OULUN SEUDUN
AMMATTIKORKEAKOULU

Esa Pesonen

IP-PAKETTIEN SUODATUS MONIYDINVERKKOPROSESSORILLA

IP-PAKETTIEN SUODATUS MONIYDINVERKKOPROSESSORILLA

Esa Pesonen
Opinnäytetyö
9.5.2011
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

OULUN SEUDUN AMMATTIKORKEAKOULU TIIVISTELMÄ

Koulutusohjelma	Opinnäytetyö	Sivuja + Liitteitä
Tietotekniikan koulutusohjelma	Opinnäytetyö	25 + 0
Suuntautumisvaihtoehto	Aika	
Ohjelmistojen kehitys	9.5.2011	
Työn tilaaja	Työn tekijä	
Rugged Tooling Oy	Esa Pesonen	

Työn nimi

IP-pakettien suodatus moniydinverkkoprosessorilla

Avainsanat

Protokollat, tietoliikenne, libpcap, pakettisuodatus, sulautetut järjestelmät

Opinnäytetyön tavoitteena oli kehittää Cavium Octeon CN5020 -verkkoprosessorille paketinkaappausohjelmisto C-kielellä. Verkkolaite asennetaan Ethernet-verkkoon, missä se tutkii verkon läpi menevät paketit. Kun paketti saapuu laitteelle, sitä verrataan olemassa oleviin kaappaussääntöihin ja säännön perusteella paketti joko monistetaan tai lähetetään normaalisti eteenpäin.

Ohjelmistokoodit ovat salaisia, joten ohjelmointiteknisiä asioita ei käsitellä tässä dokumentissa. Yleiskuva laitteen toiminnasta ja sen sisältämistä ominaisuuksista käydään läpi. Ohjelmistoa varten kehitettiin oma kontrolliviestiprotokolla, jonka avulla sääntöjä voidaan syöttää laitteen ollessa käynnissä.

Ohjelma tuntee yleisimmät tietoliikenneverkoissa käytetyt protokollat, kuten Ethernet, IPv4, IPv6, TCP, ja UDP. Pakettien kaappaus tapahtuu näiden protokollien tunnistetietojen perusteella, jolloin esimerkiksi lähdeosoitetta voidaan käyttää suodattamiseen.

Opinnäytetyön tuloksena syntyi ohjelmisto, johon voi syöttää ja josta voi poistaa kaappaussääntöjä. Kaapatut paketit monistetaan ja välitetään eteenpäin.

SISÄLTÖ

SANASTO

BPF	Berkley Packet Filter, rajapinta siirtokerrokseen
CRC	Cyclic Redundancy Check, tiivistealgoritmi virheenkorjaukseen
GPRS	General Packet Radio Service, mobiiliverkoissa käytetty siirtokerroksen protokolla
HTTP	Hypertext Transfer Protocol, sovelluskerroksen protokolla www-sivujen tiedonsiirtoon
ICMP	Internet Control Message Protocol, kontrolliprotokolla IP-verkkoihin
IP	Internet Protocol, verkkokerroksen protokolla
LAN	Local Area Network, fyysisesti rajattu lähiverkko
MAC	Media Access Control, Ethernet-kehyksessä käytetty osoite
MPLS	Multiprotocol Label Switching, siirto- ja verkkokerroksen väliin tuleva protokolla
PPP	Point-to-Point Protocol, verkkolaitteiden välillä käytetty protokolla
TCP	Transmission Control Protocol, kuljetuskerroksen protokolla
UDP	User Datagram Protocol, kuljetuskerroksen protokolla
VLAN	Virtual LAN, siirto- ja verkkokerroksen väliin tuleva protokolla, myös virtuaalilähiverkko

1 JOHDANTO

Opinnäytetyön tavoitteena oli kehittää Cavium Octeon CN5020 -verkkoprosessorille paketinkaappausohjelmisto C-kielellä. Laite kytketään verkossa reitittimen monitorointiporttiin tai verkon osaan niin, että se pääsee käsiksi kaikkeen tietoliikenteeseen, mikä verkon sisällä kulkee. Ohjelmaan täytyi toteuttaa ominaisuus, jolla paketteja voidaan kaapata tiettyjen sääntöjen mukaan. Kaappaussääntöjen syöttäminen ja poistaminen tulee onnistua samaan aikaan, kun tietoliikenne menee laitteen läpi. Luvussa 4 (kuva 10) on kuvattu laitteen paikka tietoliikenneverkossa.

Tämän ohjelmiston tilasi Rugged Tooling Oy. Yritys tuottaa tietoliikenneverkkoihin soveltuvia ohjelmistoja. Tilaajan kanssa ohjelmakoodit sovittiin salaisiksi, joten tässä dokumentissa ei käsitellä ohjelmointiteknisiä asioita. Tästä johtuen dokumenttiin ei sisällytetä lähdekoodia.

Koodin tuottamisen lisäksi suoritin testausta ja pyrin vertaamaan tätä ohjelmaa ennestään tunnettuihin sovelluksiin kuten Tcpdump. Vertailu tällaisessa tapauksessa ei ole ihan selvää, koska ohjelmien käyttötarkoitukset ovat erilaisia.

2 PERUSASIOITA PROTOKOLLISTA

Tässä luvussa kuvataan lyhyesti tämän työn kannalta kiinnostavat ja oleelliset protokollat sekä arkkitehtuuria, jonka pohjalla ne toimivat. Protokollista käydään vain ne kentät lävitse, jotka ovat työn kannalta oleellisia.

2.1 ISO/OSI-malli

OSI-malli on 7-kerroksinen arkkitehtuuri, joka on kehitetty 1970-luvun lopussa. Mallilla voidaan kuvata esimerkiksi TCP/IP-pino, joka on tämän työn keskeinen osa. TCP/IP-pinossa on vain 5 kerrosta, joten esitystapa- ja istuntokerros on hoidettu pelkällä sovelluskerroksella. Kuvassa 1 on kuvattu molemmat mallit. (Kurose – Ross 2008, 74.)

<table>
<tr><td>7. Sovelluskerros</td><td rowspan="3">Sovelluskerros (HTTP)</td></tr>
<tr><td>6. Esitystapakerros</td></tr>
<tr><td>5. Istuntokerros</td></tr>
<tr><td>4. Kuljetuskerros</td><td>Kuljetuskerros (TCP)</td></tr>
<tr><td>3. Verkkokerros</td><td>Verkkokerros (IP)</td></tr>
<tr><td>2. Siirtokerros</td><td>Siirtokerros (Ethernet)</td></tr>
<tr><td>1. Fyysinenkerros</td><td>Fyysinenkerros</td></tr>
</table>

KUVA 1. Vasemmalla OSI-malli ja oikealla TCP/IP-pino (Kurose ym. 2008, 74.)

Arkkitehtuurissa jokainen kerros hoitaa omaa tehtäväänsä. Kerrosten avulla voidaan helposti tutkia verkon tiettyä osaa, vaikka kyseessä olisi iso ja monimutkainen systeemi. Kerroksen toteutus voidaan muuttaa, jos sen tarjoamat rajapinnat pysyvät samana eli yksi kerros ei vaikuta suoraan muihin kerroksiin. (Kurose ym. 2008, 74.)

Malli ei kuitenkaan kuvaa nykyaikaista Internet-verkkoa täydellisesti, koska mukaan on tullut 2.5-kerroksen protokollia kuten VLAN ja MPLS. Esimerkiksi

näiden avulla voidaan hoitaa reititystä, vaikka se ei ole tämän kerroksen tehtävä. (Kurose ym. 2008, 76.)

Siirto-, verkko- ja kuljetuskerrokset ovat työn kannalta kiinnostavia, koska näissä protokollissa on kentät, joiden avulla pakettien yksilöinti voidaan tehdä. Kuljetuskerroksessa on kaksi eri protokollaa, joita ohjelma tukee: TCP ja UDP.

2.2 Ethernet

IEEE hallinnoi MAC-osoitteita. Osoitteessa on sellainen rakenne, että 24 ensimmäistä bittiä ovat valmistajakohtaisia ja 24 viimeistä bittiä laitekohtaisia. MAC-osoitteiden yleinen esitystapa on ff-ff-ff-ff-ff-ff. Osoitteet on suunniteltu olemaan pysyviä, joten osoite on sama, vaikka laite viedään eri paikkaan, toisin kuin IP-osoitteet. Verkossa ei pitäisi olla kahta laitetta, joilla on sama MAC-osoite. Kuvassa 2 on kuvattu Ethernet-kehyksen kentät. (Kurose ym. 2008, 497–499.)

Preamble	Vastaanottajan osoite	Lähettäjän osoite	Tyyppi	Data	CRC

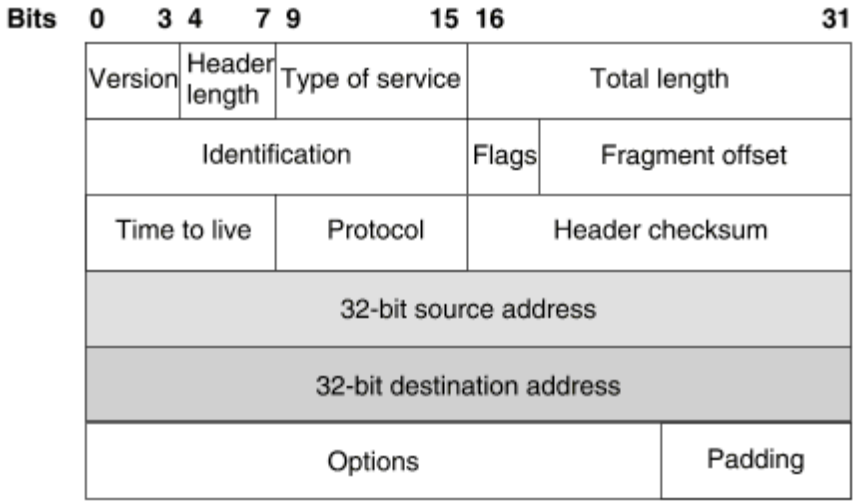
KUVA 2. Ethernet-kehyksen tietokentät (Kurose ym. 2008, 497.)

- Preamble (8 oktettia) herättää vastaanottajan ja synkronisoi kellot lähettäjän kanssa. Ominaisuus on tärkeä, koska lähetys voi tapahtua eri nopeuksilla, 10 Mbit/s, 100 Mbit/s jne.
- Vastaanottajan osoite (6 oktettia), jos Ethernet-kehyksessä on sama vastaanottajan osoite kuin kohde verkkolaitteessa, välitetään kehys verkkokerrokselle. Muuten kehys tiputetaan pois.
- Lähettäjän osoite (6 oktettia), verkkolaite laittaa tähän kenttään oman osoitteensa.
- Tyyppi (2 oktettia), kenttä, jossa ilmoitetaan, minkä verkkokerroksen protokollan tulee käsitellä data, esimerkiksi IP, IPv6, ARP jne.

- Data, (46–1500 oktettia), seuraavalla kerrokselle tarkoitettu data.
- CRC (4 oktettia), vastaanottaja voi tarkistaa, onko lähetyksessä tapahtunut virhe johtuen esimerkiksi huonosta kaapeloinnista.

2.3 IPv4

IPv4-protokolla on tällä hetkellä Internetissä käytetty verkkokerroksen protokolla. Suunnitteilla on siirtyminen IPv6-protokollaan, koska osoiteavaruus on nykyisellään liian pieni. IPv4-osoite on 4 oktettia pitkä, joten se mahdollistaa 2^{32} erilaista osoitetta eli 4 294 967 296 osoitetta. Yleinen esitystapa IPv4-osoitteille on 192.168.1.1. Osa osoitteista on kuitenkin varattu, kuten yleislähetys 255.255.255.255. Yleislähetystä käytetään, kun halutaan lähettää viesti kaikille samaan verkkoon kuuluville laitteille. Yhden verkon sisällä IP-osoitteiden on oltava yksilöllisiä, mutta eri aliverkoissa voi olla samoja osoitteita. Kuvassa 3 on kuvattu IPv4-kehyksen kentät. (Kurose ym. 2008 , 363–367.)



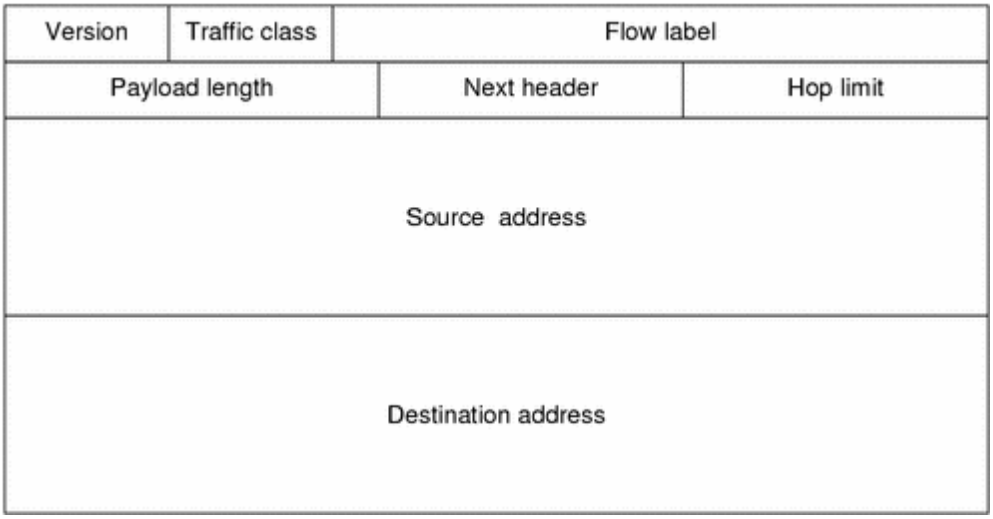
KUVA 3. IPv4-kehyksen kentät (Kurose ym. 2008 , 363.)

- Header length (2 oktettia), ilmoittaa IP-kehyksen pituuden. Kentän arvo on tyypillisesti 20, jos lisäoptioita ei ole asetettu. Tämä kenttä täytyy ottaa huomioon, jotta tiedetään, mistä kohdasta seuraavan kerroksen protokolla alkaa.

- Protocol (1 oktetti), seuraavan tason protokolla, TCP, UDP, ICMP jne.
- Source address (4 oktettia), lähettävän laitteen osoite.
- Destination address (4 oktettia), kohdelaitteen osoite.
- Data, kuljetuskerrokselle välitettävä data.

2.4 IPv6

IPv6-osoitteet ovat 128 bittiä eli 16 oktettia pitkiä. Osoitteen pituus mahdollistaa 2^{128} erilaista osoitetta. Osoiteavaruus on huomattavasti isompi kuin IPv4-protokollalla. Osoitteiden yleinen esitystapa on 2009:0ff8:0000:0000:0000:0000:1430:67fb, joka voidaan myös kirjoittaa ilman 0-kenttiä 2009:0ff8::1430:67fb. Otsikko on aina samanmittainen eli 40 oktettia pitkä. Tämän avulla tiedetään suoraan, mistä seuraavan kerroksen protokolla alkaa. Kuvassa 4 on kuvattu IPv6-kehyksen kentät. (Kurose ym. 2008, 386–388.)

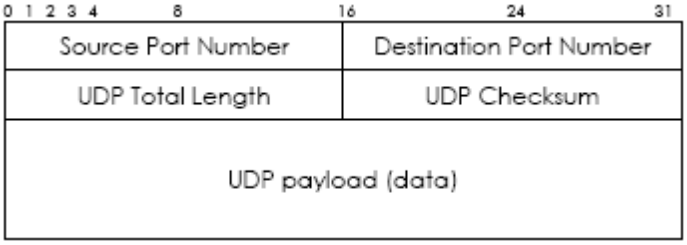


KUVA 4. IPv6-kehyksen kentät (Kurose ym. 2008, 386.)

- Next header (1 oktetti), seuraavan kerroksen protokolla.
- Source address (16 oktettia), lähettävän laitteen osoite.
- Destination address (16 oktettia), kohdelaitteen osoite.

2.5 UDP

UDP on kevyt yhteydetön protokolla ja otsikko pituudeltaan vain 8 oktettia. Otsikon jälkeen tulee data, jonka kuljetetaan sovelluskerrokselle. Protokollassa ei ole uudelleenlähetysmekanismia kuten TCP:ssä on. Jos pakettien perillemeno halutaan varmistaa, se täytyy tehdä sovellustasolla. Kuvassa 5 on kuvattu UDP-protokollan kentät. (Kurose ym. 2008, 238.)



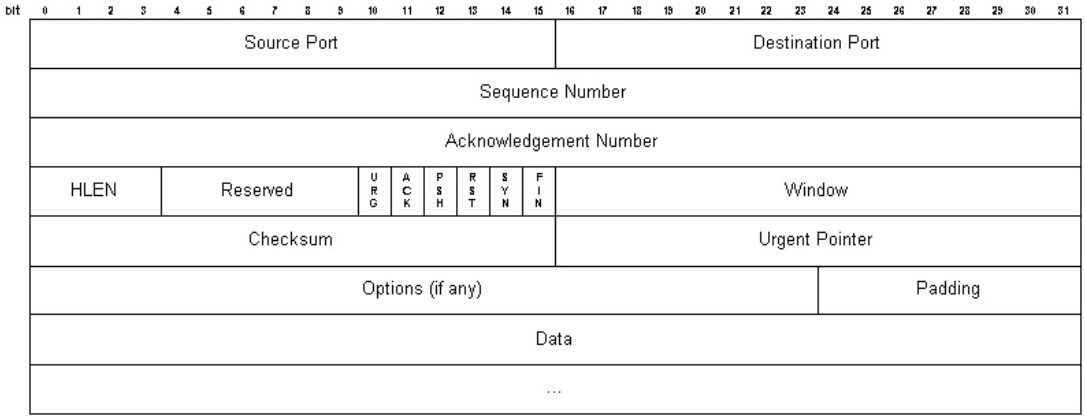
KUVA 5. UDP-protokollan otsikko (Kurose ym. 2008, 238.)

- Source Port Number (2 oktettia), lähettävä sovellus asettaa oman porttinumeron tähän.
- Destination Port Number (2 oktettia), kohde sovelluksen porttinumero.
- UDP payload (data), sovelluskerrokselle välitettävä data.

2.6 TCP

TCP-otsikon pituus on yhteensä 20 oktettia. Kuten UDP-protokollassa, porttinumerot ovat samassa kohtaa otsikkoa ja ovat myös 2 oktettia pitkiä. Toisin kuin UDP, TCP on yhteydellinen protokolla. Yhteys on vain lähteen ja kohteen tiedossa; reitittimet eivät tiedä, onko yhteyksiä päällä. (Kurose ym. 2008, 270.)

Kuten kuvasta 6 voi huomata, TCP sisältää huomattavasti enemmän tietoa kuin UDP. Otsikon sisältämiä kenttiä käytetään esimerkiksi varmistamaan, että lähetetyt paketit menevät perille. Kuten UDP:ssä, vain porttinumerot ovat kiinnostavia kenttiä tämän työn kannalta. (Kurose ym. 2008, 270.)

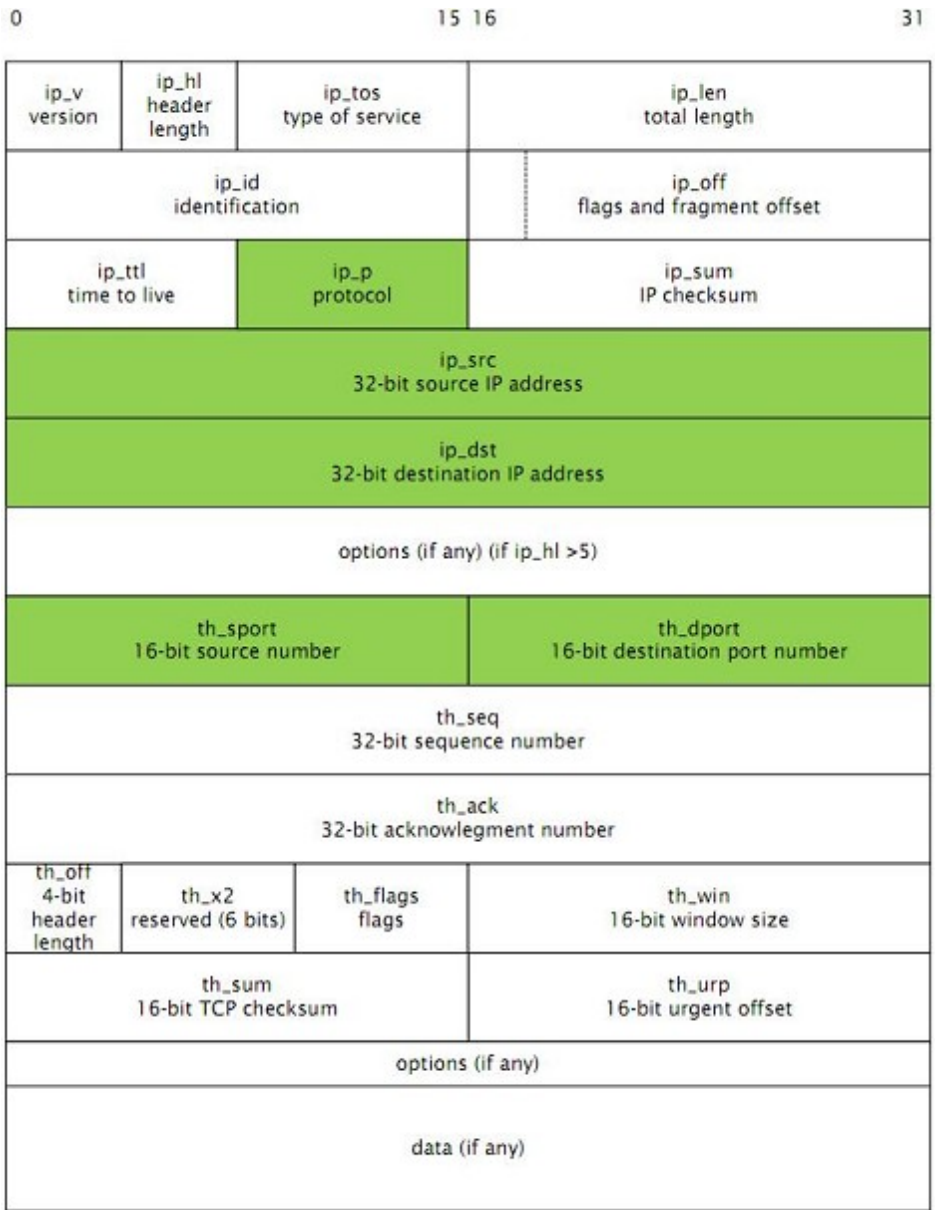


KUVA 6. TCP-protokolla otsikko (Kurose ym. 2008, 270.)

- Source Port (2 oktettia), lähettävä sovellus asettaa oman porttinsa tähän.
- Destination Port (2 oktettia), kohdesovelluksen porttinumero.
- Data, sovelluskerrokselle välitettävä data.

2.7 Five tuple

Five tuple on yleisesti käytetty käsite verkko-ohjelmoinnissa. Sen avulla voidaan yksilöidä kahden laiteen välinen yhteys sovellukseen asti. Five tuple tarvitsee IPv4- tai IPv6-protokollan verkkokerrokselta ja TCP- tai UDP-protokollan kuljetuskerrokselta. Se koostuu viidestä kentästä, IP-lähde- ja kohdeosoitteesta, protokollasta ja lähde- ja kohdeportista. Näiden kenttien sisältö on IPv4-paketissa yhteensä 13 oktettia ja IPv6-paketissa 37 oktettia. Five tuple ei ota kantaa siirto- tai sovelluskerroksen protokolliin, joten tämä tapa tutkia verkkoliikennettä toimii muissakin kuin Ethernet-verkoissa, esimerkiksi GPRS, PPP jne. Kuvassa 7 on väritettynä five tuple -kentät TCP/IP-pinosta. (Conte – Navarro – W. Hwu – Valero – Ungerer 2005, 256.)

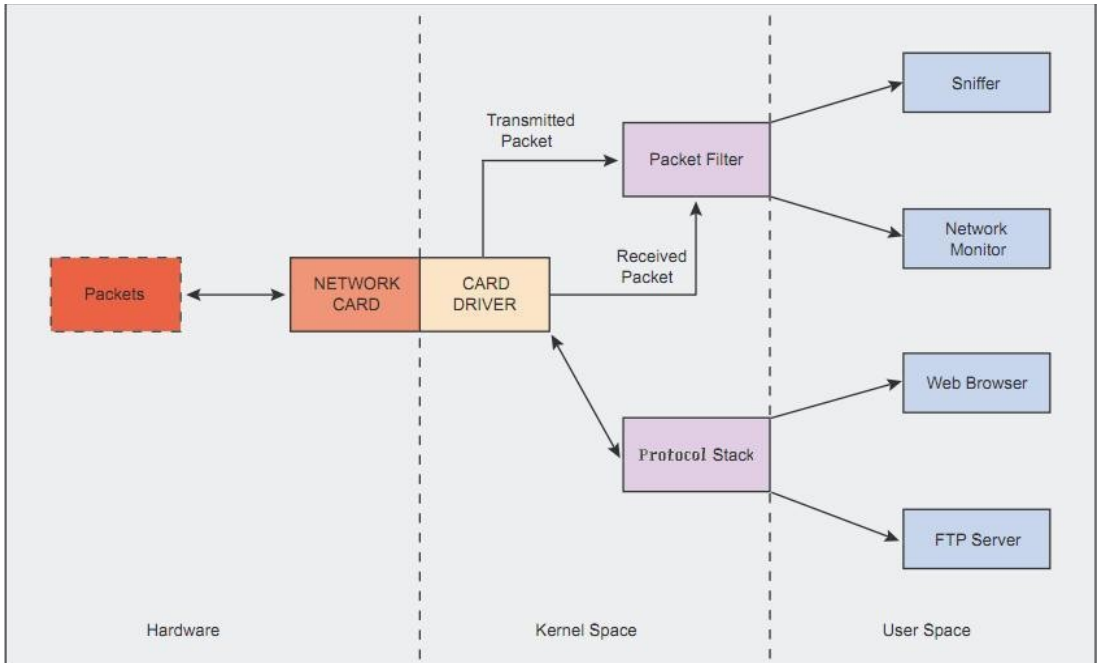


KUVA 7. IPv4 + TCP-paketti, five tuple -kentät väritettynä (Curtis 2010, 2-19.)

Verkkoprosessori, johon tämä sovellus ohjelmoitiin, osaa laskea tälle five tuplelle 16-bittisen CRC:n. Tätä ominaisuutta ei käytetty hyväksi tässä opinnäytetyössä, mutta jatkokehitysversioon se otettiin mukaan. (Curtis 2010, 2-18.)

3 LIBPCAP-PAKETINKAAPPAUSKIRJASTO

Libpcap on avoimen lähdekoodin paketinkaappauskirjasto. Tätä kirjastoa käytetään esimerkiksi sovelluksissa Tcpdump ja Wireshark. Libpcap on kehitetty Kalifornian yliopistossa vuonna 1994 alustariippumattomaksi rajapinnaksi pakettien kaappaukseen. Kirjasto toimii Unixin kaltaisissa käyttöjärjestelmissä kuten Linux, Solaris, BSD jne. Myös Windows-käyttöjärjestelmälle on olemassa vastaava kirjasto nimeltään Winpcap. Kuvassa 8 on kuvattu Libpcap-ohjelma eri rajapinnoista. (Garcia 2008, 39.)

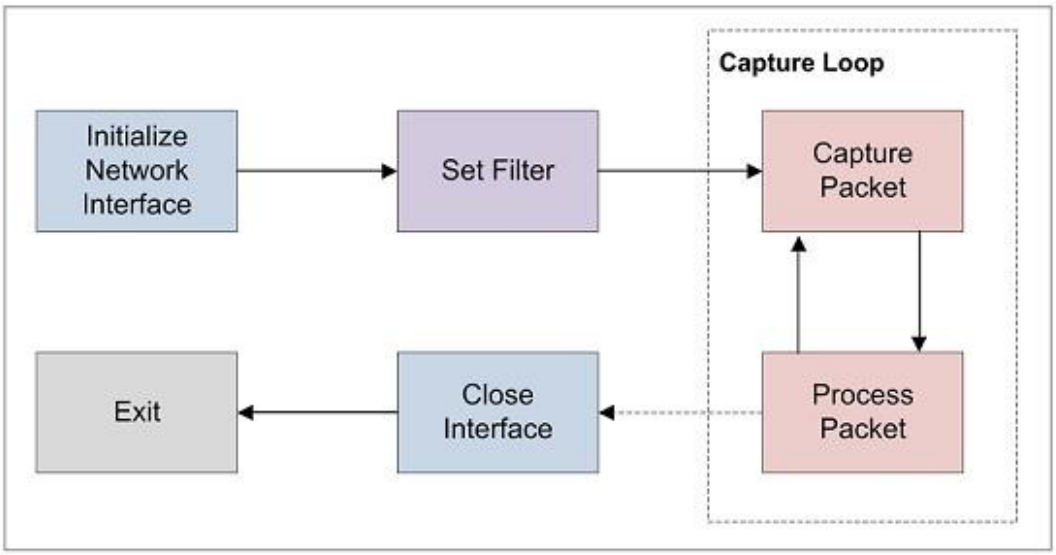


KUVA 8. Paketin kaappaus eri rajapinnoissa (Garcia 2008, 39.)

Yleensä verkkokortti tarkistaa Ethernet-kehyksestä kohdeosoitteen ja sen, onko se osoitettu tälle laitteelle. Verkkoliikenteen analysoinnissa ja statistiikan keräyksessä halutaan päästä käsiksi kaikkiin paketteihin, jotka verkkokortti näkee. Paketin kaappauksessa verkkokortti lähettää paketista kopion käyttöjärjestelmän ytimessä olevalle paketin kaapparille. (Garcia 2008, 39–40.)

Kun libpcap-kirjastoa käytetään, on ohjelmassa yleensä kuvassa 9 näkyvät kohdat:

- Initialize Network Interface, alustaa verkon rajapinnan
- Set Filter, asetetaan sääntö, minkä mukaan paketit kaapataan
- Capture Packet, kaapataan paketti ja nähdään sen sisältämä data
- Process Packet, tutkitaan paketin sisältöä
- Close Interface, suljetaan rajapinta.



KUVA 9. Tyypillisen paketin kaappausohjelman runko (Garcia 2008, 42.)

Ohjelmat, jotka käyttävät Libpcap-kirjastoa, kärsivät samoista ongelmista, jotka on ratkaistu tässä opinnäytetyössä. Libpcap vaatii, että kaappaussäännöt syötetään, ennen kuin pakettien tutkiminen aloitetaan, ja sääntöjä ei voi muuttaa ohjelman ollessa ajossa. Toinen ongelma on, että jos sääntöjä halutaan syöttää useita erilaisia, täytyy kaappausprosesseja käynnistää useita. (Garcia 2008, 42.)

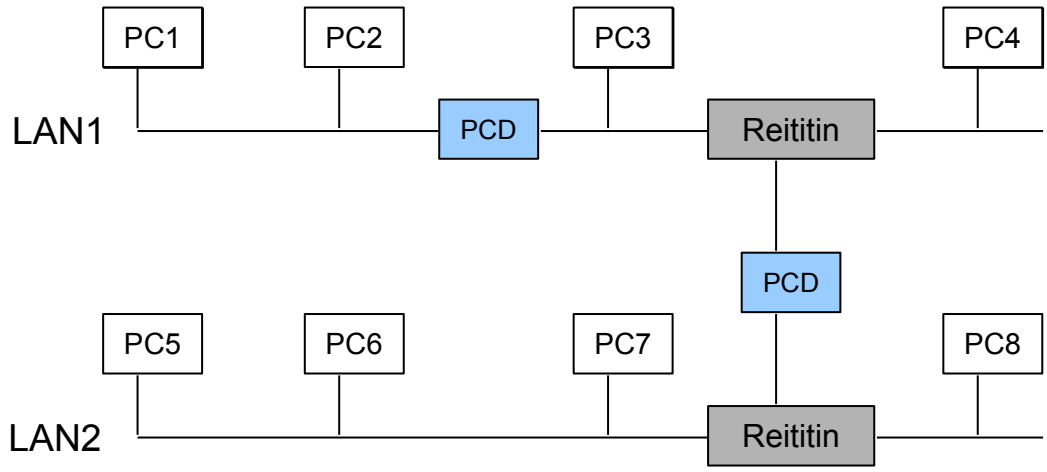
Libpcap-kirjastoa käyttävät ohjelmat pystyvät käsittelemään monipuolisempia kaappaussääntöjä kuin opinnäytetyössä kehittämäni ohjelma. Luvussa 5 tutkitaan tarkemmin molempien sääntöjen syöttöä. (Garcia 2008, 43.)

4 LAITETASON KUVAUS

Ohjelmisto kehitettiin Cavium Octeon CN5020 -mallin kaksiytimiselle verkkoprosessorille. Caviumin tarjoaman ohjelmistokehitysalustan avulla kehitetyt ohjelmat ovat yhteensopivia muiden Octeon-verkkoprosessoreiden kanssa. Octeon-prosessoreissa on 64-bittinen arkkitehtuuri ja tässä kyseisessä prosessorissa on 500 MHz:n kellotaajuus. Alustassa on kolme 1 Gbit/s nopeuksista Ethernet-porttia. Näistä kaksi on varattu läpimenevän liikenteen siltaukseen ja kolmas on monitorointia ja kontrolliviestejä varten. (Cam-0100. 2011.)

Ohjelmat voidaan ajaa itsenäisesti tai käyttöjärjestelmää käyttäen. On myös mahdollista, että yksi ydin ajaa käyttöjärjestelmää ja toinen itsenäisesti eri ohjelmaa. (Octeon cn52xx. 2011.)

Laitetta voi käyttää esimerkiksi kuvan 10 tapauksissa analysoimaan verkkoliikennettä. Ohjelmisto näkee kaikki Ethernet-paketit, jotka liikkuvat saman lähiverkon sisällä, joten syöttämällä five tuplen mukainen sääntö voidaan analysoida liikennettä esimerkiksi PC1:n ja PC3:n väliltä.



KUVA 10. Laitteen asennuskohdat tietoliikenneverkossa

- PCD, paketinkaappausohjelmistolla varustettu laite.

- PC 1–8, tietokoneita, jotka on kytketty Ethernet-verkkoon.
- LAN 1–2, eri lähiverkkoja, joihin tietokoneet on kytketty.
- Reititin, ohjaa verkkoliikennettä eri verkkojen välillä.

Ohjelmistossa on myös tuki VLAN-protokollalle, joten laitteistoa voidaan käyttää myös reitittimien välillä. VLAN-protokollalla voidaan määrittää virtuaalisia lähiverkkoja. Esimerkiksi jos PC2:n ja PC5:n välille olisi virtuaalilähiverkko määritetty, tietokoneet näkisivät toisensa samassa lähiverkossa, vaikka fyysinen verkko onkin eri. VLAN-protokollassa on tunnistuskenttä, joka on sama kaikille samaan virtuaalilähiverkkoon määritellyille laitteille. Ohjelmistoon pystyy määrittämään säännön, joka tunnistaa myös nämä VLAN-tunnisteet. Vaikka opinnäytetyön aiheena oli suunnitella kaappausrunko five tuple -tunnistukselle, suunnittelussa otettiin huomioon myös laajennusmahdollisuudet, kuten VLAN-protokolla. (Jaakkohuhta 2005, 157–160.)

Mielenkiintoisen tästä laitteistosta tekee ominaisuus, että verkko ei näe laitetta ollenkaan. Kun paketit tulevat laitteen ensimmäiseen porttiin, ne välitetään juuri samanlaisena eteenpäin, mutta kiinnostavat paketit nostetaan talteen.

5 SÄÄNTÖJEN SYÖTTÄMINEN

Kaappaussääntöjen syöttäminen Libpcap-kirjastoa käyttäviin ohjelmiin on yleensä erilaista kuin opinnäytetyön aiheena olevaan ohjelmaan. Koska opinnäytetyössä haluttiin dynaaminen sääntöjen syöttö, verkkoprosessorilla ei haluta käsitellä tekstimuotoisia sääntöjä, koska niiden käsittely voi olla liian hidasta, jos verkkokuormaa on paljon.

5.1 Tcpdump-sääntöjen syöttö

Tcpdump-ohjelmaan syötetään säännöt käynnistyksen yhteydessä BPF-tyylisenä merkkijonona. Libpcap mahdollistaa todella monipuolisten sääntöjen syöttämisen. Paketteja voidaan kaapata tutkimalla otsikkojen yksittäisiä kenttiä. Tämä ei ole mahdollista tekemässäni sovelluksessa.

Tässä on esimerkkejä BPF-tyylisistä säännöistä, joilla Tcpdump voidaan käynnistää, ja selitykset, minkälaisia paketteja niillä kaapataan:

- tcpdump src host 192.168.1.1 and tcp port 80

  Kaappaa paketit, joiden lähdeosoite on 192.168.1.1 ja tcp protokollassa lähde- tai kohdeportti 80.
- tcpdump host 10.2.1.1 and icmp

  Kaappaa paketit, joiden lähde- tai kohdeosoite on 10.2.1.1 ja IP-protokollan jälkeen ICMP-protokolla.
- tcpdump icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply

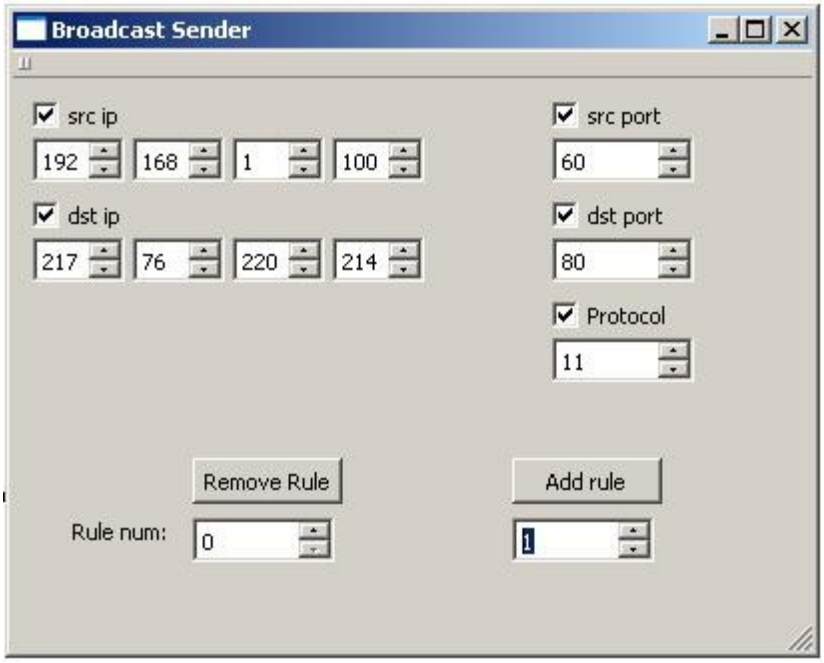
  Kaappaa kaikki ICMP-paketit, jotka eivät ole kaiutus- tai vastausviestejä, eli ping-paketteja ei kaapata.

Luettelon viimeisessä säännössä tutkitaan pakettia tarkemmin kuin opinnäytetyön sovelluksella onnistuu. Kaksi ensimmäistä sääntöä ovat sen tyyppisiä, joita tällä sovelluksella on tarkoitus pystyä kaappaamaan. (Manpage of TCPDUMP. 2009.)

5.2 Sääntöviestit

Sääntöjen syöttäminen tähän sovellukseen poikkeaa huomattavasti verrattuna Tcpdumpiin. Säännöt syötetään kontrolliportista, kun sovellus on jo käynnissä. Sovellusten vertailu on vaikeaa, koska molemmat tekevät pääasiallisesti eri asioita. Tcpdump ei ohjaa liikennettä portista toiseen, kuten tämä sovellus tekee.

Tein myös toisen sovelluksen, jonka avulla kontrolliviestit voidaan syöttää verkkoprosessorin kontrolliporttiin. Tämä ei itsessään ollut opinnäytetyön aihealueessa, joten suunnitteluun käytettiin minimaalinen aika. Kontrolliviestisovelluksella ei voi syöttää kuin five tuple -sääntöjä, vaikka ohjelman logiikka pystyy käsittelemään monimutkaisempiakin sääntöjä. Kuvassa 11 on kontrolliviestin syöttöön tehdyn sovelluksen käyttöliittymä.



KUVA 11. Sääntöjen syöttämiseen tehty sovellus

Sovellus muodostaa käyttöliittymään syötettyjen kenttien perusteella binäärimuotoisen viestin. Viesti lähetetään ainoastaan kontrolliporttiin, joten se ei häiritse muuta liikennettä, joka liikkuu prosessorin läpi. Sovellukseen asete-

taan halutut kentät ja valitaan sääntönumero, joka toimii syötetyn säännön tunnuksena. Sääntönumero on tärkeä, koska sen avulla poistetaan kyseinen sääntö, kun sitä ei haluta enää tarkkailla.

Muodostettu binäärimuotoinen data kuljetetaan UDP-protokollan datakentässä. Tätä varten täytyi kehittää kontrolliviestiformaatti, jonka pystyy käsittelemään säännön lisäyksen yhteydessä. Kontrollidatassa jokainen kenttä on pilkottu yhdeksän oktetin mittaisiin osiin. Osien lukumäärä määräytyy tutkittavien kenttien mukaan. Osat jakautuvat vielä kahteen osaan, missä on 1 oktetti kontrolliosaa ja 8 oktettia säännön arvoa eli yhteensä 9 oktettia. Five tuple -säännön lähetyksessä on viisi osaa eli jokaiselle kentälle oma ja täten datan määrä on 45 oktettia. (Kuva 12.)

KUVA 12. Kontrolliviestin rakenne five tuplen tapauksessa

- ETH, Ethernet-otsikko.
- IP, IP-otsikko.
- UDP, UDP-otsikko.
- IP SRC, IP DST, PROTO, PORT SRC, PORT DST (sääntöjen sisältämä data).

Kontrolliviestin kokonaispituus määräytyy otsikkokenttien ja säännön pituuden mukaan. Ethernet-otsikon pituus on 14 oktettia, IP-otsikon 20 oktettia ja UDP-otsikon 8 oktettia. Five tuple -säännön lähetyksessä dataa siis lähetetään yhteensä 14 + 20 + 8 + 45 = 87 oktettia. Jos viestit lähetettäisiin merkkijonona, viestin koko olisi huomattavasti isompi ja viestin muokkaaminen sopivaan muotoon veisi prosessorilta pidemmän ajan.

6 SUORITUSKYKY JA RAJOITTEET

Mittasin sovelluksen suorituskykyä muutamalla eri tavalla. Kuormitustestauksella mittasin läpikulkevien pakettien määrää ja lähetysnopeutta. Kasvatin pakettien määrää niin paljon, että prosessori ei enää ehtinyt käsitellä kaikkia paketteja. Laskin myös erilaisissa tapauksissa, montako prosessorin kellojaksoa paketin käsittelyyn meni.

6.1 Rajoitteet

Verkkoprosessorille voi syöttää noin 36 000 erilaista five tuple -sääntöä. Sääntöjä voi olla hieman enemmänkin, jos ne sisältävät vähemmän kenttiä kuin five tuple -sääntö mahdollistaa. Jos esimerkiksi kaappausta halutaan suorittaa pelkän IP-kohdeosoitteen perusteella, sääntöjä mahtuu muistiin enemmän. Sääntöjen määrää rajoittaa keolle asetettu maksimikoko. Koska jokaiselle säännölle varataan dynaamisesti muistia vain sen verran kuin se tarvitsee, sääntöjen maksimimäärälle ei ole tarkkaa lukuarvoa.

Eri protokollista voi sääntöön määrittää vain tietyt kentät. Alla olevassa luettelossa on määritelty protokollakohtaisesti sallitut kentät:

- Ethernet, lähde-, kohdeosoite tai kumpi tahansa ja seuraavan kerroksen protokolla
- MPLS ja VLAN, tunnistuskenttä, molempia mahdollista ketjuttaa
- IPv4- ja IPv6-, lähde-, kohdeosoite tai kumpi tahansa ja seuraavan kerroksen protokolla
- TCP- ja UDP-, lähde-, kohdeportti tai kumpi tahansa.

Rajoituksia myös asettaa, jos kaapatut paketit halutaan kopioida monitorointiporttiin. Koska monitorointiportin nopeus on vain 1 Gbit/s, läpimenevä liikenne molempiin suuntiin on 1 Gbit/s ja kaikki paketit halutaan kaapata. Tässä tapauksessa monitorointiportin nopeus ei riitä lähettämään kaikkia paketteja, koska liikennettä olisi tällöin yhteensä 2 Gbit/s.

6.2 Suoritusaika

Suoritusaikamittauksiin tein ohjelmakoodiin muutoksia ja lisäsin ohjelman loppuun tulostuksen, josta selvisi kellojaksojen määrä, joka kului paketin käsittelyyn. Käsittelyaika vaihteli hieman, kun testausta tehtiin ohjelman ollessa eri tilassa. Kuten taulukkosta 1 huomaa, paketin käsittelyyn kuluu aina lähes sama kellojaksojen määrä, joten sääntöjen määrä ei vaikuta erityisen paljon paketin käsittelyaikaan.

TAULUKKO 1. Keskimääräinen kellojaksojen määrä paketin käsittelyyn.

Säännöt lkm.	Sääntö osumia	Protokollat	Paketin tyyppi	Kellojaksot
0	Ei	Kaikki tuetut	IP+UDP	701
35 000	Ei	Kaikki tuetut	IP+UDP	452
35 000	Kyllä	Kaikki tuetut	IP+UDP	494
0	Ei	Five tuple	IP+UDP	472
35 000	Ei	Five tuple	IP+UDP	561
35 000	Kyllä	Five tuple	IP+UDP	518
1	Kyllä	Five tuple	IP+UDP	518

Kellojaksojen perusteella voidaan laskea teoreettiset arvot pakettien maksimi määrälle ja minimi koolle (kaavat 1 ja 2).

Minimi paketin koko kellojaksojen suhteen

$$pps = \frac{kellotaajuus}{paketin\,käsittelyn\,kellojaksot} \qquad KAAVA\ 1$$

$$minimi\ paketin\ koko = \frac{linjanopeus}{pps} \qquad KAAVA\ 2$$

pps = paketteja sekuntia kohti

Oletetaan, että paketin käsittelyyn kuluu keskimäärin 550 kellojaksoa, prosessorin kellotaajuus on 500 MHz ja vastaanottonopeus on 1 Gbit/s. Kun lasketaan pienin pakettikoko kaavoja 1 ja 2 käyttäen, saadaan arvoksi 137 tavua.

6.3 Kuormitus

Kuormitusta testasin samalla menettelyllä kuin yksittäisten pakettien prosessointiaikoja, eli eri sääntömäärällä ja erikokoisilla paketeilla. Testeissä siirtonopeus on molempiin suuntiin 1 Gbit/s. Nopeuden tutkimisessa tulee myös ottaa huomioon, että siirtolinjalla (luku 2.2) on Ethernet-protokollan preamble ja jokaisen paketin jälkeen lähettimen täytyy pitää pieni tauko, mikä riippuu lähettimen nopeudesta. (Kurose ym. 2008, 499.)

Kuten edellisestä luvusta huomaa, paketin koolla on suuri merkitys laitteen suorituskykyyn. Mitä isompi paketti, sen enemmän kellojaksoja voi käyttää yhden paketin käsittelyyn. Tässä testauksessa siirtonopeuden pysyessä koko ajan samana kiinnitetään huomiota paketin kokoon ja siihen, montako pakettia laitteen läpi menee, ennen kuin vastaanottopuskurit alkavat täyttyä.

Kuten taulukosta 2 nähdään, laite ei kykene käsittelemään kaikkia paketteja, jos niiden koko jää liian pieneksi. Tietoliikenneverkossa liikkuu monenkokoisia paketteja, mikä tasoittaa laitteelle tulevaa kuormaa. Jos suuri- ja pienikokoiset paketit saapuvat peräkkäin, suurikokoisen paketin käsittelystä jää aikaa käsitellä pienikin paketti, koska lähetysnopeus on rajallinen.

TAULUKKO 2. Pakettien määrä sekunnissa siirtonopeuden ollessa yhteen suuntaan 1 Gbit/s

Paketin koko (oktettia)	Sääntöjen määrä/osumia	Kokonaisnopeus	Paketteja sekunnissa
1 000	0 / Ei	2 Gbit/s	122 065
250	0 / Ei	2 Gbit/s	456 188
1 000	35 000 / Ei	2 Gbit/s	122 065
300	35 000 / Ei	2 Gbit/s	385 791
1 000	35 000 / Kyllä	2 Gbit/s	122 064
300	35 000 / Kyllä	2 Gbit/s	385 801
130	36 000 / Kyllä	1 Gbit/s	811 496

7 YHTEENVETO

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa paketinkaappausohjelma Octeon-verkkoprosessorille. Ohjelmaan täytyi pystyä syöttämään ja siitä poistamaan kaappaussääntöjä ohjelman ollessa ajossa ja verkkoliikenteen mennessä laitteen läpi. Työn vertailu muihin ratkaisuihin on ongelmallista, koska kukaan opinnäytetyössä mukana ollut ei tuntenut vastaavia sovelluksia.

Työssä suurin osa ajasta käytettiin ohjelmarungon suunnitteluun, jotta siitä saataisiin mahdollisimman joustava tuleville laajennuksille. Työ tällaisenaan on hieman yksipuolinen, mutta kaappausrunko on käytännössä hyväksi havaittu. Ohjelmiston koodit sovittiin salaisiksi ennen työn aloitusta, koska uskottiin, että tästä on potentiaalia kehittää jatkossa uusia sovelluksia.

Nyt raporttia kirjoittaessani on suunnittelemani rungon päälle suunniteltu kaksi erilaista kaupallista ohjelmistoa. Voidaan sanoa, että työn suunnittelu on onnistunut erinomaisesti. Molemmat sovellukset käyttävät suunniteltua runkoa, mutta niitä on laajennettu huomattavasti vastaamaan sovelluskohtaisia tarpeita.

Opinnäytetyön alussa ei tehty tarkkaa määrittelyä ohjelmistosta, vaan suunnittelun ja toteutuksen aikana käytiin tilaajan kanssa jatkuvaa keskustelua ominaisuuksista ja ongelmista. Luultavasti tämän ansiosta tästä saatiin näinkin monipuolinen ja muunneltava ohjelmisto. Tämän tyyppinen projektin vetäminen ei luultavasti onnistuisi, jos työtä olisi tekemässä enemmän kuin yksi tekijä, koska nopean ohjelmistokehityksen joustavuus kärsisi.

Kaiken kaikkiaan opinnäytetyö oli erittäin opettavainen. Tietoliikenneverkoissa käytettävät protokollat tulivat tutuksi. Ohjelmoinnissa oppi keksimään uusia tapoja ratkaista suorituskyvyn kannalta kriittisiä ongelmia. Samalla oppi tuntemaan ennestään tuntematonta verkkoprosessorien maailmaa.

LÄHTEET

Cam-0100. 2011. Saatavissa: http://www.cas-well.com/products/CAM-0100.html. Hakupäivä 8.4.2011.

Conte, Tom – Navarro, Nacho – W.Hwu, Wen-mei – Valero, Mateo – Ungerer, Theo 2005. High Performance Embedded Archirtectures and Compilers. Saksa: Springer.

Curtis, June 2010. Octeon Programmer's guide 2010. Saatavissa pyydettävä: css@caviumnetworks.com.

Garcia, Luis Martin 2008. Programming with Libpcap – Sniffing the Network From Our Own Application. Haking Issue vol. 3, nro 2. S. 38–46.

Jaakkohuhta, Hannu 2005. Lähiverkot - Ethernet. Helsinki: IT Press.

Kurose, James F. – Ross, Keith W. 2008. Computer Networking A Top-Down Approach. Boston: Addison-Wesley.

Manpage of TCPDUMP. 2009. Saatavissa: http://www.tcpdump.org/tcpdump_man.html. Hakupäivä 8.4.2011.

Octeon cn52xx. 2011. Saatavissa: http://www.caviumnetworks.com/OCTEON-Plus_CN52XX.html. Hakupäivä 8.4.2011.