTTP:lle on 8080/tcp. Osa HTTP liikenteestä ei käytä kumpaakaan näistä vakioiduista porteista. Aidosti kaiken HTTP liikenteen selvittämistä varten pitää käydä koko mitattava liikenne läpi kuljetuskerroksesta ylöspäin.

TCP on toteutettu siten, että porttiparien avulla voidaan päätellä liikenteen suunta: Asiakkaalta tulevat pyynnöt ovat sellaisia, joiden kohdeporttina on 80/tcp. Vastaavasti palvelimen vastaukset ovat lähtöisin samasta portista. Laajennutusti voisi TCP:n sekvenssinumeroita käyttää pyynnön ja vastauksen liittämisiin toisiinsa.

Tiedoston voi määritellä sen nimen ja sisällön mukaan. Tiedosto on sama, jos sekä sen nimi, että sisältö ovat identtisiä. Verkkoliikenteen analysoinnin näkökulmasta ei kuitenkaan ole kannattavaa pitää tietokantaa kaikkien tiedostojen sisällöistä. Perustellumpaa on verrata tiedostosta johdettuja suureita. [15]

HTTP:ssä tiedoston nimi on pyynnössä olevassa URI:ssa. Siitä on mahdollista päätellä tiedoston nimi. Palvelimet voivat käsitellä haluamallaan tavalla pyyntöjä ja usein käytössä tekniikoita, kuten mod_rewrite [33], joita käytettäessä pyydettävä URI ei vastaa toimitettua tiedostoa. Jos URI viittaa vain hakemistoon, on palautettava

tiedosto kiinni palvelimen toiminnallisuutena. Oletuksena palautetaan etusivu tai hakemiston indeksi HTML-muodossa.

Tiedoston sisältöön liittyvät määreet ovat vastauksessa. Tiedostoa voidaan kuvata otsikkotiedoilla, joista keskeisemmät ovat aiemmin mainitut content-length ja content-type. Otsikkotietojen käyttäminen on palvelimelle vapaaehtoista ja siksi ei voida olettaa, että jokaiselle pyydetylle tiedostolle saadaan tietoon MIME-tyyppi tai koko. Koko voidaan approksimoida olevan paketin koko, pois lukien alempien kerrosten protokollien otsikkotiedot. Tämä arvio ei riitä tiedostojen osoittamiseen identtisiksi, koska otsikkotiedot ja niiden koko vaihtelee vastauksen lähettävän palvelimen mukaan. Sitä voidaan kuitenkin käyttää tiedostojen kokoja analysoitaessa.

Samanlaisten tiedostojen etsimisen lisäksi analysoidaan MIME-tyyppien ja tiedostokokojen määrää ja jakautumista mitatussa liikenteessä.

3.3. Tekniset ratkaisut

3.3.1. Media

Tässä yhteydessä viitataan niihin tekniikoihin, jolla tietovuo välitetään mittausjärjestelmän osalta toiselle.

3.3.1.1. Tiedostojärjestelmä ja tietokanta

Molemmissa on kyse samasta asiasta: yksi osa järjestelmästä kirjoittaa datansa tallennusmatriisiin, tyypillisesti kovalevylle, josta seuraava osa sen lukee.

Ero kahden menetelmän välillä tulee siitä, minkälainen metodi tiedon järjestelyyn on. Tiedostojärjestelmän tapauksessa tiedon järjestelystä vastaa käyttöjärjestelmä ja tieto on tekstimuotoista tai sovelluskohtaisena binäärimuotona.

Tietokanta edelleen tallentaa tiedot kovalevylle, käyttäen hyväkseen käyttöjärjestelmän tarjoamia palveluita. Välissä on vain yksi ylimääräinen agentti, joka optimoi tiedon tallentamista ja noutoa. Tietokantaan tieto tallennetaan määrämuotoisina tietueina. Tietokannan käyttäminen lisää laskenta- ja muistiresurssien käyttöä.

3.3.1.2. Verkko

Verkko on monipuolisin liitäntätavoista. Standardoitu liittymistapa helpottaa rajapintojen toteuttamista. Verkko on myös yksi helpoimmista rajapinnoista, kun järjestelmä hajautetaan usealle eri laitteelle. Haittana on se, että verkkoon ei voi tallentaa tietoa pysyväisluonteisesti. Verkko on myös muita samantasoisia ratkaisuja tiedonsiirtonopeudeltaan selkeästi hitaampi.

3.3.2. Virtualisointi - VirtualBox

VirtualBox on alunperin Sun Microsystemsin kehittämä ja nykyisin se on yrityskauppojen kautta Oraclen hallussa. VirtualBox on saatavilla ilmaiseksi erilaisina

paketteina ja lähdekoodina. Sen hinta riippuu käyttötarkoituksesta ja halutuista ominaisuuksista.

Virtualbox on virtuaalisointityökalu: sen avulla on mahdollista asentaa ja suorittaa erilaisia käyttöjärjestelmiä ja ohjelmia erityisessä ympäristössä, virtuaalikoneessa. Tämä mahdollistaa luonnollisen tietokoneen ja sen käyttöjärjestelmän eli isännän käytön useiden erilaisten virtuaalikoneiden, vieraiden, ajamiseen.

Keskeisiä etuja virtuaalisoinnissa on laitteistoresurssien parempi käyttö. Tämän työn puitteissa tehty testiympäristö on muodostettu modernin kuluttajatason tietokoneelle. Jos virtuaalisointia ei olisi käytetty, olisi tämän mittausjärjestelmän tuottaminen vaatinut neljän koneen laboratorioympäristön.

Myös muita virtuaalisoinnin tuomia etuja on tämän työn tekemisessä käytetty. Kehitysvaiheessa mahdollisuus siihen, että tietokoneen tilan voi tallentaa vedoksena ja palauttaa tarvittaessa siitä, on nopeuttanut työtä. Laitteistoriippumattomuus ja kasvanut käyttöaste ovat mahdollistaneet työn tekemisen erilaisissa paikoissa.

Virtuaalisoinnin keskeisin haitta on alentunut suorituskyky verrattuna siihen, että samat ohjelmistot olisi asennettu suoraan laitteistoon. Tämän työn puitteissa ongelma kohdattiin tulossarjojen jatkokäsittelyssä. Paremmalla laitteistolla ja resurssien provisioinnilla olisi voinut vaikuttaa ongelman vakavuuteen.

Virtuaalisointi luo myös yhden ylimääräisen kerroksen, jonka asetukset ja ominaiskäyttäytyminen voivat vaikuttaa mittaustuloksiin. Esimerkiksi tämän työn puitteissa virtuaalisointiratkaisun sisäisen verkon toimintatapa esti suodatettujen pakettien välittämisen ennen kuin siihen pystyi, valmistajan ohjeilla, puuttumaan.

3.3.3. Käyttöjärjestelmä - Linux / Ubuntu

Linux on Unix käyttöjärjestelmän modulaarinen klooni, joka kehityksen aloitti suomalainen Linus Thorwalds. Se tähtää täydelliseen kattamaan POSIX ja Single Unix määrittelyjen vaatimukset. Varsinaisesti Linux viittaa vain ytimeen - käyttöjärjestelmän siihen osaan, joka vastaa sovellusten ja laitteiston välisestä kommunikaatiosta. [34]

Siinä missä Linux on ydin, on Richard Stallmanin johtaman GNU (GNU is not Unix) projektin tuotos itse käyttöjärjestelmä. Sen pyrkimys on samanlainen, täysin vapaa Unix:in kaltainen käyttöjärjestelmä, myös muilta osin kuin ytimeltä. [35]

Normaalissa kielenkäytössä merkitys on muuttunut muotoon "GNU käyttöjärjestelmän, jonka ydin Linux on, jakelupaketti". On tyypillistä, että yhteisö, yritys tai yksityishenkilö koostaa asennuspaketin, joka sisältää ytimen lisäksi valittuja ohjelmia ja ohjeita. [34] [35]

Tähän työhön on valittu käyttöjärjestelmäksi GNU/Linux sen teknisen toteutuksen, saatavien ja kustannustehokkuuden eli ilmaisuuden vuoksi. Jakeluksi valittiin Ubuntu, joka on jakeluna helppokäyttöinen, moderni, työpöytäkäyttöön soveltuva ja omaa suuren valmiin ohjelmistotarjonnan, osittain Debian-jakelun, josta Ubuntu on haarautunut, perintönä. [36]

3.3.4. GNU sovellukset

Aiemmin mainittuun GNU projektiin kuuluu paljon erilaisia ohjelmaprojekteja, joiden luomat sovellukset ovat vakiona GNU/Linux asennuksissa ja vastaavat toiminnallisuudeltaan suljettujen Unix varianttien vastaavia. Seuraavilla ohjelmilla on ollut merkittävä rooli mittausjärjestelmän toteutuksen kannalta.

coreutils - tämä paketti sisältää pieniä ohjelmia, jotka ovat tarkoitettu käytettäväksi komentoriviltä tai komentorivikehotteen skriptissä. Osa paketin ohjelmista ovat uudelleen implementoituja muiden käyttöjärjestelmien vastaavista. Käytössä on jatkokäsittelyssä sort (tiedoston järjestäminen riveittäin), wc (sanojen ja rivien lukumäärä), cat (tulostaa tiedoston), head sekä tail (tulostaa tiedoston n ensimmäistä tai viimeistä riviä) ja grep (syötteen suodatus). [35]

GCC ja peruskirjastot - GCC (Gnu Complier Collection) on GNU käyttöjärjestelmän kääntäjä se tukee useita kieliä, kuten C, C++, Java ja Fortran. Tässä työssä käytetään kääntäjää C-kielelle ja siihen liittyviä peruskirjastoja. [35]

gawk - AWK on alunperin AT&T Bellin laboratorioissa kehitetty skripteihin tarkoitettu kieli, jonka käyttöajatus on nopeasti kirjoitettavissa pienissä apuohjelmissa. Gawk on sen parserin GNU implementaatio. [35]

3.3.4.1. AWK

AWK on datalähtöinen kieli. Siinä käydään läpi tiedostoja rivi kerrallaan. Mikäli rivi vastaa koodissa olevaa määriteltyä muotoa, suoritetaan sille määrittelyä vastaava koodiblokki. AWK:n tapa käsitellä tiedostoa rivi kerrallaan mahdollistaa suurienkin tiedostojen käsittelemisen ilman kasvavia muistivaatimuksia. Tämä tekee siitä erinomaisen suurien määrämuotoisten tekstitiedostojen käsittelyyn. Ominaisuuksiensa takia AWK on otettu käyttöön tässä työssä tulossarjojen analysointiin ennen visualisointia.

3.3.5. Paketinkaappauskirjasto - libpcap

Libpcap, packet capturing library, on kirjaimellisesti paketinkaappauskirjasto. Se tarjoaa korkean tason rajapinnan pakettien kaappaamiselle verkosta. Se on suunnattu olemaan alustariippumaton sovellusrajapinta (API, application programming interface) - se tarjoaa yhtenäiset funktiot käyttöjärjestelmästä toiseen vaihtelevien paketinkaappaustoiminnallisuuksien käyttöön. [23]

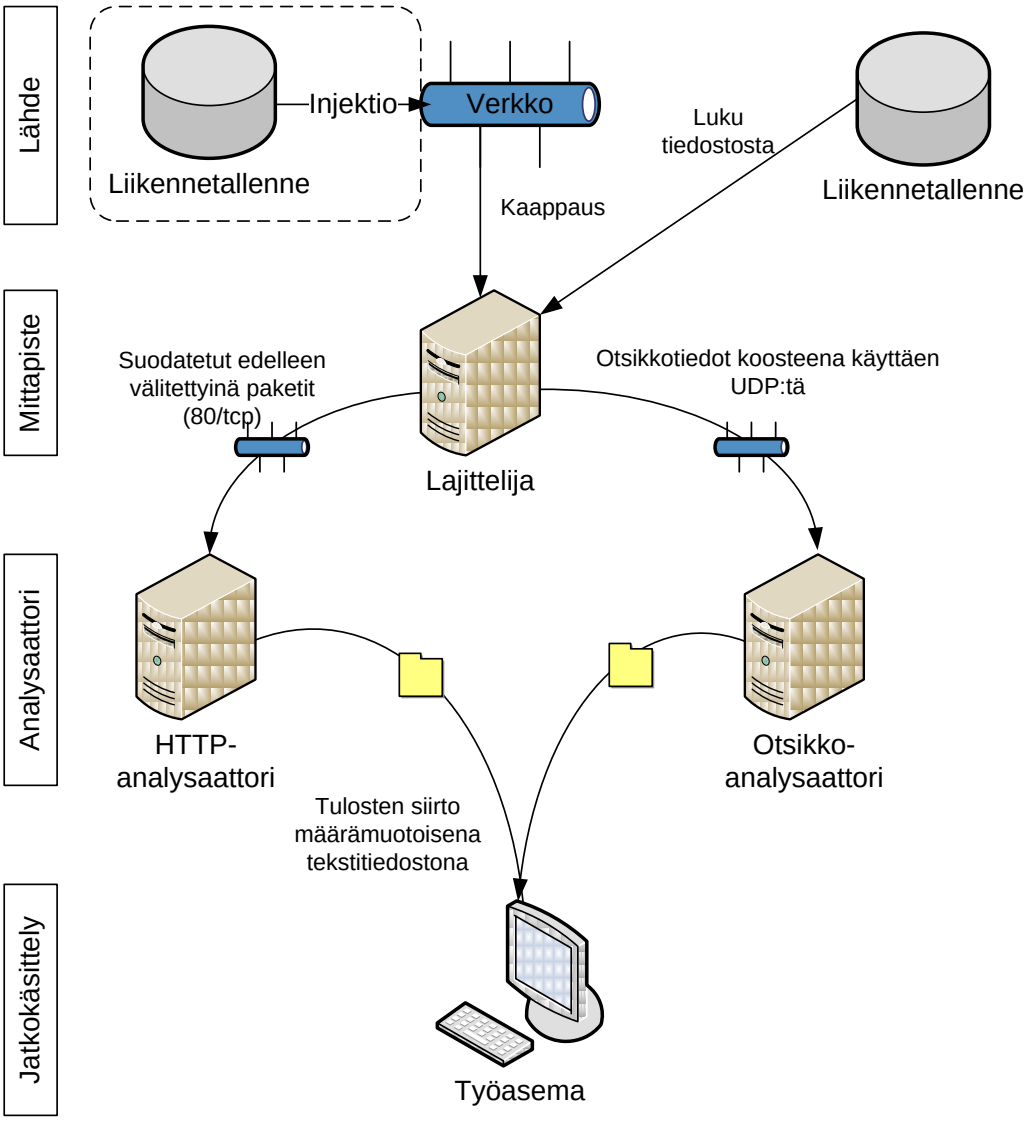
Libpcap on suunniteltu käytettäväksi C ja C++ kielten kanssa. Sille on kuitenkin tehty kääreitä, jotka mahdollistavat sen toiminnallisuuksien käytön myös muilla ohjelmointikielillä, kuten Pythonilla ja Javalla. Libpcap toimii suurimmalla osalla UNIX varianteista (Linux, Solaris, BSD, jne.) käyttöjärjestelmistä. Myös Windowsille on tehty versio, mutta se on, käyttöjärjestelmän [23]

Projektin on aloittanut vuonna 1994 Kalifornian yliopiston tutkijat ja se nykyään on Tcpdump yhteisön hallinnoima. Etu hallinnan siirtymisestä yhteisölle on se, että sitä kautta on kehittynyt yhteensopivuutta kirjaston ja sillä toteutettujen sovellusten (mm. tcpdump ja tcpreplay) välillä, kuten yhteinen tallennusmuoto. [23]

Tässä työssä on päätetty käyttää libpcap kirjastoa, koska se tukee useita erilaisia alustakonfiguraatioita, mikä tekee siihen perustuvan ohjelman helposti siirrettäväksi. Se on myös hyvin dokumentoitu, tuettu ja kehitetty. On olemassa valmiita työkaluja tukemaan sitä käyttäviä ohjelmia.

3.4. Tekninen toteutus

3.4.1. Yleiskuva



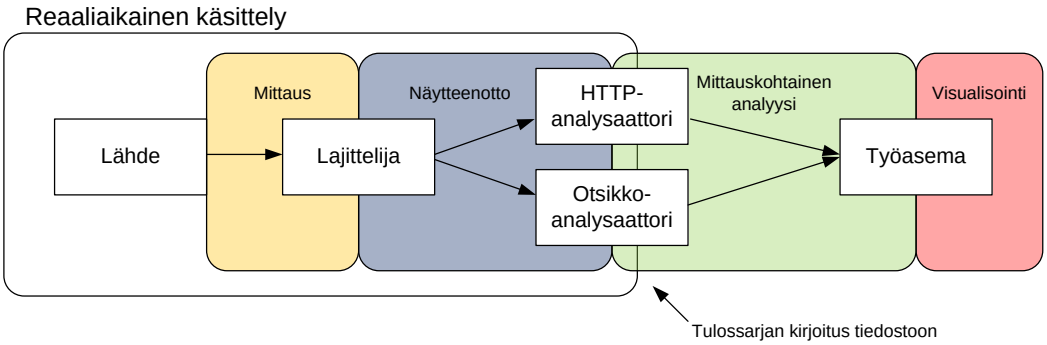
Kuva 15: Mittajärjestelmän toteutuksen arkkitehtuuri

Teknisen toteutuksen pohjana on luvussa 3.1.1. esitettyihin mittajärjestelmän osiin. Toteutuksessa on lähdetty siitä ajatuksesta, jokainen näistä osista muodostaa oman moduulinsa: sen voi poistaa, korvata tai toteuttaa uudelleen, ilman, että muihin osiin tarvitsee tehdä muutoksia.

Kuvassa 15 on esitetty mittajärjestelmän toteutuksen arkkitehtuuri. Toteutus koostuu kolmesta tietokoneesta, joista yksi on mittapiste (lajittelija) ja kaksi analysaattoria (HTTP- ja Otsikko-. Järjestelmän sisäinen viestintä on toteutettu Ethernet-verkolla. Ulkoiset rajapinnat ovat verkkoliitäntä ja tiedostot mitattavalle liikenteelle sekä tulossarjat tiedostoina analysaattoreista jatkokäsittelyä varten.

Mittaus- ja testiympäristö on toteutettu VirtualBox virtualisointiratkaisun avulla. Lajittelija ja analysaattorit ovat virtuaalikoneita ja kaikki kuvassa 15 esitetyt verkot virtuaaliratkaisun sisäisiä.

3.4.1. Työn kulku



Kuva 16: Työn kulku mittajärjestelmän toteutuksessa

Kuvassa 16 on esitetty työn kulku mittausjärjestelmän toteutuksessa. Nimetyt laatikot vastaavat kuvassa 15 samalla tavalla nimettyjä osia. Samoin niiden väliset nuolet ovat aiemmin esitettyjen tietovoiden kanssa yhteneviä.

Mittaus toteutetussa mittausjärjestelmässä alkaa, kun liikenne saavuttaa lajittelijan verkkosovittimen tai kun mittauskohde luetaan lajittelijaohjelmistoon suoraan tiedostosta. Lajittelijaohjelmisto tekee myös ensimmäisiä vaiheita näytteenotosta, kun se pilkkoo PDU:ta osia lähetettäväksi eteenpäin otsikkoanalysaattorille. Varsinaisen näytteenoton tekevät analysaattoriohjelmistot, kun ne eristävät tutkittavat parametrit mittaustuloksista eli lajittelijan välittämästä liikenteestä.

Merkittävä tapahtuma työn kulussa on hetki, kun analysaattoriohjelmistojen tuloste ohjataan tiedostoon. Siihen asti käsittely on ollut reaaliaikaista. Osa käytössä olevista analyysimenetelmistä vaativat rajatun ja staattisen lähtöjoukon, jolloin reaaliaikaisuuden toteutus on mahdotonta.

Analysaattoriohjelmistojen voi katsoa tekevän pientä osaa mittauskohtaisesta analyysistä, mutta varsinaisesti se jakautuu skripteillä automatisoituun menetelmiin analysaattoreissa ja, tekniikasta riippuen, myös työasemassa.

Visualisointi on pääasiallisesti manuaalisesti tehtävää tiedon esitystavan muuttamista. Se tehdään työasemassa tai muussa tutkijan valitsemassa järjestelmässä.

3.4.2. Lähde

Kuten on todettu aikaisemmin: lähde on se osa mittausjärjestelmää, joka tuottaa mitattavan liikenteen. Oikeassa käyttötapauksessa lähteenä olisi mitattava verkko, johon lajittelija on liitetty jollain luvussa 2.4.2. esitetyistä tavoista. Verkkoon voidaan tuoda liikennettä vanhemmasta tallenteesta, esimerkiksi tässä työssä käytetyn tcpreplay-ohjelman avulla.

Tässä työssä lähde sijaitsee samassa virtuaalikoneessa kuin lajittelija. Lähteenä olevaa liikennettä on sekä injektoitu koneen verkkosovittimeen että luettu suoraan tallennettua liikennettä tiedostosta. Pääasiallisesti käytössä on ollut jälkimmäinen tapa, joka oli käytetyllä laitteistolla selvästi nopeampi - tarkkoja suorituskyky-mittauksia ei tehty.

Lähde voisi olla myös toinen verkkolaite, jolloin kyseessä olisi aktiivinen mittaus. Tällaisen liikenteen vastaanottamiselle ei ole teknistä estettä, mutta tämän työn puitteissa mittausjärjestelmään ei ole luotu toiminnallisuutta, joka mahdollistaisi aktiivisten mittausten tekemisen.

3.4.3. Lajittelija

Lajittelijan tehtävä on yksinkertainen: se käy läpi verkkoliikennettä, muuttamatta sitä, ja valituin kriteerein välittää osia havaitusta verkkoliikenteestä analysaattoreille. Tämä toteutuu ottamalla käsittelyyn jokaisen määritellyssä verkkosovittimessa havaitun paketin ja vertaamalla sitä käytössä oleviin kriteereihin, joilla mittaustuloksia välitetään analysaattoreille.

Nyt vertailu tapahtuu ohjelman käyttäjätilassa, mutta libpcapilla olisi mahdollista luoda ydintason pakettisuotimeen uusia määreitä. Tätä mahdollisuutta ei käytetä, koska sen käyttö estäisi joidenkin pakettien pääsyn itse ohjelmaan. Tällöin ei olisi mahdollista esimerkiksi toteuttaa mittaustuloksia kahdelle analysaattorille, joiden määreet liikenteelle ovat ristiriitaiset keskenään.

Lajittelija välittää mitatun liikenteen kahdella tavalla. Ensimmäinen on valittujen pakettien kopioiminen toiselle verkkosovittimelle. Haittana on, että vastaanottajan ja välissä olevien verkkolaitteiden pitää toimia siten, että kaikki paketit välitetään ja otetaan vastaan. Käytännössä siis analysaattorilla pitää olla käytössä verkkokortilla ns. "promiscuous mode".

Toinen tapa on lähettää paketeista valitut osat normaalina verkkoliikenteenä eteenpäin. Valittu tekniikka on UDP, koska silloin liikenteen ylimääräinen osa on mahdollisimman pieni. Virheenkorjausta ei tehdä, mutta siirtoon käytetään sille varattua yksinkertaista verkkoa, jolloin virheet voidaan olettaa olemattomiksi.

Ensimmäistä tapaa käytetään analyysitavoille joille on tärkeää, että koko paketti on käsiteltävissä. Tällaisia tapauksia ovat tilallisesti usealla protokollatasolla liikennettä analysoivat työkalut, kuten IDS ja tässä työssä käytettävä HTTP-analysaattori. Toinen tapa sopii analysointitavoille, jotka keskittyvät vain tiettyihin kenttiin tai tunnuslukuihin liikenteessä. Tällöin on perusteltua, että lajittelija valitsee erikseen analysaattorille lähetettävät parametrit tai osat havaituista PDU:ista.

Tässä versiossa lajittelija ottaa vastaan rajallisesti komentoja komentoriviltä. Sille voi määrittää TCP-portin, jonka mukaan paketit kopioidaan määriteltävälle verkkosovittimelle ja IP-osoitteen sekä UDP-portin, jonne pakettiotsikot lähetetään. Lähteeksi voi valita halutun verkkosovittimen tai tiedoston. Lähteen valinnat ovat toisensa poissulkevia

Lajittelijan ohjelmoinnin lähtökohtana libpcapin osalta on ollut Tim Carstensin kirjoittama artikkeli *Programming with pcap* [37] ja Haking lehden artikkeli [23]. UDP asiakasosuus on tehty Gunnar Gunnarssonin oman opinnäytetyönsä ohessa luodun esimerkin/muistilistan pohjalta. [38] Lajittelijan lähdekoodi on esitetty liitteessä 1.

3.4.4. Analysaattori

Analysaattori on ohjelma, joka ottaa vastaan toista lajittelijan tarjoamista syötteistä, suodattaa ja tekee näytteenoton, mittauskohtaista analyysiä ja tulostaa vakiomuotoiltuna tekstimuotoisena taulukkona tulossarjat.

Osa jatkokäsittelystä voidaan tehdään analysaattoreissa, mutta teknisesti se voidaan siirtää myös toiseen laitteeseen, jonka käyttöjärjestelmä on sopiva käytetyille työkaluille. Tässä luvussa näiden työkalujen käyttö kuvaillaan aliluvussa "Jatkokäsittely".

3.4.4.1. HTTP-analysaattori

HTTP-analysaattori ottaa vastaan lajittelijalta liikenteestä ne paketit, joiden protokolla on 6 (TCP) ja kohde- tai lähdeportit 80. Analysaattori tarkistaa nämä ehdot jokaisesta vastaanottamastansa paketista, joten sen voi myös liittää mitattavaan kohteeseen itsenäisenä osana.

Analysaattoriohjelma perustuu lajittelijan varhaisemman version lähdekoodiin. Myös sen paketinkaappauksesta vastaa libpcap. Käsiteltävistä paketeista pyritään löytämään HTTP-protokollan pyynnöt, joiden metodi on 'GET' ja niiden vastaukset.

Pyyntöviesti tunnistetaan siitä, että sen alku on muotoa "GET <URI> HTTP/1.". URI:sta erotetaan haettavan tiedoston nimi. URI:n syntaksin mukaan tiedoston nimeä edeltää hakemisto ja sitä voi mahdollisesti seurata kysely (GET pyynnön optiot). Hakemistot, joita seuraa toinen hakemisto (tai tiedosto) seuraa '/'. '?'-merkki aloittaa kyselyosuuden. [39] Näiden pohjalta pyynnön URI:sta erotetaan tiedoston nimeksi se alijono, joka alkaa viimeisen '/'-merkin tai niiden puuttuessa URI:n alun ja päättyy '?'-merkkiin tai URI:n loppumiseen. Näin saatu tiedostonimi tallennetaan paketin otsikkotiedoista lähde- ja kohdeosoitteet ja -porttien kanssa ajonaikaiseen muistiin linkitettyyn listaan.

Vastausviesteiksi tulkitaan ne paketit, joiden lähde- ja kohdeosoitteet ja -portit vastaavat käänteisesti (tallennettu lähdeosoite on kaapatun paketin kohdeosoite ja samoin porttinumeroille) jonkin muistiin tallennetun pyynnön vastaavia. Tällöin tutkitaan vastauksen HTTP otsikkotiedot. Otsikkojen content-length, content-type ja content-range sisällöt tulostetaan yhdessä aiemmin tallennetun tiedoston nimen kanssa. Mikäli content-length (eli tiedoston kokoa) ei ole ilmoitettu otsikkotiedoissa,

tehdään oletus, että tiedosto on kokonaan tässä paketissa ja siten arvio tiedoston kooksi on paketin koko vähennettynä IP ja TCP otsikoiden koolla.

Analysaattori tuottaa syötteen joka sisältää tiedoston nimen ja koon, sekä MIME-tyypin content-range otsikon arvon, mikäli sellainen on löytynyt. Syöte tulostetaan, mutta se on mahdollista ohjata tiedostoon tai käsiteltäväksi käyttänen käyttöjärjestelmän putkia.

HTTP-analysaattorin lähdekoodi on esitetty liitteessä 2.

3.4.4.2. Otsikkoanalysaattori

Otsikkoanalysaattori ottaa vastaan lajittelijan lähettämää otsikkosyötettä. Ohjelma kuuntelee tiettyä UDP-porttia järjestelmässä. Se ottaa vastaan paketteja, joiden hyötykuormana ovat toisen paketin otsikot.

Itse otsikkoanalysaattorin ohjelmalogiikka on yksinkertainen. Vastaanotetun hyötykuorman alku jäsennetään IP otsikoksi. Mikäli protokollanumero on 6 tai 17 (TCP tai UDP), jäsennetään seuraava pala hyötykuormaa vastaavan protokollan otsikoksi. Otsikoista valitaan IP:n protkollakenttä ja koko paketin koko sekä porttinumerot, mikäli kuljetuskerroksen protokolla niitä käyttää. Ne esitetään yhtenä rivinä. Syöte tulostetaan, mutta se on mahdollista ohjata tiedostoon tai jatkokäsiteltäväksi käyttänen käyttöjärjestelmän putkia.

Aiemmin esitetystä viisikosta on tarkoituksella jätetty lähettäjän ja vastaanottajan IP osoite tulostamatta. Tähän on kaksi syytä: ensimmäinen on se, että valitut analyysimenetelmät eivät vaadi IP-osoitteiden tuntemista ja toinen se, että ne voidaan mieltää tunnistetiedoiksi ja siksi niiden tallentaminen ilman lisämenetelmiä, kuten näennäistä anonyymisointia, ei ole hyvän tavan mukaista. Teknisesti IP-osoitteiden tallentaminen ei ole ongelmallista ja vaatii vain kaksi riviä koodia lisää.

Otsikkoanalysaattorin lähdekoodi on esitetty liitteessä 3. Palvelintotetus perustuu Gunnarssonin työhön. [38]

3.4.5. Jatkokäsittely

Jatkokäsittely jakautuu kahteen vaiheeseen. Ensimmäisessä vaiheessa tulossarjoja käsitellään komentorivityökaluilla puolittain automatisoidusti skripteillä. Näin saadaan pienempiä tulossarjoja, jotka on helpompi visualisoida käyttäen hyväksi haluamiaan työkaluja.

Ensimmäinen vaihe on dokumentoitu kummankin käytössä olevan mittaustavan osalta seuraavissa aliluvuissa. Visualisointiin on käytetty tässä työssä Matlab laskentaohjelmaa ja Excel taulukkolaskentaa. Kummassakaan ei käytetty ohjelmoituja toiminteita. Siksi niiden käyttöä ei ole esitellä tarkemmin.

3.4.5.1. HTTP-analyysi

Analysaattorilta saatuja tuloksia suodatetaan kolmella eri kriteerillä: nimellä, MIME-tyypillä ja <nimi,koko,tyyppi>-kolmikolla. Jokaisen kriteerin kohdalla toistetaan seuraavat toimenpiteet: tuloksista eristetään kriteerin mukaiset uniikit löydöt erilliseen

listaan. Jokaista uniikkia löydöstä kohden aggregoidaan saman kriteerin täyttävien havaintojen lukumäärä ja yhteiskoko. Nämä tiedot tallennetaan uuteen tiedostoon. Tämän lisäksi tuloksista otetaan erilleen tiedoston koot listana. Alkuperäisistä tuloksista ja aggregoiduista listoista voidaan laskea tilastollisia muuttujia siihen tarkoitukseen erikseen tehdyn AWK ohjelman avulla.

HTTP-analysaattorin tuloksien käsittelyyn käytettyjen skriptien lähdekoodit ovat liitteessä 4.

3.4.5.1. Otsikkoanalyysi

Analysaattorin tuottamasta tulossarjasta aggregoidaan kokonaisliikenne (pakettien kokojen summa) ja pakettien lukumäärä erikseen protokollanumeron ja portteja käyttävien protokollien (TCP ja UDP) osalta sekä kohdeportin (ns. upstream) että lähdeportin (downstream) mukaan. Lisäksi mittaustuloksista erotetaan erikseen paketin koot omaksi tiedostokseen. Tämä tehdään AWK ohjelman avulla. Sen lähdekoodi on esitetty liitteessä 4.

Protokolla- ja porttinumerointeihin riippuvissa analyyseissä tarkistetaan ne soveltuvin osin tunnettujen numeroiden listoja vastaan.

4. Tulokset

Työhön liittyvissä mittauksissa, on käytetty mitattavana liikenteenä Funetista (Finnish university network) 21.1.2010 4.16:51-21.57:52 välisenä aikana tallennettua liikennettä. Liikenne tuotiin järjestelmään käyttäen hyväksi lajittelijan mahdollisuutta lukea liikenne suoraan tiedostosta.

4.1. HTTP

4.1.1. Uniikit tiedostot

Kahta eri metodia käytettiin tulkitsemaan tiedostojen identtisyyttä. Ensimmäisessä metodissa kolme tietoa (nimi, koko, MIME tyyppi) pitivät olla identtisiä, että tiedoston katsotaan olevan sama. Toisena metodina käytettiin pelkkää tiedoston nimeä. Jälkimmäinen metodi on valittu vertailukohdaksi ensimmäiselle.

HTTP-analysaattorille asetettiin ajaksi jona se säilöö pyynnöistä tallennetut tiedot muistiin 60 sekuntia. Tällöin analysaattori löysi metodillaan yhteensä 267890 siirrettyä tiedostoa. Ensimmäisellä metodilla näistä oli 137348 uniikkeja. Vastaavasti pelkällä tiedoston nimellä uniikkeja löytyi 80399.

Taulukko 2: Uniikkien tiedostojen tilastollisia perussuureita

	Metodi 1: Uniikit tiedostot	Metodi 2: Tiedoston nimet
Lukumäärä	137348	80399
Keskiarvo	1,95	3,34
Moodi	1	1
Keskiarvon keskihajonta (s)	95,18	66,85
Varianssi (s^2)	9059,76	4469,18
Vaihtelukerroin	48,69	20,02

Taulukossa 2 on esitetty tilastolliset perusmuuttujat havaintojen lukumäärälle per luokka (identtinen tiedosto). Oletuksen mukaisesti vertailumetodilla, jossa oli löysempi kriteeri samankaltaisuudelle, löytyi vähemmän luokkia ja löydöksiä per luokka oli enemmän. Yllättävää sen sijaan oli, että verrokkimetodilla hajonta oli pienempää kuin tarkemmalla metodilla.

Taulukossa 3 on listattu yleisimmät tiedostot kummallekin metodille. "<indeksi>" ilmaisee sellaista pyyntöä, jolla on haettu hakemistoideksiä eli URI:n viimeinen merkki on ollut '/'. Indeksitiedostoja ovat myös index.php ja index.html, mutta niihin viitataan suoraan. __utm.gif on Googlen jäljityskuva. Lisäämällä kuvan verkkosivulle, käyttäjän selain tekee sivulla käydessä myös pyynnön Googlen palvelimelle hakeakseen kuvan. Tästä pyynnöstä jää jälki ja siten Google pystyy seuraamaan, tietyin rajoituksin, selaajan toimia.

Taulukko 3: Kymmenen yleisintä tiedostoa eri metodeilla

	Metodi 1: Uniikit tiedostot		**Metodi 2: Tiedoston nimi**	
Sija	**Tiedosto**	**Lkm**	**Tiedosto**	**Lkm**
1	imp	3564	<indeksi>	16199
2	<indeksi>	3083	imp	4381
3	webSearch	2166	webSearch	3541
4	__utm.gif	2122	__utm.gif	2239
5	ping	1812	search	2235
6	stellenangeboteFinden.html	1327	ping	1870
7	ip.pl	978	ip.pl	1701
8	recommend.php	892	ads	1694
9	translate_p	862	index.php	1665
10	view.asp	834	index.html	1643

Mittauksessa havaittujen HTTP:lla siirrettyjen tiedostojen kokonaiskoko on noin 36 teratavua. Jos välimuistin käyttö olisi täydellistä eli jokainen tiedosto kulkisi mittapisteen kautta vain kerran, säästettäisiin lähes yhdeksän teratavun edestä siirtokapasiteettia mittausajalta eli noin 12Tt vuorokaudessa.

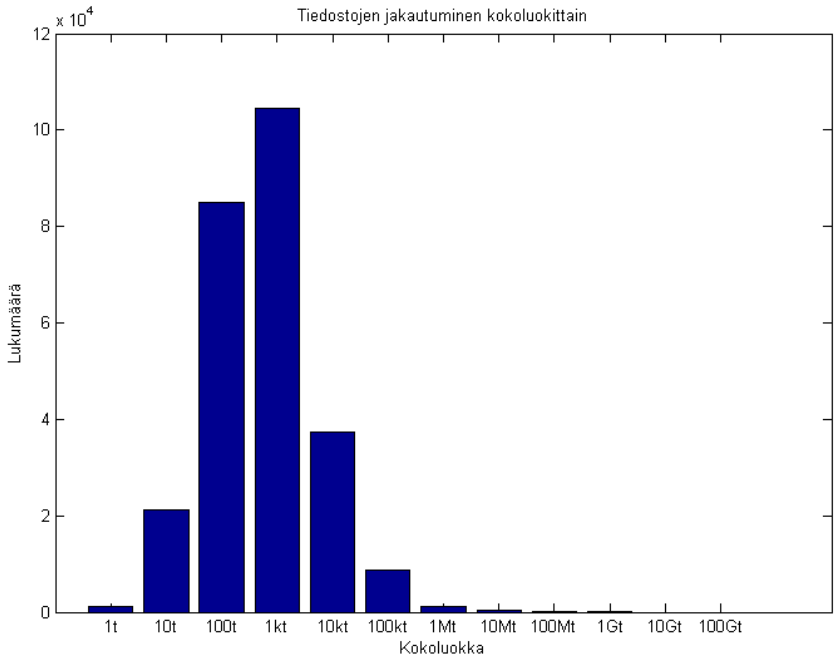
4.1.2. Tiedoston koot

HTTP-analysaattorin tuloksista analysoitiin myös siirrettyjen tiedostojen kokoja, ilman että niitä on ensin (dis)aggregoitu. Ennen käsittelyä on kuitenkin ne tiedostot, joiden koko on 0 poistettu listasta. Niitä oli yhteensä 8690. Olemattoman kokoiset tiedostot johtuvat palvelimen asettamasta vastauksen otsikosta. Taulukossa 4 on siirrettyjen tiedostojen koille laskettu tilastollisia muuttujia.

Taulukko 4: HTTP:llä siirrettyjen tiedostojen koon tilastollisia muuttujia

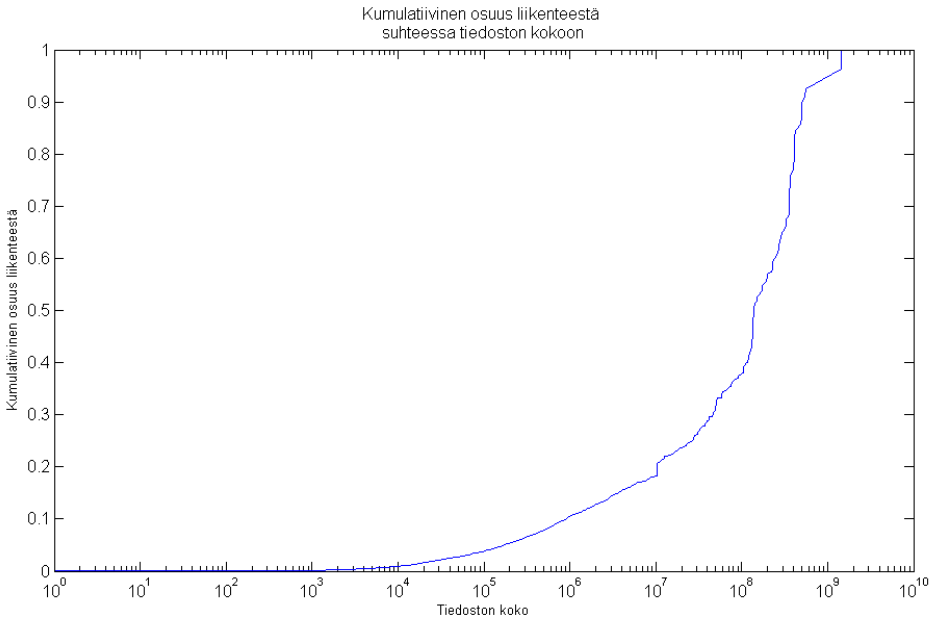
Keskiarvo	144,39	kt
Mediaani	1,41	kt
Keskiarvon keskihajonta (s)	6,11	Mt
Varianssi (s^2)	41065300000000	
Vaihtelukerroin	22071,27	

Erittäin suuri varianssi ja suhteellinen ero mediaanin ja keskiarvon välillä kertoo suurista ääriarvoista. Kuvassa 17 on histogrammi siirrettyjen tiedostojen koista. Luvussa 2 esitetyt metodit eivät sopineet ääriarvojen erittäin suuren eron vuoksi. Korien kooksi on valittu kasvava noin 10-logaritminen asteikko. Erona on se, että joka kolmas porras on 10-kertaisen sijaan 10,24-kertainen eli siirrytään tavuista kilotavuihin, jne; korit ovat tällöin helpommin ymmärrettävissä.



Kuva 17: HTTP:llä siirrettyjen tiedostojen lukumäärä kokoluokittain

Kuvaajasta nähdään, että lukumäärältään suurin osa on kooltaan hyvin pieniä tiedostoja. Voidaan olettaa, että yhden ja kymmenen tavun kategorian sekä osa 100t kategorian sijoittuvista havainnoista johtuu palvelimen hyvän tavan ja protokollan vastaisesta toiminnasta. Vaikka se jätettäisiin huomioimatta, on valtaosa tiedostoista alle sadan kilotavun kokoisia.



Kuva 18: Kumulatiivinen summa liikenteestä suhteessa tiedoston kokoon

Käänteinen ilmiö on nähtävissä tiedoston koon suhteen. Kuvassa 18 on kuvattu kumulatiivinen summa tiedoston koon funktiona järjestetystä sarjasta. Käytännössä

yli 10Mt tiedostot vastaavat lähes koko liikenteestä tavumääräisesti. Kuvaajan loppupäästä nähdään vielä pari jyrkkää nousua, jotka johtuvat muutamasta hyvin suuresta tiedostosta.

Kun kuvaajia verrataan toisiinsa huomataan, että niiden painopiste on erilainen: pieniä tiedostoja siirretään erittäin paljon, mutta niiden osuus siirrettyjen tavujen määrästä on häviävän pieni. Käänteisesti suuria tiedostoja siirretään huomattavasti vähemmän, mutta muodostavat valtaosan siirretyistä tiedostoista tavuissa.

Sama dispariteetti on havaittavissa myös analysoitaessa MIME-tyyppien jakatumista. Silloin saadaan myös tarkempaa tietoa siitä, minkälaista dataa molempien ryhmien tiedostot pääasiallisesti ovat.

4.1.1. MIME-tyypit

Tässä työssä analysointiin myös eri MIME-tyyppien osuuksia kaapatusta liikenteestä. Jokaista uniikkia tyyppiä kohti on aggregoitu analysaattorin tuloksista frekvenssi ja tiedostojen kokonaiskoko eli liikennemäärä. Taulukossa 4 on esitetty molemmille muuttujille tilastollisia muuttujia.

Taulukko 5: MIME tyyppit lukumäärän ja koon mukaan - tilastolliset muuttujat

	Lukumäärä	Liikennemäärä	
Keskiarvo	5634	757,49	Mt
Mediaani	10	285,4	kt
Keskiarvon keskihajonta (s)	32683,33	4458,37	Mt
Varianssi (s^2)	1068200000	2,18551E+19	
Vaihtelukerroin	5,80	6027,00	

Huomionarvoista on liikennemäärän merkittävän suuri vaihtelukerroin ja se, että keskiarvo on noin yli kaksituhattakertainen verrattuna mediaaniin. Tästä voidaan päätellä, että suurimmaksi osaksi tyyppien siirtomäärät ovat maltillisia, mutta muutamassa tyypissä on kokonaisliikenteeltään huomattavan suuri. Sama ilmiö on nähtävissä pienempänä myös tyyppien frekvenssissä.

Taulukossa 5 on listattu kummallakin kriteerillä kymmenen yleisintä MIME-tyyppiä.

Kymmenen yleisintä tyyppiä lukumäärän mukaan vastaavat noin 93,72% kaikkien siirrettyjen tiedostojen lukumäärästä. Vastaavat luvut yhteenlaskettujen tiedostojen kokojen mukaan lajitelluille tyypeille on 89,02% ja 25,00%. Tämä vahvistaa aikaisemmin tehtyjä oletuksia.

Tämä dispariteetti johtuu todennäköisesti siitä, että HTTP:tä käytetään luonteeltaan varsin erilaisen datan siirtämiseen. Normaalit www-sivut ovat vain tekstimuotoisia dokumentteja, mutta myös tiedostoja, jotka ovat luonteeltaan varsin erilaisia kuten kuvia, ääntä ja videota.

Taulukko 6: 10 yleisintä MIME-tyyppiä lukumäärän ja koon mukaan

	Lukumäärä	%	Liikennemäärä	%
1	text/html	35,77	video/x-msvideo	39,95
2	image/jpeg	16,92	application/octet-stream	14,66
3	mage/gif	11,23	application/x-gzip	9,40
4	image/gif	11,23	application/pdf	8,49
5	text/plain	4,36	video/x-flv	4,55
6	application/ x-javascript	3,69	text/plain	3,15
7	image/png	3,19	video/x-ms-wmv	2,85
8	application/pdf	2,90	image/jpeg	2,21
9	text/xml	2,48	audio/mpeg	2,13
10	text/css	1,96	application/ vnd.ms-powerpoint	1,64

4.2. Otsikkoihin perustuva analyysi

4.2.1. Liikenne kuljetuskerroksen protokollittain

Taulukossa 7 on esitetty liikenteen jakautuminen protokollittain. Vertailusuureena on käytetty sekä pakettien lukumäärää, että protokollan osuutta kokonaisliikennemäärästä (tavuina).

Taulukko 7: Liikenne protokollittain

Protokolla-numero	Protokolla	Paketteja	%	Liikennemäärä (tavua)	%
1	ICMP	68113	0,04	1933043513	0,03
6	TCP	162377266	99,90	5551680000000	99,93
17	UDP	91817	0,06	1948486036	0,04

Suurin osa liikenteestä koostuu TCP:stä. Sen lisäksi havaittiin vain kahta muuta protokollaa, UDP:tä ja ICMP:tä. Niiden osuus kummankin suureen mukaan oli vain noin puoli promillea tai alle. Mikäli tarkoitus on tarkastella koko liikennettä tarkastelevia ilmiöitä, on niiden vaikutus häviävän pieni.

4.2.2. Pakettien koot

Pakettien koista lasketut tilastolliset muuttujat ovat taulukossa 8. Keskiarvon keskihajonta kattaa lähes koko mahdollisen koon (1-65536), jota rajoittaa IP otsikon 16 bitin kokoinen kenttä. Voidaan siis olettaa, että paketin koon jakauma on tasainen tai ainakin hyvin lavea.

Taulukko 8: Pakettien kokojakauma - tilastolliset muuttujat

Keskiarvo	34180,30
Mediaani	32258
Keskiarvon keskihajonta (s)	20526,74
Varianssi (s^2)	421347000
Vaihtelukerroin	0,60

4.2.3. Liikenne porteittain

4.2.3.1. TCP

Taulukko 9: Liikenne porteittain, 10 huippuporttia: TCP, lähdeportti

	Liikenteen määrä			Pakettien lukumäärä		
#	**Portti**	**Protokolla**	**%**	**Portti**	**Protokolla**	**%**
1	443	HTTPS	33,14	443	HTTS	27,89
2	80	HTTP	29,32	80	HTTP	20,72
3	9001	TOR	2,66	9001	TOR	3,03
4	44282		1,95	44282		1,55
5	46082		1,93	5001	Iperf	1,44
6	60934		1,61	46082		1,18
7	37343		1,54	33442		1,01
8	33442		1,40	60934		0,98
9	36652		0,93	37343		0,94
10	5001	Iperf	0,58	36652		0,68

Taulukot 9 ja 10 ovat samanlaisia: niissä on esitetty kymmenen yleisintä porttia ja niiden osuus liikenteestä sekä pakettien lukumäärällä että liikenteen määrällä laskettuna. Tunnetuille porteille on lisäksi esitetty ylemmän tason protokollat. Taulukkoon 9 on analysoitu liikenne lähdeporttien mukaan ja taulukkoon 10 kohdeporttien mukaan.

Taulukoista nähdään, että liikennettä dominoivat portit 80 ja 443 (HTTP ja HTTPS). Syy miksi osuus on selkeästi suurempi, kun ne ovat lähdeportteja eli silloin kun liikenne tulee palvelimelta asiakkaalle, johtuu HTTP protokollan asymmetrisuudesta: Pyynnöt ovat suhteellisen lyhyitä. Vastaukset ja niiden sisältämät tiedostot ovat huomattavan suuria ja jakautuvat useisiin paketteihin.

Taulukko 10: Liikenne porteittain, 10 huippuporttia: TCP, kohdeportti

	Liikenteen määrä			Pakettien lukumäärä		
#	**Portti**	**Protokolla**	**%**	**Portti**	**Protokolla**	**%**
1	443	HTTPS	16,67	443	HTTS	23,06
2	80	HTTP	5,46	80	HTTP	12,31
3	5001	TOR	5,07	9001	TOR	3,28
4	9001	Iperf	3,32	5001	Iperf	3,10
5	54546		1,98	44282		1,27
6	42429		1,93	54546		1,19
7	9006	SCTP	1,40	42429		1,17
8	53095		1,39	9006	SCTP	1,01
9	44282		1,22	53095		0,97
10	59562		0,91	59562		0,62

4.2.3.2. UDP

Taulukko 11: Liikenne porteittain, 10 huippuporttia: UDP, lähdeportti

	Liikenteen määrä			Pakettien lukumäärä		
#	**Portti**	**Protokolla**	**%**	**Portti**	**Protokolla**	**%**
1	27866		1,25	27866		1,70
2	6881	Bittorrent	1,01	31958		1,12
3	42383		1,00	123	ntp	1,09
4	123	ntp	1,00	27226		1,07
5	31958		0,82	6881	Bittorrent	0,97
6	36624		0,61	42383		0,79
7	27226		0,54	8566		0,78
8	22053		0,47	22053		0,64
9	8566		0,43	32229		0,52
10	32229		0,38	36624		0,47

Taulukossa 11 ja 12 on esitetty, taulukkojen 9 ja 10 tapaan, kymmenen yleisintä porttia ja niiden osuudet liikenteestä liikenteen määrän ja pakettien lukumäärän mukaan. Kuljetuskerroksen analysoitavana protokollana on UDP.

Liikenne on sirpaleisempaa kuin TCP:n tapauksessa. Eri porttien osuudet ovat kummassakin tapauksessa suhteellisen pieniä. Kymmenestä yleisimmästä portista voidaan kuitenkin nähdä jotain yleisiä käyttökohteita UDP:lle, kuten NTP (Network time protocol), liikenne erään tiedostonjako-ohjelman (Bittorrent) vakioporttiin ja merkittävä määrä liikennettä porttiin 8080, joka TCP:nä olisi varattu vaihtoehtoiseksi HTTP portiksi, mutta on haittaohjelmien käytössä UDP:nä.

Taulukko 12: Liikenne porteittain, 10 huippuporttia: UDP, kohdeportti

	Liikenteen määrä			Pakettien lukumäärä		
#	**Portti**	**Protokolla**	**%**	**Portti**	**Protokolla**	**%**
1	34136		5,69	34136		6,54
2	26043		4,36	51413		4,90
3	51413		3,96	26043		4,03
4	8080	Troijalainen?	3,73	8080	Troijalainen?	3,97
5	21911		2,34	21911		2,10
6	42326		2,26	42326		2,05
7	28018		2,08	28018		1,92
8	48912		1,72	48912		1,57
9	6881	Bittorrent	1,71	6881	Bittorrent	1,57
10	13357		1,37	13357		1,36

5. Johtopäätökset

5.1. Mittausjärjestelmä

Johtopäätöksenä on se, että on mahdollista, tietyissä tapauksissa suositeltavaa, toteuttaa yksinkertainen hajautettu reaaliaikainen tietoverkon liikenteen mittausjärjestelmä, jolla voidaan mitata tietoverkon liikennettä ja analysoida sitä käyttäen monia erilaisia menetelmiä.

Mittausjärjestelmälle asetetut kriteerit on tavoitettu kohtalaisesti tai hyvin: Modulaarisuus on saavutettu suunnitellusti ja sopivalla tarkkuudella. Kustannustehokkuus on erinomainen; kaikki käytetyt työkalut ovat ilmaisia tai vapaasti saatavilla. Laitteiston käyttöaste on hyvä virtuaalisoinnin ansiosta.

Käytettävyys on kohtalainen ja saavutettu huonoiten neljästä kriteeristä: järjestelmä tarjoaa vahvan muokattavuuden, mutta se pitää tehdä lähdekooditasolla. Se onkin parhaimmillaan kehitysalustana uusien mittausjärjestelmille.

Tietoturvan kolmesta osa-alueesta on kaksi toteutettu vahvasti: mittausjärjestelmä aiheuttaa mahdollisimman pienen vaikutuksen verkon saatavuuteen. Luottamuksellisuus on tavoitettu hyvin, mitään luottamuksellista tietoa ei mittausjärjestelmän analysaattoreista päästetä edelleen. Menetelmiä tiedon eheyden varmistamiseksi ei ole otettu käyttöön.

5.1.1. Jatkotutkimuskohteet ja suositukset

Kehittämismahdollisuuksia on eniten mittausjärjestelmän keskeisissä osissa: lajittelijassa ja analysaattoreissa. Tällä hetkellä teknisesti täydellisin niistä on lajittelija. Se toimii, kuten on tarkoitettu ja sen koodirakenne sallii myös helposti toiminnallisuuden lisäämisen. Lajittelijan komentorivillä tehtäviä toimenpiteitä voi laajentaa ja samoin myös sallia mielivaltaisen määrän erilaisia suodatusehtoja.

Otsikkoanalysaattorin kaappaamia parametreja voisi laajentaa kattamaan koko viisikon. Koodirunko sallii myös pakettikohtaisen analyysin käytön. Lisäksi käyttöön voisi ottaa aikaleimat lajittelijan analysaattorille lähettämissä otsikoissa. Näiden toiminnallisuuksien tarpeellisuus riippuu siitä, mihin mittausjärjestelmää käytettään.

HTTP-analysaattorin algoritmi ei tällä hetkellä ole välttämättä paras mahdollinen. Kuten tuloksissa huomattiin, parempi algoritmi myös tarkoittaa vähemmän löydettyjä identtisiä tiedostoja. Kehitysmahdollisuuksia ovat hajakoodausalgoritmien käyttö tiedoston sisällön tallentamisessa ja niiden vertailussa sekä TCP-sekvenssinumeroiden tarkasteleminen pyynnön ja vastauksen parittamisessa.

5.2. Mittaukset

5.2.1. HTTP-analyysi

HTTP liikenteen analysoinnissa havaittiin, että merkittävä määrä siirretyistä tiedostoista on identtisiä muiden siirrettyjen tiedostojen kanssa. Tämä tarkoittaa sitä, että osa liikenteestä on redundanttia. Identtiset tiedostot määrittelevä algoritmi vaikuttaa niiden lukumäärään. Tarkistuskriteerejä lisäämällä ja tiukentamalla saadaan parempi tulos identtisten tiedostojen lukumäärästä, joka on myös pienempi.

Osa tiedostoista on identtisiä ja verkkoliikennettä voi vähentää erilaisin välimuistiratkaisuin. Siirretyillä tiedostoilla voi olla myös merkityksiä, joita ei analysoitavasta liikenteestä voi tulkita. Esimerkiksi liikenteestä havaittiin merkittävä määrä identtisiä kuvia, joita Google käyttää tosiasiallisesti verkkosivujen käyttäjien tilastointiin.

5.2.2. Otsikkoanalyysi

TCP:n jakautuminen on hyvin painottunutta muutamaan protokollaan. HTTP ja sen salattu variantti HTTPS olivat kaikilla listoilla 1. ja 2. sijalla. Suurempi etumatka lähdeporttien kohdalla johtuu protokollien luonteesta: sanomat palvelimelta asiakkaalle ovat suurempia ja jakautuvat useampaan pakettiin kuin toisinpäin.

UDP:n jakatuminen porteittain on hyvin sirpaleista. Yksittäisen portin osuus, suunnasta ja laskutavasta riippumatta, on vain muutamia prosenttiyksikköjä. Yleisimpien porttien joukosta oli kuitenkin mahdollista erottaa joitakin käyttötapauksia ja muutama protokolla.

5.2.3. Jatkotutkimuskohteet ja suositukset

Työssä käytetyllä laitteistolla ja menetelmillä ei kyetty tekemään riittävää analyysiä paketin koon jakaumasta. Tämä johtui siitä, että tulossarjassa oli enemmän alkioita kuin oli mahdollista käsitellä. Tuloksien saamiseksi olisi suositeltavaa käyttää muita menetelmiä tai tehokkaampaa laitteistoa.

HTTP:n ja HTTPS:n dominanssi TCP porttiavaruudessa ja UDP:n porttiavaruudessa havaitut käyttötapaukset ovat löytöjä, joista voi löytyä mielenkiintoisia jatkotutkimuskohteita.

6. Yhteenveto

Kiinnostusta tietoverkon liikenteen mittaamiseen on ollut lähes siitä asti, kun tietoverkkoja on ollut olemassa. TCP/IP protokollaperhe ja sen ympärille rakentunut ekosysteemi on de facto toteutus tietoverkolle. Sille on kattava dokumentaatio, mikä tekee omien työkalujen kehittämisen helpoksi.

Mittausjärjestelmä koostuu neljästä osasta: lähteestä, joka tuottaa analysoitavan liikenteen, mittauksesta, jossa liikenne suodatetaan, analysoinnista, joka vastaa näytteenotosta ja mittauskohtaisesta analyysistä, ja visualisoinnista, jolloin analysoinnin tulokset tuotetaan ihmisen ymmärrettävään muotoon.

Mittausjärjestelmän toteutuksen kriteereinä ovat modulaarisuus, kustannustehokkuus, käytettävyys ja tietoturva. Niiden pohjalta on tehty suunnitelma mittausjärjestelmäksi. Käytännön työ oli sen toteutus. Järjestelmä luotiin virtuaalisoituun ympäristöön. Se koostuu kolmesta virtuaalikoneesta sekä niiden liitäntäpisteistä. Yksi koneista on mittapiste, joka suodattaa liikennettä ja välittää sitä edelleen. Kaksi muuta ovat analysaattoreita, joista kumpikin käyttävät eri menetelmiä liikenteen analysointiin.

Otsikkoihin perustuvassa analysoinnista jokaisesta paketista kaapataan vain otsikkotiedot, niistä otetaan talteen IP otsikon protokollanumero ja paketin kokonaiskoko sekä, jos kuljetuskerroksen protokolla sen sallii, myös porttinumerot. Liikenne aggregoidaan portteihin ja protokollanumeroon. Erikseen analysoidaan pakettien kokojen jakauma.

Toisessa analysaattorissa on kohteena HTTP. Siinä pyritään etsimään liikenteestä pyyntö-vastaus-pareja, joista saadaan kerättyä siirrettyä tiedostoa koskeva yksilöivät tiedot: nimi, koko ja sisältötyyppi. Niistä tutkitaan identtiset tiedostot, aggregoidaan tiedostojen lukumäärä ja liikenteen kokonaismäärä erikseen sisältötyypin ja koon mukaan.

Mittausjärjestelmää on käytetty analysoimaan annettu liikennetallenne. Tuloksissa havaittiin, että merkittävä osa HTTP:llä siirretyistä tiedostoista on identtisiä ja että niiden luonne on hyvin dualistinen. Lukumääräisesti hyvin pienet tiedostot vastasivat suurimmasta osasta liikennettä. Liikenteen määrällä laskettuna merkitsevässä asemassa oli muutama erittäin suuri siirretty tiedosto.

Otsikkoanalyysin tulosten pohjalta voidaan todeta, että pakettikoko on lähes tasajakautunut. Suurin osa liikenteestä on TCP liikennettä ja suurin osa siitä on HTTP:tä ja sen salattua varianttia HTTPS:ä. Myös Tor-verkon liikenne on esillä. UDP liikenteen jakautuminen eri porteille on hyvin tasaista eikä samanlaisia ilmiöitä ollut havaittavissa.

Tämän työn merkittävin tulos on toteutettu mittausjärjestelmä ja sen suunnittelun ja toteuttamisen aikana kerätty tieto. Ne antavat hyvät oman mittausjärjestelmän luomiseen tai nyt toteutetun jatkokehitykseen.

Lähdeluettelo

[1] Jarno Yliluoma, ""Rahat vai kolmipyörä" - Tieteellinen kirjoitusprosessi diplomityön tekijän näkökulmasta," Aaltoyliopisto, Espoo, 2011.

[2] Leonard Kleinrock, "Information flow in large communication nets," *RLE Quaterly Progress Report*, Heinäkuu 1961.

[3] Kleinrock Leonard, *Communication nets: Stochastic message Flow and delay*.: McGraw-Hill, 1964.

[4] Barry M. Leiner et al., "A brief history of the Internet," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 5, Lokakuu 2009.

[5] Lawrence Roberts and Thomas Merrill, "Toward a cooperative network of time-shared computers," in *Fall AFIPS-konferenssi*, 1966.

[6] R. Braden, *RFC 1122: Requirements for Internet Hosts -- Communication Layers*.: Internet Engineering Task Force, 1989.

[7] ISO/IEC, *2382-26: Open systems Interconnection*., 1993.

[8] Information Sciences Institute, University of Southern California, *RFC 791: Internet Protocol DARPA Internet program protocol specification*., 1981.

[9] J. Arkko, Brandmer S., and Ericsson. (2008, Helmikuu) RFC5237: IANA Allocation Guidlines for the PRotocol Field.

[10] Deering S. and Hinden R. (1998) RFC 2460: Internet Protocol, Version 6 (IPv6) specification.

[11] Lucie Smith and Ian Lipner. (2011, Helmikuu) Free Pool of IPv4 Address Space Depleted. [Online]. http://www.nro.net/news/ipv4-free-pool-depleted

[12] Information Sciences Institute, University of Southern California. (1981, Syyskuu) RFC 793: Transmission control protocol.

[13] IANA. (2011, Huhtikuu) Port numbers. [Online]. http://www.iana.org/assignments/port-numbers

[14] Postel J. (1980, Elokuu) RFC 768: User Datagram Protocol.

[15] R. Fielding et al. (1999, Kesäkuu) Hypertext Transfer Protocol -- HTTP/1.1.

[16] A. Barth. (2011, Tammikuu) HTTP State Management Mechanism (draft).

[17] Dept. of Electrical Engineering, University of Delaware. (1982, Elokuu) RFC 822: Stadard for the format of ARPA Internet text messages.

[18] J. Postel. (1994, Maaliskuu) Media Type Registration Procedure.

[19] Alliance of Telecommunications Industry Solutions. (2007) ATIS Telecom Glossary 2007.

[20] Patri Arlos, *On the Quality of Computer network measurements*.: Blekinge Institute of Technology, 2005.

[21] IEEE. (2006) 802.1Q-2005: IEEE Standard for Local and metropolitan area networks - Virtual Bridged Local area networks.

[22] J. Case, M. Fedor, M. Schoffstall, and Davin J. (1990) RFC 1157: A Simple Network Management Protocol (SNMP).

[23] Luis Martin Garcia, "Programming with Libpcap - Sniffing the Network From Our Own Application," *Haking*, vol. 2008, no. 2, pp. 38-46.

[24] Mark L. Woodward. (2007) The Myth of Fife Nines. [Online]. http://www.mohawksoft.org/?q=node/38

[25] Pertti Laininen, *Todennäköisyys ja sen tilastollinen soveltaminen*, Viides korjattu painos ed., 2001.

[26] Nevil Brownlee and Margaret Murray, "Streams, Flows and Torrents," 2001.

[27] M. Luoma, and M. Peuhkuri. M. Ilvesmäki. (2007) Course textbook (draft) for the S-38.3183 Internet traffic measurements and measurement analysis course. [Online]. http://www.netlab.hut.fi/opetus/s383183/k07/

[28] Ilkka Mellin. (2011, Toukokuu) Ennustaminen ja aikasarja-analyysi: Aikasarja luentokalvot. [Online]. http://www.sal.tkk.fi/vanhat_sivut/Opinnot/Mat-2.128/IMluennot/EAAS100.pdf

[29] Ilkka Mellin, *Todennäköisyyslaskenta*, Teknillinen korkakoulu, Ed., 2006.

[30] Tilastokeskus. (2011) Johdatus tilastolliseen ajatteluun. [Online]. http://www.stat.fi/tup/verkkokoulu/data/tt/02/09/index.html

[31] Jon Silander, *Henkilötietojen hallinta tietoverkoissa*.: Teknillinen Korkeakoulu, 2008.

[32] Suomen Laki, *Sähköisen viestinnän tietosuojalaki 16.6.2004/516*., 2004.

[33] Apache Software Foundation. Apache Module mod_rewrite. [Online]. http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

[34] Linux Kernel Organization. The Linux Kernel Archives. [Online]. http://www.kernel.org

[35] Free software foundation. The Gnu Operating System. [Online]. http://www.gnu.org

[36] Canonical Ltd. ubuntu. [Online]. http://www.ubuntu.com/

[37] Tim Carstens. Programming with pcap. [Online]. http://www.tcpdump.org/pcap.htm

[38] G Gunnarsson. Writing a simple client/server in a Unix environment. [Online]. http://www.abc.se/~m6695/udp.html

[39] T Berners-Lee, R Fielding, and L Masinter. (2005) RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. [Online]. http://tools.ietf.org/html/rfc3986

Liite 1: Lajittelijan lähdekoodi

```
#include <pcap.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>
#include <net/ethernet.h>
#include <netinet/ether.h>
#include <netinet/tcp.h>
#include <string.h>

/*Structit*/
struct tcpdump_ip
{
   u_int8_t ip_vhl;    /* header length, version */
#define IP_V(ip)  (((ip)->ip_vhl & 0xf0) >> 4)
#define IP_HL(ip) ((ip)->ip_vhl & 0x0f)
   u_int8_t ip_tos;    /* type of service */
   u_int16_t   ip_len;         /* total length */
   u_int16_t   ip_id;          /* identification */
   u_int16_t   ip_off;         /* fragment offset field */
#define  IP_DF 0x4000                /* dont fragment flag */
#define  IP_MF 0x2000                /* more fragments flag */
#define  IP_OFFMASK 0x1fff           /* mask for fragmenting bits */
   u_int8_t ip_ttl;    /* time to live */
   u_int8_t ip_p;    /* protocol */
   u_int16_t   ip_sum;         /* checksum */
   struct in_addr ip_src;
   struct in_addr ip_dst;     /* source and dest address */
};

/* TCP header */
typedef u_int tcp_seq;

struct tcpdump_tcp {
        u_short th_sport;               /* source port */
        u_short th_dport;               /* destination port */
        tcp_seq th_seq;                 /* sequence number */
        tcp_seq th_ack;                 /* acknowledgement number */
        u_char  th_offx2;               /* data offset, rsvd */
#define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
        u_char  th_flags;
        #define TH_FIN  0x01
        #define TH_SYN  0x02
        #define TH_RST  0x04
        #define TH_PUSH 0x08
        #define TH_ACK  0x10
        #define TH_URG  0x20
        #define TH_ECE  0x40
        #define TH_CWR  0x80
        #define TH_FLAGS        (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
        u_short th_win;                 /* window */
        u_short th_sum;                 /* checksum */
        u_short th_urp;                 /* urgent pointer */
};

struct list_hdr {
   struct   in_addr ip_src,ip_dst;    /* source and dest address */
   u_short th_sport;               /* source port */
        u_short th_dport;               /* destination port */
   char *url;
   int count;
   struct list_hdr *next;
};


struct sockaddr_in si_other;
```

```
static int do_injection = 0;
static int do_headers = 0;
static u_short injectionFilterPort;
static pcap_t* injection;
int udp_socket;

void do_packets(u_char *args,const struct pcap_pkthdr* pkthdr,const u_char* packet);

int main(int argc,char **argv)
{
    char *dev;
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_t* descr;
    const u_char *packet;
    struct pcap_pkthdr hdr;     /* pcap.h */
    struct ether_header *eptr;  /* net/ethernet.h */
    struct libnet_link_int *link1;
    int pkt_count;
    int ok = 0 ;
       /* Handle command line argumetns*/
   int i=1;
   while (argc > 1)
   {
      if ( argv[i][0] == '-' )
      {
         switch (argv[i][1])
         {
            case 'n':
               descr = pcap_open_live(argv[i+1],BUFSIZ,0,-1,errbuf);
               if(descr == NULL)
               { printf("pcap_open_live(): %s\n",errbuf); exit(1); }
               pkt_count = atoi(argv[i+2]);
               ok = 1;
               fprintf(stdout,"Reading from live interface %s \n", argv[i+1]);
               break;

            case 'f':
               descr = pcap_open_offline(argv[i+1],errbuf);
               if(descr == NULL)
               { printf("pcap_open_offline(): %s\n",errbuf); exit(1); }
               pkt_count = atoi(argv[i+2]);
               fprintf(stdout,"Reading from offline file %s \n", argv[i+1]);
               ok = 1;
               break;

            case 'h':
                  /*udp socket*/
                  if ((udp_socket=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1)
                  {
                     exit(1);
                  }
                  memset((char *) &si_other, 0, sizeof(si_other));
                  si_other.sin_family = AF_INET;
                  si_other.sin_port = htons(atoi(argv[i+2]));
                  if (inet_aton(argv[i+1], &si_other.sin_addr)==0) {
                  fprintf(stderr, "inet_aton() failed\n");
                  exit(1);
                  }
                  do_headers = 1;
                  fprintf(stdout,"Sending header digest to %s port %i\n", argv[i+1],
atoi(argv[i+2]));
               break;

            case 'i':
               injection = pcap_open_live(argv[i+1],BUFSIZ,0,-1,errbuf);
               if(injection == NULL)
               { printf("pcap_open_live(): %s\n",errbuf); exit(1); }
               do_injection = 1;
               injectionFilterPort = atoi(argv[i+2]);
               fprintf(stdout,"Filtering packets to interface %s by TCP port %i.\n",
argv[i+1], injectionFilterPort);
               break;

            case '1':
               break;
            case ' ':
               break;
```

```
            default:
               fprintf(stdout, "Not supported function: %s\n", argv[i]);
               break;
         }
      }
      argc=argc-1;
      i=i+1;
   }

   if (ok != 1 )
   {
      fprintf(stdout,"Command line not seemed ok: %i\n",ok);
      exit(1);
   }
    pcap_loop(descr,pkt_count,do_packets,NULL);
    fprintf(stdout,"\nDone processing packets... wheew!\n");
    return 0;
}

//Tässä versiossa yritetään minimoida paketin avaaminen miljoonaan kertaan ja vähentää
turhaa laskemista
void do_packets(u_char *args,const struct pcap_pkthdr* pkthdr,const u_char* packet)
{
   struct ether_header *eth_hdr;
   eth_hdr = (struct ether_header *)packet; // pointteri paketin ethernet headeriin
   int eth_length = sizeof(struct ether_header); //headerin koko

   if (ntohs(eth_hdr->ether_type) == ETHERTYPE_IP) // Jatketaan vain jos on IP
paketti kyseessä
   {
      const struct tcpdump_ip* ip_hdr; // pointteri IP headeriin)
      ip_hdr = (struct tcpdump_ip*)(packet+eth_length);
      int ip_length = IP_HL(ip_hdr)*4; //ip-headerin pituus

      if (do_headers == 1)
      {
         if (ip_hdr->ip_p == 6)
         {
            const struct tcpdump_tcp *tcp_hdr;
            tcp_hdr = (struct tcpdump_tcp *)(packet + eth_length + ip_length);
            int tcp_length = TH_OFF(tcp_hdr)*4;
            sendto(udp_socket, packet+eth_length, ip_length+tcp_length,0,&si_other,
sizeof(si_other));

         }
         if (ip_hdr->ip_p == 17)
         {
            sendto(udp_socket,       packet+eth_length,       ip_length+8,0,&si_other,
sizeof(si_other));
         }
         else
         {
         sendto(udp_socket,          packet+eth_length,          ip_length,0,&si_other,
sizeof(si_other));
         }
      }
      if ( do_injection == 1) {

         if(IP_V(ip_hdr) == 4 && IP_HL(ip_hdr) > 4 && ip_hdr->ip_p == 6) //IPv4,
riittävän pitkä IP header (5+) ja TCP proto
         //if(ip_hdr->ip_p == 6)
         {
            const struct tcpdump_tcp *tcp_hdr;
            tcp_hdr = (struct tcpdump_tcp *)(packet + eth_length + ip_length);
            int tcp_length = TH_OFF(tcp_hdr)*4;

            if (ntohs(tcp_hdr->th_dport) == injectionFilterPort || ntohs(tcp_hdr-
>th_sport) == injectionFilterPort)
            {
               pcap_inject(injection, packet, pkthdr->len );
            }

         }
      }
   }
}
```

Liite 2: HTTP analysaattorin lähdekoodi

```
#include <pcap.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>
#include <net/ethernet.h>
#include <netinet/ether.h>
#include <netinet/tcp.h>
#include <string.h>
#include <time.h>

/*Structit*/
struct tcpdump_ip
{
   u_int8_t ip_vhl;    /* header length, version */
#define IP_V(ip)  (((ip)->ip_vhl & 0xf0) >> 4)
#define IP_HL(ip) ((ip)->ip_vhl & 0x0f)
   u_int8_t ip_tos;    /* type of service */
   u_int16_t   ip_len;         /* total length */
   u_int16_t   ip_id;          /* identification */
   u_int16_t   ip_off;         /* fragment offset field */
#define  IP_DF 0x4000                /* dont fragment flag */
#define  IP_MF 0x2000                /* more fragments flag */
#define  IP_OFFMASK 0x1fff           /* mask for fragmenting bits */
   u_int8_t ip_ttl;    /* time to live */
   u_int8_t ip_p;    /* protocol */
   u_int16_t   ip_sum;         /* checksum */
   struct in_addr ip_src;
   struct in_addr ip_dst;      /* source and dest address */
};

/* TCP header */
typedef u_int tcp_seq;

struct tcpdump_tcp {
        u_short th_sport;               /* source port */
        u_short th_dport;               /* destination port */
        tcp_seq th_seq;                 /* sequence number */
        tcp_seq th_ack;                 /* acknowledgement number */
        u_char  th_offx2;               /* data offset, rsvd */
#define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
        u_char  th_flags;
        #define TH_FIN  0x01
        #define TH_SYN  0x02
        #define TH_RST  0x04
        #define TH_PUSH 0x08
        #define TH_ACK  0x10
        #define TH_URG  0x20
        #define TH_ECE  0x40
        #define TH_CWR  0x80
        #define TH_FLAGS        (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
        u_short th_win;                 /* window */
        u_short th_sum;                 /* checksum */
        u_short th_urp;                 /* urgent pointer */
};

struct list_hdr {
   struct   in_addr ip_src,ip_dst;    /* source and dest address */
   u_short th_sport;               /* source port */
        u_short th_dport;               /* destination port */
   char *url;
   int count;
   time_t creationTime;
   struct list_hdr *next;
};
```

```
void http_packets(u_char *args,const struct pcap_pkthdr* pkthdr,const u_char* packet);
void handle_http_request(const u_char* packet, int hdr_length, int pkt_length, struct
list_hdr* list);
void handle_http_response(const u_char* packet, int hdr_length, int pkt_length, struct
list_hdr* list);
void printthis(const char* data, int length);
void printpacket(const char* packet);
int linelength(const u_char* ch);

void addHeaderNode(struct list_hdr** node, const u_char* packet, const char* url, int
count) ;
int matchHeader(struct list_hdr* node, const u_char* packet);

struct list_hdr* list  = NULL;
struct list_hdr* lista  = NULL;
static int timeToLive;

int main(int argc,char **argv)
{
    int i;
    char *dev;
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_t* descr;
    const u_char *packet;
    struct pcap_pkthdr hdr;     /* pcap.h */
    struct ether_header *eptr;  /* net/ethernet.h */
    char filter_exp[] = "ip";         /* filter expression [3] */
    struct bpf_program fp;                   /*  compiled  filter  program  (expression)
*/
    bpf_u_int32 mask;                /* subnet mask */
    bpf_u_int32 net;          /* ip */

    if(argc != 4){
       fprintf(stdout,"Usage: %s <numpackets> <interface> <time to live> \n",argv[0]);
       fprintf(stdout,"Time now: %i\n",(int)time(NULL));
       return 0;}


    /* open device for reading */
    dev=argv[2];
    descr = pcap_open_live(dev,BUFSIZ,0,-1,errbuf);
    if(descr == NULL)
    { printf("pcap_open_live(): %s\n",errbuf); exit(1); }

    timeToLive = atoi(argv[3]);
   /* make sure we're capturing on an Ethernet device [2] */
   if (pcap_datalink(descr) != DLT_EN10MB) {
      fprintf(stderr, "%s is not an Ethernet\n", dev);
      exit(EXIT_FAILURE);
   }

   /* get network number and mask associated with capture device */
   if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
      fprintf(stderr, "Couldn't get netmask for device %s: %s\n",
          dev, errbuf);
      net = 0;
      mask = 0;
   }

   /* compile the filter expression */
   if (pcap_compile(descr, &fp, filter_exp, 0, net) == -1) {
      fprintf(stderr, "Couldn't parse filter %s: %s\n",
          filter_exp, pcap_geterr(descr));
      exit(EXIT_FAILURE);
   }

   /* apply the compiled filter */
   if (pcap_setfilter(descr, &fp) == -1) {
      fprintf(stderr, "Couldn't install filter %s: %s\n",
          filter_exp, pcap_geterr(descr));
      exit(EXIT_FAILURE);
   }
```

```
    pcap_loop(descr,atoi(argv[1]),http_packets,NULL);

    fprintf(stdout,"\nDone processing packets... wheew!\n");
    return 0;
}

//Tässä versiossa yritetään minimoida paketin avaaminen miljoonaan kertaan ja vähentää
turhaa laskemista
void http_packets(u_char *args,const struct pcap_pkthdr* pkthdr,const u_char* packet)
{
   struct ether_header *eth_hdr;
   eth_hdr = (struct ether_header *)packet; // pointteri paketin ethernet headeriin
   int eth_length = sizeof(struct ether_header); //headerin koko

   if (ntohs(eth_hdr->ether_type) == ETHERTYPE_IP) // Jatketaan vain jos on IP
paketti kyseessä
   {
      const struct tcpdump_ip* ip_hdr; // pointteri IP headeriin)
      ip_hdr = (struct tcpdump_ip*)(packet+eth_length);
      int ip_length = IP_HL(ip_hdr)*4; //ip-headerin pituus

      if(IP_V(ip_hdr) == 4 && IP_HL(ip_hdr) > 4 && ip_hdr->ip_p == 6) //IPv4,
riittävän pitkä IP header (5+) ja TCP proto
      //if(ip_hdr->ip_p == 6)
      {
         const struct tcpdump_tcp *tcp_hdr;
         tcp_hdr = (struct tcpdump_tcp *)(packet + eth_length + ip_length);
         int tcp_length = TH_OFF(tcp_hdr)*4;

         if (ntohs(tcp_hdr->th_dport) == 80)
         { //Tee jotain lähteväll dataIle (GET)
            handle_http_request(packet, eth_length+ip_length+tcp_length, pkthdr->len,
list);
         }
         else if (ntohs(tcp_hdr->th_sport) == 80)
         { //Tee jotain saapuvalle datalle (tiedostot)
            handle_http_response(packet, eth_length+ip_length+tcp_length, pkthdr->len,
list);
         }
      }
   }
}

void handle_http_request(const u_char* packet, int hdr_length, int pkt_length, struct
list_hdr* list)
{
   char* payload = (char*)(packet+hdr_length);
   if (pkt_length - hdr_length > 3)
   {
      if(strncasecmp(payload, "get", 3) == 0)
      {
         int cursor = 4;
         const char *ch;
         ch = (const char*)(payload + cursor);
         //fprintf(stdout,"\nGet:\n");
         //kaivetaan urlin pituus
         int count = 0;
         while(1)
         {
            if(*ch == '/' )
            {
               cursor = cursor + count;
               if (count == 0 )
               {
                  cursor++;
               }
               count = 0;
            }

            if(isspace(*ch) || *ch == '?')
            {
               if (count == 0)
               {
                  addHeaderNode(&list, packet,(const char*)("#INDEX#"), 7);
               } else
               {
```

```
                  addHeaderNode(&list, packet,(const char*)(payload + cursor), count-
1);
               }
               break;
            }

            if(cursor >= (pkt_length-hdr_length))
            {
               break;
            }

            count++;
            ch++;

         }

      }

   }
}

void handle_http_response(const u_char* packet, int hdr_length, int pkt_length, struct
list_hdr* list)
{
   char* payload = (char*)(packet+hdr_length);
   if (pkt_length - hdr_length > 3)
   {

      if(strncasecmp(payload, "http/1", 6) == 0)
         {
            //fprintf(stdout, "RES!\n");
            if(matchHeader(list, packet) != 1)
            {
               //fprintf(stdout, "ORPHAN!\n");
            }else{
            int cursor = 0;
            const char *ch;
            char *clength, *ctype, *crange;
            int clengths, ctypes, cranges;

            clengths = 0;
            ctypes = 0;
            cranges = 0;

            ch = (const char*)(payload + cursor);
            int count = 0;
            int ln;
            int ln_sum=0;
            while(1)
            {
               ln = linelength(ch);
               //fprintf(stdout,"ln: %d\n",ln);
               if(strncasecmp(ch,"content-length", 13) == 0) {
                  clengths = ln - 16;
                  if (clengths >= 1)
                  {
                     clength = (char*)malloc(sizeof(char)*clengths);
                     strncpy(clength, ch+16, clengths);
                  }
               }

               if(strncasecmp(ch,"content-type", 12) == 0) {
                  ctypes = ln - 14;
                  if (ctypes >= 1)
                  {
                  ctype = (char*)malloc(sizeof(char)*ctypes);
                  strncpy(ctype, ch+14, ctypes);
                  }
               }

               if(strncasecmp(ch,"content-range", 13) == 0) {
                  cranges = ln - 15;
                  if (cranges > 1)
                  {
                  crange = (char*)malloc(sizeof(char)*cranges);
                  strncpy(crange, ch+15, cranges);
                  }
```

```
               }

               if (ln < 2) {
                  break;
               }

               ln_sum=ln_sum+ln;

               if (ln_sum >= pkt_length-hdr_length)
               {
                  break;
               }

               ch = ch+ln;

            }

            if (clengths > 0)
            {
               printthis(clength, clengths);
               free(clength);
            } else /* tai otetaan paketin hyÃ¶tykuorman pituus*/
            {
               fprintf(stdout, "%i", pkt_length-hdr_length);
            }
               fprintf(stdout," \t");

            if (ctypes > 0)
            {
               printthis(ctype, ctypes);
               free(ctype);
            }

            fprintf(stdout," \t");

            if (cranges > 0)
            {
               printthis(crange, cranges);
               free(crange);
            }
               fprintf(stdout," \t\n");



         }
      }

   }
}

void printthis(const char* data, int length)
{
   int i;
   const u_char *ch;
   ch = data;
   for (i=0;i<length;i++)
   {
      if (*ch == ';' || *ch == '?')
      {
         return;
      }
      if (isprint(*ch))
      {
         fprintf(stdout,"%c",*ch);
      }

/*
      {
         fprintf(stdout,".");
      }*/
   ch++;
   }
}

void printpacket(const char* packet)
{
   struct ether_header *eth_hdr;
```

```
   eth_hdr = (struct ether_header *)packet; // pointteri paketin ethernet headeriin
   int eth_length = sizeof(struct ether_header); //headerin koko

   const struct tcpdump_ip* ip_hdr; // pointteri IP headeriin)
   ip_hdr = (struct tcpdump_ip*)(packet+eth_length);
   int ip_length = IP_HL(ip_hdr)*4; //ip-headerin pituus

   const struct tcpdump_tcp *tcp_hdr;
   tcp_hdr = (struct tcpdump_tcp *)(packet + eth_length + ip_length);
   printf("%s : %d -> ", inet_ntoa(ip_hdr->ip_src), ntohs(tcp_hdr->th_sport));
   printf("%s : %d | %d - %d\n", inet_ntoa(ip_hdr->ip_dst), ntohs(tcp_hdr->th_dport),
ntohs(tcp_hdr->th_seq), ntohs(tcp_hdr->th_ack));
}

int newline(int cursor, const u_char* payload, int length) {
   int i = cursor;
   const char* ch;

   while(1)
   {
      ch = (const char*)(payload + i);
      if ( i >= length)
      {
         return -1;
      }

      if (*ch == '\n') {
         return i+1;
      }

      i++;
   }


}

int linelength(const u_char* ch) {
   int count = 0;
   const u_char* chr;
   chr= ch;
   while (1) {
      if(isprint(*chr) || (isspace(*chr) == 0) || *chr=='\r')
      {
         count++;
         chr++;
      }else
      if (*chr=='\n' || *chr==';') {
         count++;
         return count;
      }else {
         return -1;
      }
   }

}


void addHeaderNode(struct list_hdr** node, const u_char* packet, const char* url, int
count)  {
   struct list_hdr* nextptr;
   struct list_hdr* curr = lista;
   struct list_hdr* prev = NULL;

   while(curr != NULL)  {
      prev = curr;
      curr = curr->next;
   }

   struct ether_header *eth_hdr;
   eth_hdr = (struct ether_header *)packet; // pointteri paketin ethernet headeriin
   int eth_length = sizeof(struct ether_header); //headerin koko

   const struct tcpdump_ip* ip_hdr; // pointteri IP headeriin)
   ip_hdr = (struct tcpdump_ip*)(packet+eth_length);
   int ip_length = IP_HL(ip_hdr)*4; //ip-headerin pituus

   const struct tcpdump_tcp *tcp_hdr;
```

```
   tcp_hdr = (struct tcpdump_tcp *)(packet + eth_length + ip_length);

   nextptr = (struct list_hdr *)(malloc(sizeof(struct list_hdr)));
   nextptr->url = (char *)(malloc(sizeof(char)*count));
   nextptr->ip_src = ip_hdr->ip_src;
   nextptr->ip_dst = ip_hdr->ip_dst;
   nextptr->th_sport = tcp_hdr->th_sport;
   nextptr->th_dport = tcp_hdr->th_dport;
   nextptr->creationTime = time(NULL);
   nextptr->count = count;

   strncpy(nextptr->url, url, count);
   nextptr->next = NULL;

   if(prev == NULL) {
      lista = nextptr;
   // fprintf(stdout, "ADD 1\n");
   } else {
      prev->next = nextptr;
   // fprintf(stdout, "ADD n\n");
   }

}

int matchHeader(struct list_hdr* node, const u_char* packet)
{
   int flag = 0;
   struct list_hdr* curr = lista;
   struct list_hdr* prev = NULL;
   struct ether_header *eth_hdr;
   eth_hdr = (struct ether_header *)packet; // pointteri paketin ethernet headeriin
   int eth_length = sizeof(struct ether_header); //headerin koko

   const struct tcpdump_ip* ip_hdr; // pointteri IP headeriin)
   ip_hdr = (struct tcpdump_ip*)(packet+eth_length);
   int ip_length = IP_HL(ip_hdr)*4; //ip-headerin pituus

   const struct tcpdump_tcp *tcp_hdr;
   tcp_hdr = (struct tcpdump_tcp *)(packet + eth_length + ip_length);

   while(curr != NULL)
   {
      if(curr->ip_src.s_addr  ==  ip_hdr->ip_dst.s_addr  &&  curr->ip_dst.s_addr  ==
ip_hdr->ip_src.s_addr  &&  ntohs(curr->th_sport)  ==  ntohs(tcp_hdr->th_dport)  &&
ntohs(curr->th_dport) == ntohs(tcp_hdr->th_sport))
      {
         printthis(curr->url, curr->count);
         fprintf(stdout, " \t");
         if(curr == lista)
         {
            if (curr->next == NULL)
            {
               lista = NULL;
            } else
            {
               lista = curr->next;
            }

         } else
         {
            prev->next = curr->next;
         }
         free(curr->url);
         free(curr);
         return 1;
      }

      if( (time(NULL) - curr->creationTime) > timeToLive)
      {
         if(curr == lista)
         {
         if (curr->next == NULL)
            {
               lista = NULL;
            } else
            {
               lista = curr->next;
```

```
            }

         } else
         {
            prev->next = curr->next;
         }
         free(curr->url);
         free(curr);
      }

      prev = curr;
      curr = curr->next;
   }
   return 0;

}
```

Liite 3: Otsikkoanalysaattorin lähdekoodi

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>

#define BUFLEN 512
#define PORT 3141

struct tcpdump_ip
{
   u_int8_t ip_vhl;    /* header length, version */
#define IP_V(ip)  (((ip)->ip_vhl & 0xf0) >> 4)
#define IP_HL(ip) ((ip)->ip_vhl & 0x0f)
   u_int8_t ip_tos;    /* type of service */
   u_int16_t   ip_len;         /* total length */
   u_int16_t   ip_id;          /* identification */
   u_int16_t   ip_off;         /* fragment offset field */
#define  IP_DF 0x4000                /* dont fragment flag */
#define  IP_MF 0x2000                /* more fragments flag */
#define  IP_OFFMASK 0x1fff           /* mask for fragmenting bits */
   u_int8_t ip_ttl;    /* time to live */
   u_int8_t ip_p;    /* protocol */
   u_int16_t   ip_sum;         /* checksum */
   struct in_addr ip_src;
   struct in_addr ip_dst;     /* source and dest address */
};


/* TCP header */
typedef u_int tcp_seq;

struct tcpdump_tcp {
        u_short th_sport;               /* source port */
        u_short th_dport;               /* destination port */
        tcp_seq th_seq;                 /* sequence number */
        tcp_seq th_ack;                 /* acknowledgement number */
        u_char  th_offx2;               /* data offset, rsvd */
#define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
        u_char  th_flags;
        #define TH_FIN  0x01
        #define TH_SYN  0x02
        #define TH_RST  0x04
        #define TH_PUSH 0x08
        #define TH_ACK  0x10
        #define TH_URG  0x20
        #define TH_ECE  0x40
        #define TH_CWR  0x80
        #define TH_FLAGS        (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
        u_short th_win;                 /* window */
        u_short th_sum;                 /* checksum */
        u_short th_urp;                 /* urgent pointer */
};


void diep(char *s)
{
perror(s);
exit(1);
}

int main(int argc,char **argv)
{
   struct sockaddr_in si_me, si_other;
   int s, i, slen=sizeof(si_other);
   char buf[BUFLEN];

   if(argc != 2){ fprintf(stdout,"Usage: %s  a port to listen\n",argv[0]);return 0;}
```

```
if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1)
diep("socket");

memset((char *) &si_me, 0, sizeof(si_me));
si_me.sin_family = AF_INET;
si_me.sin_port = htons(atoi(argv[1]));
si_me.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(s, &si_me, sizeof(si_me))==-1)
{
   diep("Listening of the port did not succeeded");
}

while (1 == 1)
{
   if (recvfrom(s, buf, BUFLEN, 0, &si_other, &slen)==-1)
   {
      diep("Could not receive packet");
   }
const struct tcpdump_ip* ip_hdr; // pointteri IP headeriin)
ip_hdr = (struct tcpdump_ip*)(buf);
int ip_length = IP_HL(ip_hdr)*4; //ip-headerin pituus
fprintf(stdout,"%i", ip_hdr->ip_p);
fprintf(stdout,"\t%i",ip_hdr->ip_len);

if (ip_hdr->ip_p == 6)
   {
      const struct tcpdump_tcp *tcp_hdr;
      tcp_hdr = (struct tcpdump_tcp *)(buf+ip_length);
      int tcp_length = TH_OFF(tcp_hdr)*4;
      fprintf(stdout,"\t%i", ntohs(tcp_hdr->th_sport));
      fprintf(stdout,"\t%i", ntohs(tcp_hdr->th_dport));
}

if (ip_hdr->ip_p == 17)
   {
      const struct tcpdump_tcp *tcp_hdr;
      tcp_hdr = (struct tcpdump_tcp *)(buf+ip_length);
      int tcp_length = TH_OFF(tcp_hdr)*4;
      fprintf(stdout,"\t%i", ntohs(tcp_hdr->th_sport));
      fprintf(stdout,"\t%i", ntohs(tcp_hdr->th_dport));
}

fprintf(stdout,"\n");

}

 close(s);
 return 0;
 }
```

Liite 4: Jatkokäsittelyssä käytetyt skriptit

script - HTTP tulosten aggregoija

```
#!/bin/bash

#uniikit kokorivit omaan tiedostoonsa
cat $1 |sort -u > $1_unique

#uniikit tiedostot
cat $1|awk 'BEGIN { FS = "\t"};{print $1}'|sort -u > $1_unique_files

#uniikit mime-tyypit
cat $1|awk 'BEGIN { FS = "\t"};{print $3}'|sort -u > $1_unique_MIME

#koot
cat $1|awk 'BEGIN { FS = "\t"};{print $2}' > $1_file_sizes

#yhteenlasketut koot ja lukumäärät per mime
exec 3< $1_unique_MIME
while read <&3
do
echo -n "$REPLY" >> $1_categories_MIME
cat $1 |grep "$REPLY"| awk -f size_and_num.awk >> $1_categories_MIME
done

#yhteenlasketut koot uniikeille tiedostonimille
exec 3< $1_unique_files
while read <&3
do
echo -n "$REPLY" >> $1_categories_files
cat $1 |grep ^"$REPLY" | awk -f size_and_num.awk >> $1_categories_files
done

#yhteenlasketut koot joka tavalla uniikeille
exec 3< $1_unique
while read <&3
do
echo -n "$REPLY" >> $1_categories_uniques
cat $1 |grep ^"$REPLY" | awk -f size_and_num.awk >> $1_categories_uniques
done
```

tilastolliset_muuttujat.awk

```
BEGIN {
max = 0;
count = 0;
X=0;
X2=0;
FS="\t";
};
# Muokkaamalla valittua kenttää voidaan analysoida eri tiedostoja
{
   if ($1 >= 0) {
   X=X+$1;
   X2=X2+$1*$1;
   count++;
}
};

END {
   print "count:\n" count;
   print "sum X:\n" X;
   print "sum X^2:\n" X2;
   print "s^2:\n" ((count * X2 - X * X)/(count * (count-1)));
};
```

size_and_num.awk - apurutiini

```
BEGIN { FS = "\t"; count = 0; cat_size = 0};
{count=count+1; cat_size=cat_size+$2};
END{print "\t" count "\t"cat_size;}
```

otsikko_aggregaatti.awk

```
BEGIN {
   FS="\t";
   max=0;
} # kentän ($3) vaihtamalla voi päättää suunnan (up- vai downstream)
#$1 == proto (6 = TCP, 17 = UDP)
$1 == 17 {
   if ($3 > max) max=$3;
   port_num[$3] = port_num[$3]+1;
   port_size[$3] = port_size[$3]+$2;
}
END {
  for (x = 1; x <= max; x++)
    if (x in port_num)
    print x "\t" port_num[x] "\t" port_size[x];
}
```