



# Muistinhallinta DPDK-pohjaisessa sovelluksessa

Janne Saarela

OPINNÄYTETYÖ  
Toukokuu 2021

Tieto- ja viestintätekniikka  
Sulautetut järjestelmät ja elektroniikka

## TIIVISTELMÄ

Tampereen ammattikorkeakoulu  
Tieto- ja viestintätekniikan tutkinto-ohjelma  
Sulautetut järjestelmät ja elektroniikka

SAARELA, JANNE:  
Muistinhallinta DPDK-pohjaisessa sovelluksessa

Opinnäytetyö 41 sivua  
Toukokuu 2021

---

Opinnäytetyön aiheena oli Data Plane Development Kit (DPDK), joka on verkkopakettien prosessointia nopeuttamaan kehitetty sovelluskehys. Se esiteltiin yleisellä tasolla, kerrottiin sen historiasta ja siitä mitä se sisältää. Työssä kuvailtiin tarvetta pakettiprosessoinnin nopeuttamiseen verkkotoimintojen siirtyessä toimintokohtaisesta laitteistosta yleiskäyttöisille prosessoreille. Lisäksi esiteltiin lyhyesti muitakin pakettiprosessoinnin nopeuttamiseksi kehitettyjä sovelluksia.

Työssä esiteltiin lyhyesti DPDK:ta käyttäviä sovelluksia, kuten Open vSwitch, OpenDataPlane ja MoonGen, sen osoittamiseksi, mitä kaikkea DPDK:ta käyttäen voidaan tehdä. Lisäksi keskityttiin tarkastelemaan DPDK:n tärkeimpiä kirjastoja ja kerrottiin, miten ne hoitavat verkkopaketteja, muistia ja ajastusta. Muistinkäytön osalta kerrottiin ennen muuta muistin sivutuksesta ja DPDK:n muistinkäsittelyn eroista päivitysten 17.11 ja 18.11 välillä. Lisäksi esiteltiin Poll Mode -ajureita, joita käyttäen DPDK kyselee verkkokortille saapuvia paketteja.

Työssä tutkittiin valmista DPDK:n päälle kehitettyä sovellusta ja esiteltiin sen tapaa hoitaa muistin varausta, käyttöä sekä vapautusta. Tämän lisäksi pohdittiin mahdollisia tarpeita ja tapoja muuttaa sovelluksen muistinkäsittely dynaamiseksi, sen sijaan että muisti varattaisiin sovelluksen käynnistyttyä yhteydessä. Työssä testattiin muistinhallinnan muutoksia valmiissa sovelluksessa ja todettiin, ettei nopeasti tapahtuvalle viestinnälle kannattanut tehdä dynaamista muistinvarausta, mutta kerran tapahtuville varauksille pystyttiin käyttämään DPDK:n muistinvarausfunktioita.

## **ABSTRACT**

Tampereen ammattikorkeakoulu  
Tampere University of Applied Sciences  
Degree Programme in ICT Engineering  
Embedded Systems and Electronics

SAARELA, JANNE:  
Memory handling in DPDK based application

Bachelor's thesis 41 pages.  
May 2021

---

The subject of this thesis was Data Plane Development Kit (DPDK) which is a framework meant to speed up packet processing.

In this thesis, a general introduction about DPDK was given including its history and what comes with DPDK. In this work the need for faster packet processing was described as network functions are moving from application-specific hardware to virtualized versions. Other software for faster packet processing were briefly described.

Some applications built on DPDK were introduced to give a picture of what is possible when using DPDK. These applications were Open vSwitch, OpenData-Plane and MoonGen. The DPDK core libraries were reviewed and their use in packet handling, memory handling and timing was presented. Memory paging and development of memory use between DPDK's patch 17.11 and 18.11 were also discussed. Poll Mode Drivers which in DPDK are used for polling network interface cards were discussed.

An application built on DPDK was studied. It's mechanisms for allocating, using and freeing the memory was presented. Possible needs for more dynamic memory handling and new ways to implement those mechanisms in such ways were thought out as the system worked in a way that reserved all needed memory at the beginning of its operations.

The changes in the memory handling of the application were tested, and it was found that it was not useful to use dynamic memory handling in parts where speed was needed. For allocations that were used once, DPDK's memory allocation functions could be used.

---

Key words: DPDK, PMD, virtualization

## SISÄLLYS

1	JOHDANTO .....	6
2	DPDK yleisesti .....	7
2.1	Mikä DPDK on?.....	7
2.2	DPDK:lle syntynyt tarve .....	9
2.3	Ratkaisuja pakettiprosessoinnin hitauteen .....	10
3	DPDK ja käyttäjäsovellus .....	12
3.1	DPDK:n ja käyttäjäsovelluksen sovituserros .....	12
3.2	DPDK:ta hyödyntäviä sovelluksia.....	12
3.2.1	MoonGen.....	12
3.2.2	OpenDataPlane .....	13
3.2.3	Open vSwitch .....	14
4	DPDK pintaa syvemältä .....	15
4.1	Yleistä .....	15
4.2	DPDK:n abstraktiotaso .....	16
4.3	DPDK ja ajastus .....	16
4.4	DPDK ja verkkopaketit .....	17
4.5	DPDK ja muisti .....	18
4.5.1	Kirjastot .....	18
4.5.2	Sivutus.....	19
4.5.3	DPDK:n muistinkäytön kehitys.....	20
4.6	Poll Mode -ajurit .....	21
4.7	DPDK:n suorituskyky .....	23
5	Tutkittu käyttäjäsovellus .....	28
5.1	Käyttäjäsovelluksen muistinhallinta .....	28
5.2	Muistin varaus .....	30
5.3	Varatun muistin käyttöönotto.....	32
5.4	Muistin vapautus .....	34
5.5	Muutokset muistin toimintaan.....	35
6	POHDINTA .....	37
	LÄHTEET .....	39

**LYHENTEET JA TERMIT**

DPDK	Data Plane Development Kit, ohjelmistokehys nopeaa verkkopakettien prosessointia varten.
HPET	High Precision Event Timer, tarkka laitteistoajastin.
Hugepage	Suuri muistisivu, muistin sivutuksessa käytettyjä suuria sivukokoja.
pps	Packets per second, paketteja sekunnissa, esimerkiksi verkkokortin suorituskykyä esittävä arvo.
Netmap	Rengaspuskurityylinen sovellus nopeaa verkkopakettien kuljetusta varten laitteiston ja sovelluskerroksen välillä.
NFV	Network Function Virtualization, verkkotoimintoa suorittavan fyysisen laitteen luonti virtuaalisena.
NIC	Network Interface Card, verkkokortti, tietokoneen verkkoon yhdistävä laite.
NPU	Network Processing Unit, ohjelmoitava verkkoprosessori eri verkkolaitteiden kehitystä varten.
PF_RING	Rengaspuskurimoduuli nopeaa verkkopakettien siirtoa varten.
PMD	Poll Mode Driver, DPDK:n mukana tulevia ajureita, joilla kysellään verkkokorttiin saapuvia paketteja.
RSS	Receive Side Scaling, tekniikka, jolla voidaan jakaa saapuvat verkkopaketit jonoihin ja siitä edelleen eri prosessoriytimille prosessoitavaksi tasaten prosessoinnin aiheuttavaa kuormaa.
TLB	Translation Lookaside Buffer, puskuuri, joka sisältää viimeksi käytetyn muistin fyysisiä muistiosoitteita vastaavat virtuaaliset osoitteet.
TSC	Time-Stamp Counter, prosessorin kellojaksoja laskeva laskuri.

## 1 JOHDANTO

Alati kehittyvissä ja nopeutuviissa järjestelmissä jokin osa on aina hitain. Tämän työn tapauksessa verkkotoimintoja kehitettäessä yleiskäyttöisiä prosessoreita käyttäen, on hidastavaksi tekijäksi ilmennyt käyttöjärjestelmien tapa prosessoida verkkopaketteja.

Tämän opinnäytetyön tarkoituksena on tutustua Data Plane Development Kit -ohjelmistokehykseen yleisesti käsittäen DPDK:n kirjastoja, ajureita ja syitä sen käytön tarpeellisuudelle. Syvemmin tutustutaan DPDK:n muistinkäyttöön ja kuinka verkkopakettien käsittely tapahtuu. Työssä syvennyttään tarkemmin DPDK:n ja työpaikalla käytetyn valmiin sovelluksen muistia käyttävään osaan ja pohditaan, onko tarvetta muutoksille.

Opinnäytetyön tavoitteena on tuottaa tekijälle kuva DPDK:n toiminnasta, sen käytön tarpeesta, sen eri osasista sekä yleisesti kasvattaa tietämystä laitteistosta ja ohjelmistosta. Lisäksi tavoitteena on tutustuttaa tekijä työpaikalla käytettävän järjestelmän yksittäiseen osaan tulevaisuudessa tehtäviä töitä tukemaan.

Opinnäytetyön lähdemateriaalina on käytetty saatavilla olevaa kirjallisuutta, artikkeleita, sekä työn aiheena olevan ohjelmistokehyksen dokumentaatiota. Työ tehtiin yritykselle Nokia Oyj, joten työssä tutkittavan valmiin sovelluksen selvitys on kerrottu hyvin pintapuolisesti.

## 2 DPDK yleisesti

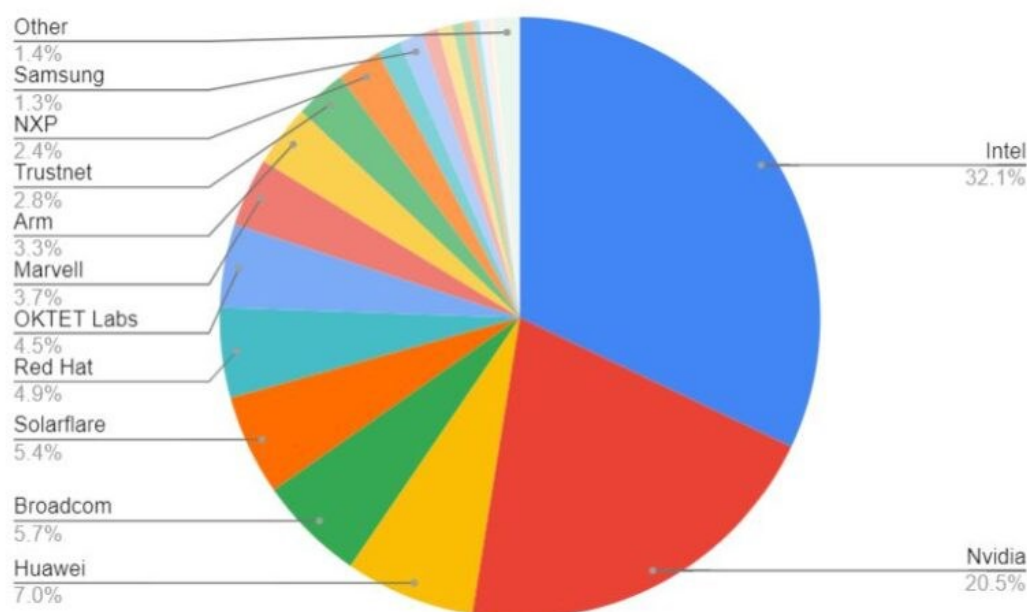
### 2.1 Mikä DPDK on?

DPDK on sovelluskehitysalusta, jota käytetään nopeaan verkkopakettien prosessointiin tarkoitettujen käyttäjätasossa toimivien sovellusten pohjana. Alustan mukana tarjotaan monia kirjastoja, jotka luodaan tapauskohtaisesti halutulle laitteistolle. (DPDK overview, n.d.)

Tässä työssä DPDK:ta käytetään Linux-käyttöjärjestelmän Ubuntu-jakelussa, mutta se on saatavilla monille muillekin käyttöjärjestelmille, kuten Debian, Red Hat ja FreeBSD. DPDK on tekeillä myös Windowsille. Tässä työssä mainitut asiat pätevät Linux-ympäristöön.

DPDK on Intelin vuonna 2010 aloittama projekti. Vuonna 2013 ranskalainen verkko-ohjelmistoyritys 6WIND avasi verkkosivun DPDK.org, joka on toiminut projektin kotisivuna siitä lähtien. Vuonna 2017 DPDK siirtyi Linux Foundationin alaiseksi projektiksi. (DPDK about, n.d.)

Kuviosta 1 nähdään nykyään DPDK:n mukana olevia yrityksiä, joista suurimpana on työn alulleen laittanut Intel ja muita merkittäviä osallistujia ovat esimerkiksi Nvidia, Broadcom ja Huawei ja nämä omalta osaltaan osallistuvat DPDK:n kehitykseen (DPDK 20.11, 2020).



KUVIO 1. DPDK-päivityksen 20.11 suurimmat osallistajat (DPDK 20.11, 2020)

DPDK tukee yleisimpiä prosessoriarkkitehtuureja kuten x86, arm ja powerpc sekä eri verkkokortteja valmistajilta kuten Mellanox, Intel ja Huawei (DPDK supported, n.d.). Alkuun DPDK tuki vain Intelin prosessoreita mutta ajan saatossa tuki on kasvanut huomattavasti.

Kuten mainittiin, DPDK tarjoaa sovelluskehittäjälle monia kirjastoja pakettiprosessointia varten. Tärkeimpiä kirjastoja on viisi, tai kuusi jos malloc otetaan mukaan. Nuo muut viisi ovat EAL eli Environment Abstraction Layer, ring-kirjasto rengaspuskureita varten, mempool-kirjasto muistin varausta varten, mbuf-kirjasto verkkoviestien kuljetusta varten ja timer-kirjasto eri toimintojen ajastusta varten.

DPDK:n erikoisuutena on myös monien suurien verkkokorttivalmistajien (Cisco, Mellanox) kirjoittamat ajurit, PMD:t, eli Poll Mode Driverit. Näiden ajureiden ideana on omalta osaltaan nopeuttaa pakettien käsittelyä nimensä mukaisesti pollaamalla eli kyselemällä verkkokortin resursseja sen sijaan että verkkokortti lähettäisi prosessorille keskeytyspyynnön kuten normaalissa ympäristössä tapahtuu.

Normaalisti PC-ympäristössä verkkokortti eli NIC (Network Interface Card) toimii siten, että verkosta saapuva viesti tulee kortille, kortti siirtää sen oikosiirrolla suoraan systeemin muistiin, prosessorille annetaan keskeytyspyyntö, jonka jälkeen prosessori käsittelee paketin.

## 2.2 DPDK:lle syntynyt tarve

Verkkojärjestelmissä on monia eri toimintoja ja näitä hoitamassa monet erilaiset laitteet kuten reitittimet, kytkimet, palomuurit ja kuormituksen tasaajat. Ennen nämä toiminnot tai verkkofunktiot toteutettiin pitkälti tarkoitukseen suunnitelluilla ja valmistetuilla ASIC-piireillä. ASIC:ien ongelmana ovat pitkä ja kallis kehitys, mutta korkeilla valmistusmäärillä yksittäisen piirin hinta saadaan tuotua alas. Mukaan on tullut myös FPGA-piirit, eli laitteistokuvauskielillä uudelleenohjelmoitavat piirit. FPGA-piireillä voidaan luoda prototyyppejä halutuista verkkofunktioista. FPGA-piirien käyttö tuo mukanaan myös tarpeen näiden laitteistokuvauskielten osajille. ASIC- ja FPGA-piirien lisäksi markkinoilla on pakettiprosessointia varten kehitettyjä NPU-järjestelmiä eli verkkoprosessoreita. Näiden piirien ominaisuuksia on niiden ohjelmoitavuus, moniytiminen rinnakkainen suorittaminen sekä verkkotoimintoihin liittyvät asiat kuten datan koodaus ja dekodeaus. NPU-järjestelmien hyödyiksi mainitaan juuri tuon ohjelmoitavuuden ja korkean suorituskyvyn mutta huonoja puolia näissä piireissä ovat niiden kapea käyttötarkoitus ja valmistajakohtainen käskykanta, jonka vuoksi kehittäjien täytyy syvällisesti tutustua laitteen datalehteen ja käytettävään käskykantaan. (Heqing Zhu, 2021, 5, 6).

Vuonna 2012 pidetyssä ETSI:n konferenssissa teleoperaattorit tekivät aloitteen NFV:n eli verkkofunktioiden virtualisoinnin puolesta. Toisin sanoen näitä mainittuja verkkotoimintoja toteutettaisiin sovelluskohtaisen laitteiston sijaan ohjelmallisesti. (Ying Zhang, 2018, kappale 1.)

Verkkotoimintojen virtualisoinnin ajatuksena on erotella nuo toiminnot alla olevasta laitteistosta. Tämän ansiosta verkkotoimintoja voitaisiin toteuttaa ohjelmallisesti ja ajaa niitä yleiskäyttöisillä prosessoreilla. Tästä syntyy rahallisia säästöjä sillä nuo aiemmin mainitut toimintokohtaiset laitteet ovat kalliita. Virtualisointi antaa myös joustavuutta verkkotoimintoihin, kun niitä voidaan tarpeen vaatiessa

käynnistää palvelimella virtuaalijärjestelmiin ja lopettaa tarvittaessa näin tehden järjestelmästä dynaamisemman. (Heqing Zhu, 2021, 267.)

Aiemmin yleiskäyttöiset prosessorit olivat nopeita, mutta niissä ei ollut monia ytimiä. Moniytimisyys oli lähinnä noiden verkkotoimintojen tarpeisiin kehitettyjen piirien ominaisuus, mutta nykyään myös yleiskäyttöisissä huokeissakin prosessoreissa voi olla monta ydintä. Tämä on avannut mahdollisuuksia tuohon aiemmin mainittuun virtualisointiin ja pakettiprosessointia suorittavien ohjelmien kehittämiseen yleisillä ohjelmointikielillä. (Heqing Zhu, 2020, 4.)

Verkkoliikenteen määrä on koko ajan kasvussa ja tämän takia tarve entistä nopeammalle pakettiprosessoinnille kasvaa. Verkkokortit kykenevät nykyään siirtämään jo 40 gigabittiä sekunnissa, mutta pakettien prosessointi laahaa perässä käyttöjärjestelmien ytimien verkkopinojen vuoksi. (Cerović ym. 2018, 1.)

Verkkoliikenteen jatkuvasti kasvaessa, virtualisoinnin yleistyessä ja yleiskäyttöisten prosessorien ja verkkokorttien kehittyessä aina vain paremmiksi, eivät itse verkkoliikenteen paketteja käsittelevät tavalliset verkkopinot enää pysy perässä. Tähän pulmaan on koetettu kehittää erilaisia ratkaisuja.

### **2.3 Ratkaisuja pakettiprosessoinnin hitauteen**

Pakettiprosessoinnin hitautta on koetettu ratkaista erilaisilla keinoilla ja tekniikoilla. Näitä ovat esimerkiksi RSS, Netmap, PF\_RING ja tämän työn aiheena oleva DPDK. RSS eli Receive Side Scaling on tekniikka, jolla montaa RX/TX-jonoa tukeva verkkokortti lähettää vastaanottamansa paketin suodattimen sääntöjen perusteella eri jonoihin, joista paketit sitten jaetaan ytimille suoritettaviksi (Herbert, Bruijn, n.d.; Cerović ym. 2018, 4). Netmap taas on ohjelmistokehys, jonka tarkoituksena on nopeuttaa pakettien prosessointia siten, että se erottaa verkkokortin käyttöjärjestelmän verkkopinosta ja siirtää paketteja sen sijaan Netmap-rajapinnan kautta käyttäjäsovellukselle (Rizzo, L 2012.; Cerović ym. 2018, 7). PF\_RING on Netmapin tyylinen. PF\_RING-dokumentaation mukaan se on paketinkaappauskirjasto, jolla voidaan tavallisesta tietokoneesta muuttaa tehokas ja huokea mittalaite verkkoliikenteen mittaamiseen ja sitä käytetään niin, että

verkkokortti siirtää saamansa datan rengaspuskuriin, josta käyttäjäsovellus sitten voi ottaa samat paketit käsiteltäväksi (PF\_RING dokumentaatio n.d.; Wenjun, Z ym. 2018, 2).

### **3 DPDK ja käyttäjäsovellus**

#### **3.1 DPDK:n ja käyttäjäsovelluksen sovituseros**

Suuremmissa ohjelmistokokonaisuuksissa ei ole järkevää, että käyttäjäsovelluksesta ollaan suoraan yhteydessä DPDK:n tarjoamaan rajapintaan. Väliin tarvitaan sovituseros, joka toimii DPDK:n ohjelmointirajapinnan ja oman sovelluksen välissä olevana kerroksena. Tätä kerrosta kutsutaan omasta sovelluksesta tekemään eri operaatioita ja se käyttää DPDK:n tarjoamaa rajapintaa, jota olisi vaikea käyttää sellaisenaan omasta sovelluksesta. Sovellus myöskään ei tällöin ole niin altis DPDK:n muutoksille.

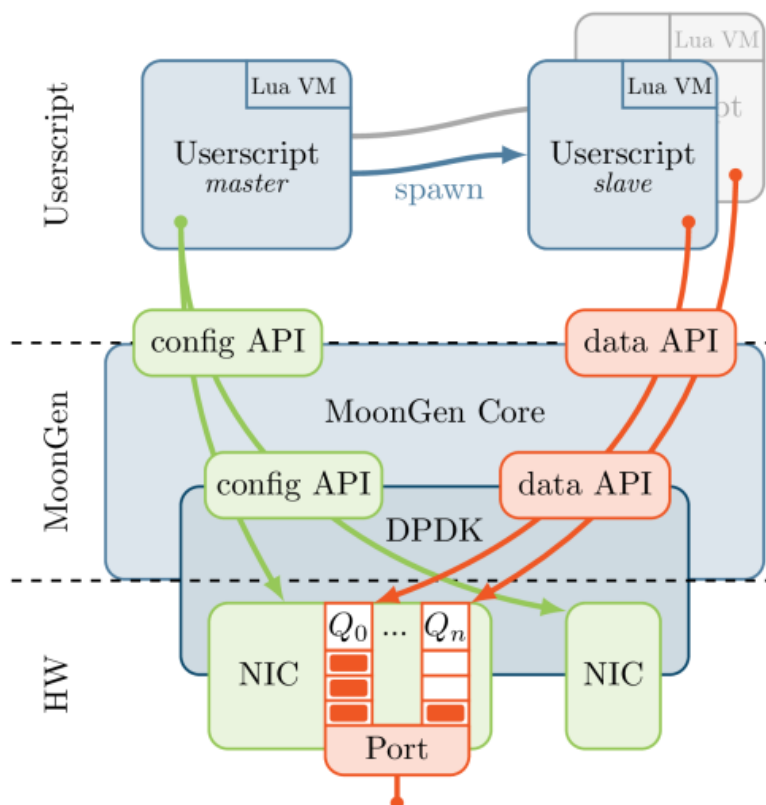
#### **3.2 DPDK:ta hyödyntäviä sovelluksia**

DPDK tarjoaa työkalut nopeaan pakettien prosessointiin, ja DPDK:ta käyttäen onkin kehitetty monia ohjelmia kuten esimerkiksi Moongen, joka on pakettigeneraattori, Open Data Plane joka toimii sovituseroksena ja Open vSwitch joka on virtuaalinen kytkin jota käytetään esimerkiksi palvelimissa virtuaalikoneiden yhdistämiseen.

##### **3.2.1 MoonGen**

Moongen on LUA-skriptikielellä LuaJIT-kääntäjää käyttäen DPDK:n päälle kirjoitettu pakettigeneraattori. Näillä skripteillä ohjattava pakettigeneraattori tarjoaa käytön joustavuutta verrattuna laitteistolla toteutettuihin tai muihin ohjelmallisesti toteutettuihin generaattoreihin. Moongenillä on saatu minimikokoisia paketteja ja yhtä prosessoriydintä käyttäen 14,88 Mpps nopeuden 10 gigabitin Ethernet-rajapinnasta mitattuna. (Emmerich, 2015, 1.) MoonGeniä ohjaillaan käyttäjäavaruudesta LUA-skripteillä. Käyttäjän skripti, joka sisältää tiedon pakettien generoinnista, ajetaan ensin, käynnistäen pääfunktion, joka alustaa ja konfiguroi tarvitta-

van laitteiston ja viestipuskurit. Tämän jälkeen käynnistetään alifunktio tai funktioita, jotka lähettävät tai vastaanottavat paketteja MoonGenin ja sen alla olevan DPDK:n läpi (kuvio 2). (Emmerich, 2015, 3).



KUVIO 2. Moongenin arkkitehtuuria (Moongen github, n.d.)

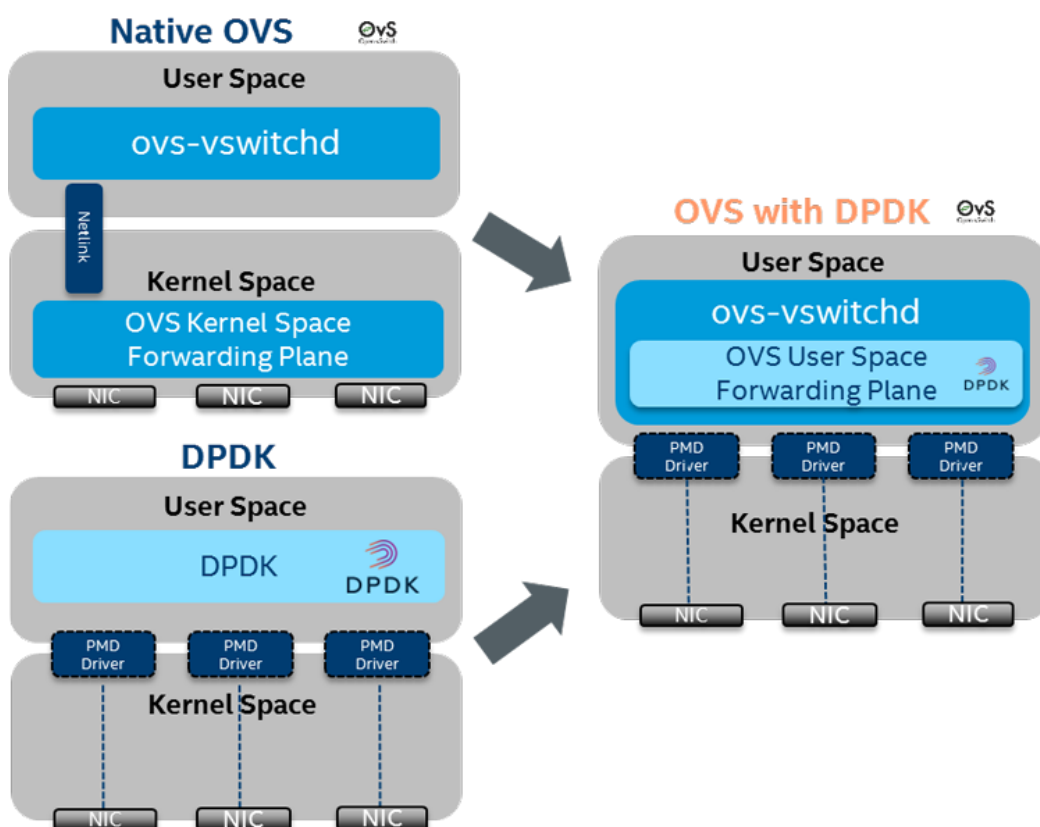
Tällaista pakettigeneraattoria voidaan käyttää esimerkiksi lähettämään jollekin tai vastaanottaen joltakin testattavalta systeemiltä paketteja, näin esittäen oikeaa verkkoliikennettä.

### 3.2.2 OpenDataPlane

ODP on DPDK:n päälle rakennettu abstraktoitu ohjelmointirajapinta, jonka tarkoituksena on yksinkertaistaa alla olevan laitteiston käyttöä, kun tehdään tietoliikenneverkon datatason ohjelmia. ODP:n hyötynä on myös sen helppo siirrettävyys eri laitteistoille, kunhan ne vain tukevat ODP:tä. Aivan kuten DPDK:kin, myös ODP tukee monia erilaisia prosessoriarkkitehtuureja kuten x86, ARM ja niin edelleen. (ODP faq n.d.)

### 3.2.3 Open vSwitch

Open vSwitch on käyttäjäavaruudessa toimiva virtuaalinen kytkin. Nykyään siinä on kaksi käytettävää datapolkua, natiivi- ja DPDK-datapolku. Tämä natiivi datapolku vie paketit käyttöjärjestelmän ytimen kautta, mutta kuten jo DPDK:n oleellisista ominaisuuksista mainittiin, DPDK datapolun kautta paketit kulkevat laitteistolta eli verkkokorteilta suoraan käyttäjätasoon (kuvio 3) (Giller, 2016).



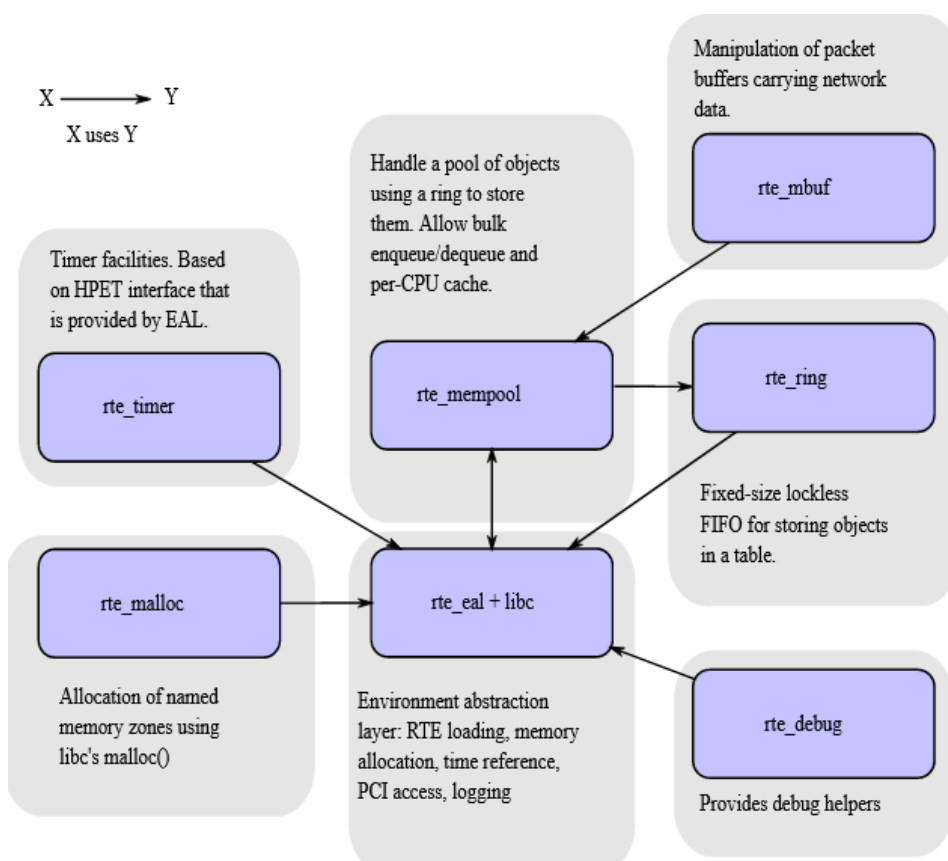
KUVIO 3. Open vSwitch DPDK:n kanssa ja ilman (Giller 2016)

Open vSwitchin natiivia datapolkua käyttäen viestit kulkevat Linux-ytimen verkkopinon kautta. Natiivi datapolku ei silti riitä ihan kaikkeen, ja tässä tulee apuun DPDK joka, kuten jo mainittu, ohittaa ytimen ja yhdistää verkkokortit suoraan käyttäjäavaruuteen DPDK:n ominaisten Poll Mode -ajureiden avulla. (Giller, 2016.)

## 4 DPDK pintaa syvemmältä

### 4.1 Yleistä

DPDK:n mukana tulee joitakin peruskirjastoja, joiden päälle voidaan rakentaa oma käyttäjäsovellus. Noita kirjastoja on viisi kappaletta ja niihin kuuluu ajastuksen hallinta timer-kirjastolla, muistiasioiden hallinta ring- ja mempool-kirjastoilla, viestipuskureiden hallinta mbuf-kirjastolla sekä nämä yhteen sitova kirjasto EAL, joka on tavallaan se alin kerros ennen laitteistoa, jonka päälle DPDK käännetään (kuvio 4). Tämän lisäksi on myös DPDK:n oma malloc-kirjasto, joka tarjoaa C-ohjelmointikielen malloc-funktion toiminnallisuudet DPDK:n lisäämillä ominaisuuksilla. DPDK tukee C ja C++ ohjelmointikieliä.



KUVIO 4. DPDK:n tärkeimmät kirjastot (DPDK overview n.d.)

DPDK:n mukana tulee muitakin kirjastoja mutta tässä osassa keskitytään edellä mainittuihin keskeisiin kirjastoihin. Kirjastoihin perehtymisen lisäksi pohditaan myös tarkemmin sitä, miten DPDK:ta käyttäen erilaisia asioita hoidetaan, kuten ajastusasiat, sillä DPDK:n yhteydessä ei käytetä Linuxin vuorottajaa jakamaan

prosesseja eri prosessoriytimille mutta eri prosesseja voidaan suorittaa eri ytimillä siten, ettei niitä käytetä mihinkään muuhun.

Muistinkäsittelystä myös mennään syvemmin DPDK:n tarpeeseen käyttää Hugepageja ja asioihin kuten TLB ja niin edelleen.

## 4.2 DPDK:n abstraktiotaso

EAL eli Environment Abstraction Layer on DPDK:ssa se kerros, joka pääsee käsiksi käytettävän laitteen rautaan eli muistiin ja I/O-laitteisiin. EAL on geneerinen rajapinta ja se tarkoitus on tarjota paljon erilaisia toimintoja käytettäväksi ja se on myös vastuussa alustuksista ja käytettävien resurssien allokoinnista. EAL sisältää monia eri ominaisuuksia kuten itse DPDK:n lataaminen ja käynnistys, prosessien sijoittaminen eri ytimille, muistin varaaminen ja muuta. (DPDK EAL. n.d.)

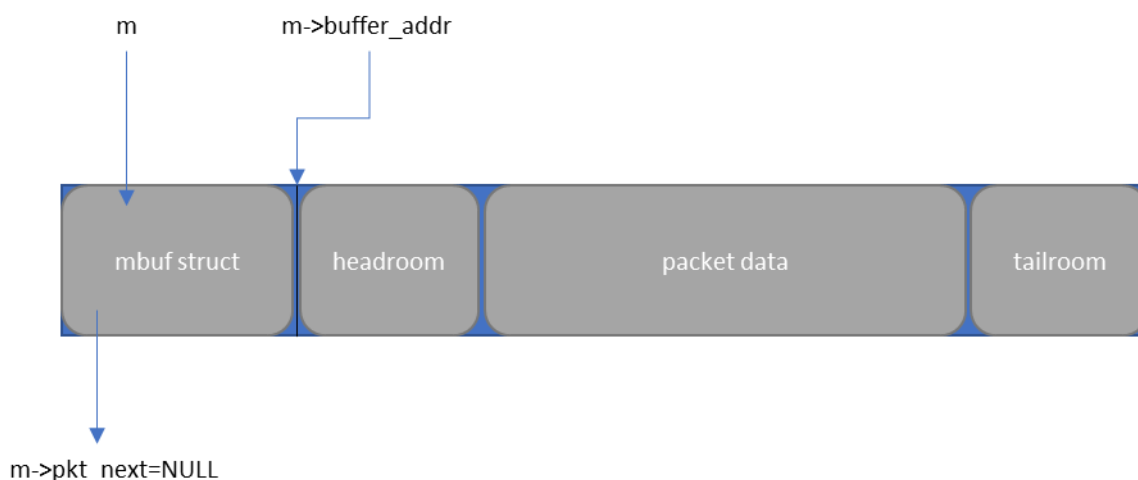
## 4.3 DPDK ja ajastus

DPDK:ta käytettäessä voidaan prosessoriytimille määrittää suoritettavaksi tietty toiminnallisuus siten että se suorittaa ikuista silmukkaa sille määrätyllä ytimellä. Tämän ansiosta käyttöjärjestelmä ei katkaise prosessin suoritusta kesken kaiken ja anna ajoaikaa jollekin muulle prosessille. DPDK:ssa voidaan silti ajastaa eri tapahtumia.

Dokumentaation mukaan käyttöä tälle timer-kirjaston ajastukselle olisi lähinnä roskien siivoaminen ja tilakoneille. Tuon kirjaston ajastimet voivat toimia jaksoittain tai kerran, jolloin ne pitää käynnistää erikseen uudelleen uutta käyttöä varten. Ajastimien tarkkuus riippuu siitä, kuinka usein kirjaston `rte_timer_manage`-funktiota kutsutaan. Usein kutsuttuna tarkkuus paranee, mutta se laskee suorituskykyä. Kirjasto saa aikansa EAL:n kautta, joka saa referenssiaikansa prosessorin ajastimilta TSC tai HPET (DPDK Timer-library, n.d.; DPDK EAL, n.d.)

#### 4.4 DPDK ja verkkopaketit

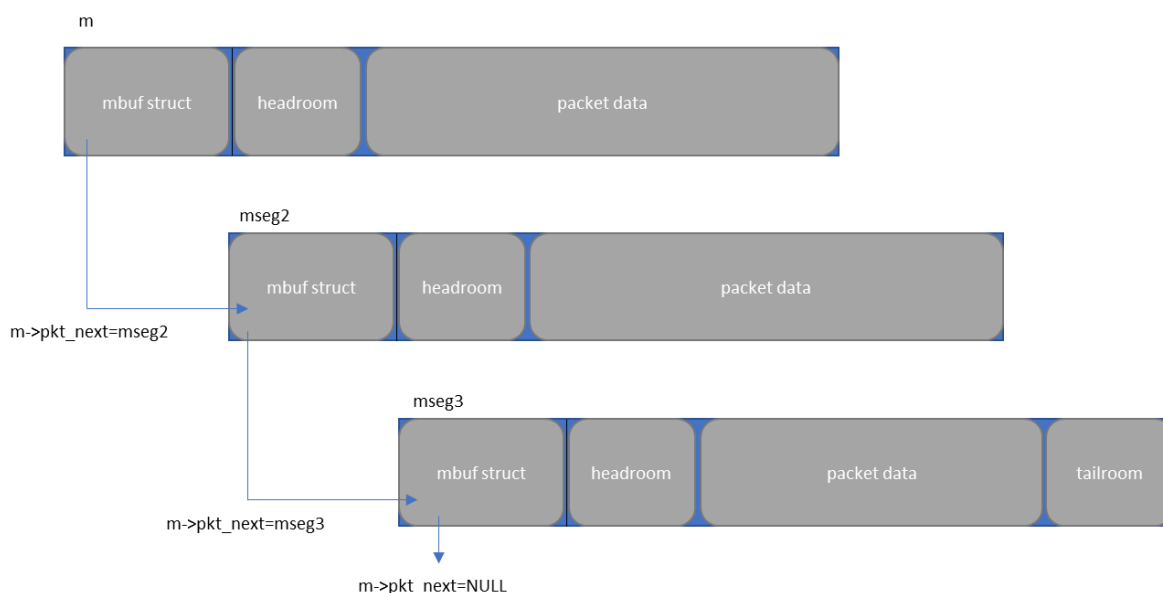
Verkkokortin portteihin sisään tulevien ja ulos lähetettävien pakettien säilöntään käytetään mbuf-muistipuskureita, jotka on määritelty mbuf-kirjastossa. DPDK:n dokumentaation mukaan mbuf-puskuri on koostettu siten, että paketin metadata sijoitetaan muistipuskuriin, jonka perässä on vakiokokoinen alue pakettidataa varten (kuvio 5). Tämän alueen koko on kuitenkin muutettavissa mutta yleensä vakiokooksi on asetettu 2 kilotavua. (Heqing Zhu, 2021, 129.)



KUVIO 5. Mbuf-muistipuskurin rakenne

Kuviossa 5 rakenteen headroom- ja tailroom-osiot ovat paketin muokkausta varten valmiina olevaa tilaa. Muokkaus voi olla esimerkiksi protokollan otsikkokehysten muutos tai paketin eheyden tarkastaminen. (Heqing Zhu, 2021, 130.)

Mbuf-puskureita voidaan myös ketjuttaa, joten ne tukevat niin kutsuttuja jumbo frame -paketteja, jotka ovat sellaisia Ethernet-paketteja, joiden tietosisältö on enemmän kuin 1500 tavua (IBM Jumboframes, n.d.). Ketjutus tapahtuu siten että vain ensimmäisessä puskurissa on paketin metadata ja tieto seuraavasta mbuf-puskurista, jossa pakettidata jatkuu (kuvio 6). Kuviossa 5 näkynyt arvo pkt\_next ei ole enää NULL, vaan se on saanut arvokseen ketjussa seuraavaksi tulevan puskurin osoitteen. Huomataan myös, että seuraavaan puskuriin jatkuvassa puskurissa on koko puskurin loppuosa otettu pakettidatan käyttöön.



KUVIO 6. Ketjutettuja mbuf-puskureita

Mbuf-kirjasto käyttää mempool-kirjastoa varatakseen muistialtaan puskureille ja ne sisältävät tiedon siitä, mihin muistialtaaseen ne kuuluvat (DPDK API reference mbuf n.d.). Paketin tullessa verkkokortille, pyytää PMD puskuria, joka otetaan siihen tarkoitukseen alustetusta muistialtaasta ja paketin lähtiessä puskuri vapautetaan takaisin muistialtaaseensa (Heqing Zhu, 2021, 132). Kirjaston puskureita voidaan verkkopakettien lisäksi käyttää myös järjestelmän sisällä muuhunkin viestintään kuten ohjausdatan siirtoon (DPDK mbuf n.d.).

## 4.5 DPDK ja muisti

### 4.5.1 Kirjastot

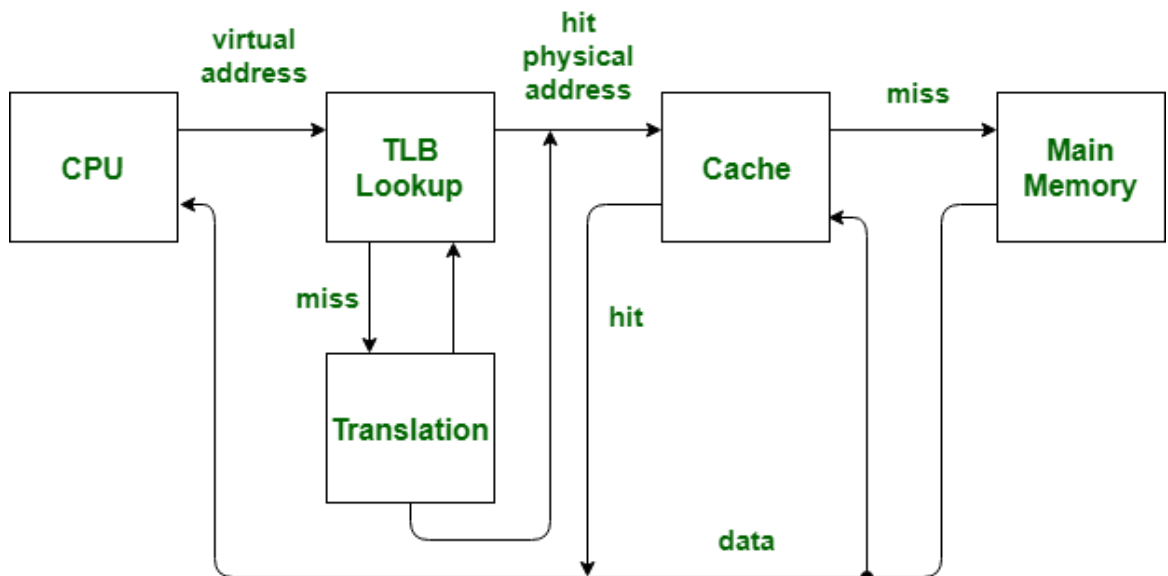
DPDK:n keskeisissä kirjastoissa on kaksi kirjastoa, jotka oleellisesti liittyvät muistin hallintaan. Nämä ovat mempool ja ring. Mempool-kirjasto on ennalta määritetyn kokoisten objektien allokointia varten ja allokointi tapahtuu altaalle annetun nimen perusteella. Kirjasto sisältää myös prosessoriydinkohtaisen välimuistin luonnin ja tasausta (alignment) helpottavia toimintoja jotta "objektit" jakautuvat tasaisesti muistiin.

Mempool-kirjasto käyttää ring-kirjastoa ja luo muistialtaat rengastyyliksi. Toinen vaihtoehto on käyttää pinotyyllisiä muistialtaita. Lisäksi noita rengastyyllisiä jonoja voidaan käyttää myös eri ytimille asetettujen prosessien väliseen viestintään. Jos rengaspuskurityylistä muistiallasta käyttää moni prosessori, voidaan prosessorityimille varata ydinkohtainen välimuisti muistialtaaseen. Tähän pinotyylliseen välimuistiin voidaan rengaspuskurista hakea kerralla enemmän dataa kunkin prosessorityimen käsiteltäväksi näin vähentäen muistialtaan rengaspuskurille tapahtuvia kyselyitä. (DPDK mempool n.d.; DPDK ring n.d.)

#### 4.5.2 Sivutus

Sivutus (paging) on yleinen käyttöjärjestelmien tapa hallita tietokoneen muistia. Siinä fyysinen muisti sivutetaan virtuaaliseksi muistiksi, joka on jaettu eri kokoihin sivuihin kuten 4 kB, 2 MB ja 1 GB. Näitä virtuaalisia osoitteita vastaavat fyysiset osoitteet haetaan TLB:n (Translation Lookaside Buffer) kautta, ja mitä suurempia sivut ovat, sitä vähemmän on sivuja vastaavia osoitteita. Jos sivua ei löydykään TLB:stä, tapahtuu ”huti”, oikea sivu joudutaan etsimään ja tämä vie aikaa. Kun sivun osoite on haettu, se asetetaan TLB:hen. TLB:n tarkastuksen jälkeen haetaan osoitetta vastaava data välimuistista, ja jos se ei ole siellä, joudutaan se hakemaan muistista (kuvio 7).

DPDK dokumentaation mukaan DPDK:n käyttämä fyysinen muistin varataan EAL:n toimesta hugelbs-tiedostojärjestelmästä käyttämällä mmap()-funktia. Näitä hugepageja käytetään sen vuoksi että DPDK tarvitsee suuria muistialtaita pakettipuskureille ja suuria sivuja käyttämällä TLB:n tarvitsee harvemmin etsiä sivuja vastaavia fyysisiä osoitteita, näin parantaen suorituskykyä. (DPDK System Requirements, n.d.)



KUVIO 7. Muistin toimintaa (Geeksforgeeks 2020)

DPDK:ssa voidaan mahdollisesti siirtää pienessä ajassa hyvin paljon dataa ja pienten sivujen käyttö lisää näiden TLB-hutien riskiä, eli kun sovellukselle annetaan suurempia sivuja käyttöön, sivuja on vähemmän, ja suuremmalla todennäköisyydellä näin löytyvät TLB:stä.

#### 4.5.3 DPDK:n muistinkäytön kehitys

DPDK:n muistinkäyttö muuttui oleellisesti versioiden 17.11 ja 18.11 välillä. Aikaisemmissa versioissa sovelluksen kehittäjän täytyi tietää, kuinka paljon muistia tarvittiin ennen sovelluksen käynnistystä ja tämän perusteella hugepageja varattiin tarvittava määrä DPDK:n käytettäväksi. Uudemmassa versiossa tämä varailu muuttui dynaamiseksi niin, että DPDK sovellus voi varata ja vapauttaa hugepageja tarpeensa mukaan.

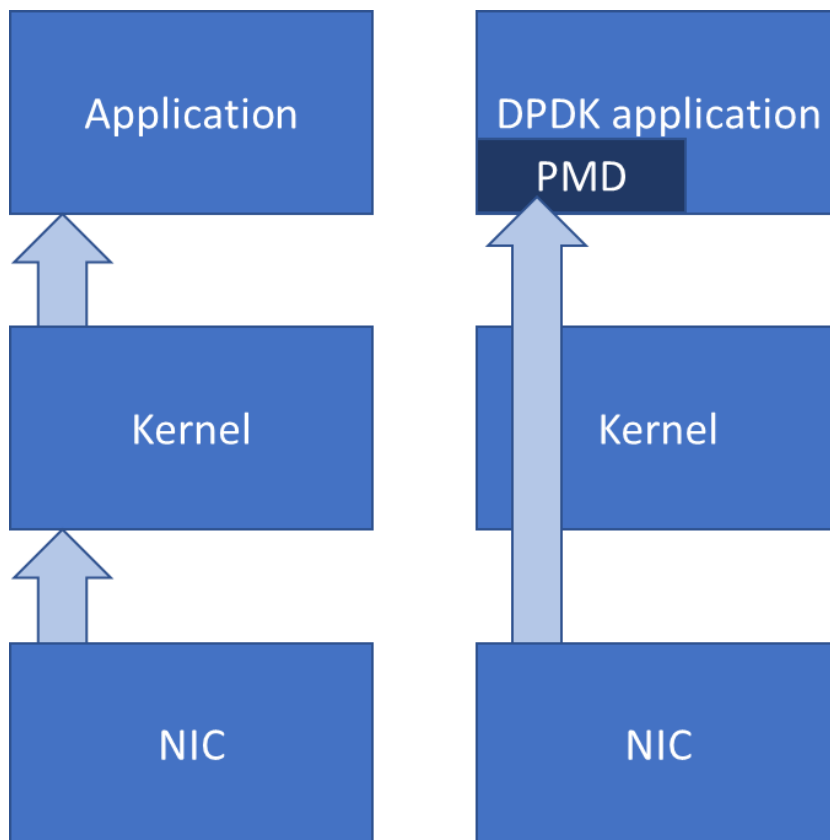
Versiossa 17.11 ja sitä ennen siis tarvittavan muistin määrä oli tiedettävä etukäteen sillä muistia ei ollut mahdollista varata lisää tai vapauttaa alustamisen jälkeen. Tässä versiossa hugepage-varaus myös toimi siten, että jos EAL:lle ei erikseen määritetty halutun muistin määrää, otti DPDK kaikki mahdolliset hugepaget jotka se vain sai järjestelmältä. (Burakov 2019.)

Uudemmassa versiossa 18.11 jossa muistinhallinnassa oli tehty oleellisia muutoksia, DPDK ei enää vaatinut alustusvaiheessa muistia, vaan se käynnistettiin ilman varattua muistia, ja kykeni varaamaan ja vapauttamaan muistia sen mukaan mitä prosessit tarvitsivat. Tämä muutos antoi etua siinä, ettei enää tarvinnut etukäteen tietää kuinka paljon muistia tarvitsee varata DPDK:lle ja jokaiselle prosessille sekä siinä että DPDK ei yksin vie kaikkea tarjolla olevaa muistia, jota muutkin ohjelmat voisivat tarvita. (Burakov 2019.)

Lisäksi vielä tiedoksi, että tässä dynaamisessa moodissa mikä tahansa DPDK:n muistia varaava toiminto kuten `rte_malloc` ja `rte_memzone_reserve` varaavat lisää hugepageja ja vapautettaessa ne sivut vapautetaan takaisin järjestelmälle. Muutoksien myötä ei silti tehty vanhoja sovelluksia täysin toimintakyvyttömiksi sillä DPDK:n uutta versiota voidaan ajaa "legacy-modessa" joka mukailee vanhaa edellisten versioiden tapaa hallita muistia. (DPDK EAL n.d.)

#### **4.6 Poll Mode -ajurit**

Kun verkkokortti vastaanottaa paketin verkosta, tämä paketti siirretään verkkokortin muistiin, jonka jälkeen tehdään keskeytyspyyntö käyttöjärjestelmän ytimelle, paketti siirretään ytimen toimesta verkkokortin muistista pois ytimen verkkopinon käsiteltäväksi ja tämän kautta paketti annetaan eteenpäin käyttäjäavaruuteen sovelluksen käyttöön. (Overview of Packet Reception n.d.) Poll Mode -ajurit taas hakevat paketteja suoraan verkkokortilta ohittaen ytimen verkkopinon (kuvio 8).



KUVIO 8. Verkkopaketin kulku tavallisessa Linux-ympäristössä ja DPDK-ympäristössä

Keskeytyksen käyttö merkitsee sitä, että prosessorin täytyy vaihtaa prosessia (context switch), ja suorittaa tämä keskeyttävä prosessi. Kontekstin vaihto vaatii aikaa, sillä ennen keskeytystä suoritettavan prosessin data on siirrettävä rekistereistä talteen, jotta prosessia voidaan keskeytyksen jälkeen jatkaa. Tämä keskeytyksillä tapahtuva pakettien vastaanottaminen ei välttämättä ole haitallista, jos prosessori on huomattavasti I/O-laitteen, tässä tapauksessa verkkokortti, suoritussnopeutta nopeampi, mutta ongelmia ilmenee, kun siirrytään huomattavasti nopeampiin verkkokortteihin (Heqing Zhu, 2021, 12).

Datansiirtonopeuden kasvaessa vastaanotettavien ja lähetettävien pakettien määrä voi olla niin suuri, että keskeytyksiä tulee jatkuvasti. Verkkokortin ajurin voi olla mahdollista helpottaa tilannetta käsittelemällä yhden keskeytyksen aikana monta pakettia mutta lopulta tämä jatkuvien keskeytysten aiheuttama "hinta" voi kasvaa liian korkeaksi (Heqing Zhu, 2021, 12).

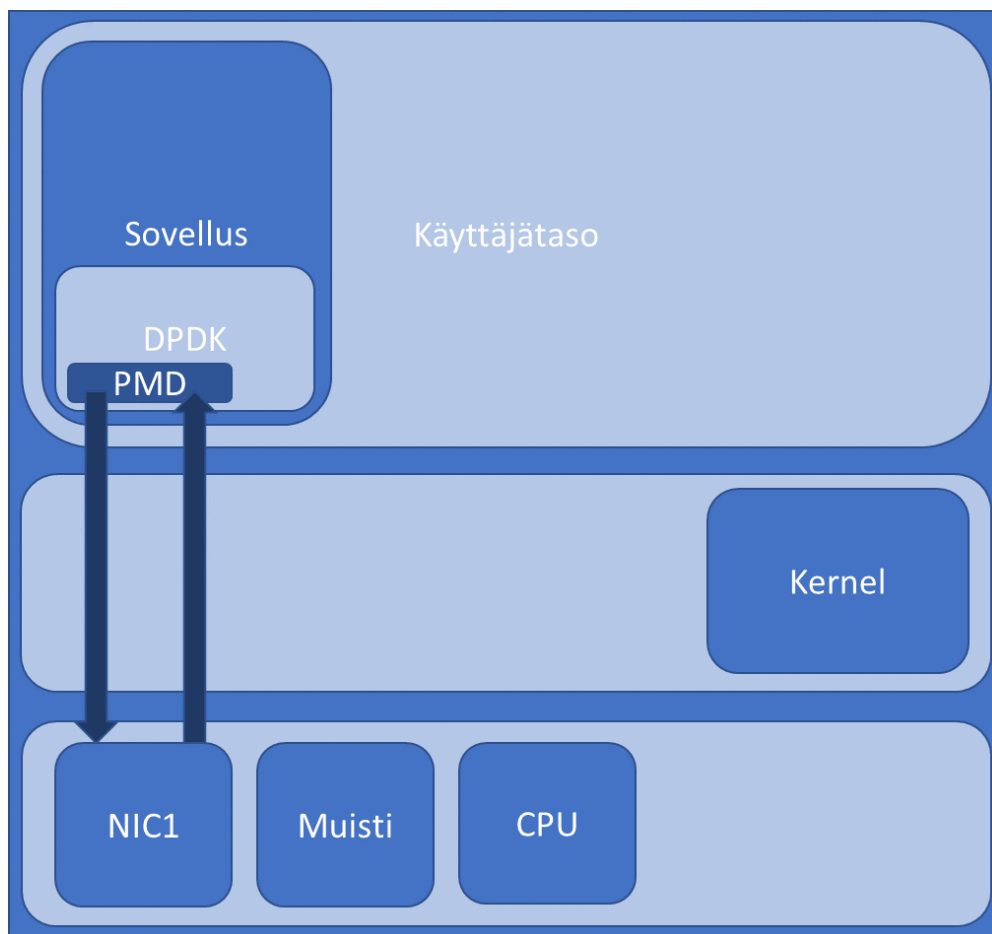
Näihin ongelmiin auttaa DPDK:n PMD:t eli Poll Mode Driverit. Nämä ajurit ovat eri DPDK-projektin kanssa yhteistyössä toimivien verkkokorttivalmistajien kehittämiä ajureita, jotka toimivat käyttäjätasossa DPDK:n ja verkkokortin välissä. Näitä ajureita on esimerkiksi Mellanoxin, Ciscon ja Intelin verkkokorteille (DPDK supported, n.d.).

DPDK:sta löytyy PMD myös Virtiolle joka on standardi rajapinta jonka kautta virtuaalikoneessa toimiva paravirtualisoitu käyttöjärjestelmä on yhteydessä emuloi-tuun laitteeseen. Kyseessä oleva käyttöjärjestelmä tietää olevansa virtualisoitu ja tarvitsee ajureita toimintaansa, mutta tämä tuo etuja verrattuna kokonaan virtu-alisoidun käyttöjärjestelmän ja emuloidun laitteen väliseen kanssakäymiseen ver-rattuna. (IBM Virtio 2010.)

Nämä PMD:t toimivat käyttäjäavaruudessa tarjoten pääsyn verkkokortin rengas-puskuriin ilman keskeytyksiä, eli nimensä mukaisesti paketteja kysellään jatku-vasti verkkokortilta. Tämä onnistuu siten että DPDK:ssa on mahdollista asettaa looginen ydin (eli säie, pthread) suorittamaan yhtä prosessia kuten pollaamaan PMD:n rajapintaa saapuneiden pakettien toivossa. DPDK tarjoaa tähän kaksi ta-paa, jotka ovat run-to-completion ja pipeline-malli. Run-to-completion mallissa tehtävänsä värvätty ydin kyselee saapuneita viestejä, prosessoi ne, ja antaa ne portille lähetettäväksi. Pipeline mallissa yksi ydin voi olla se, joka kyselee paket-teja ja toimittaa ne seuraavalle ytimelle joka vuorostaan prosessoi paketit ja antaa ne portille lähetettäväksi. (DPDK PMD n.d.)

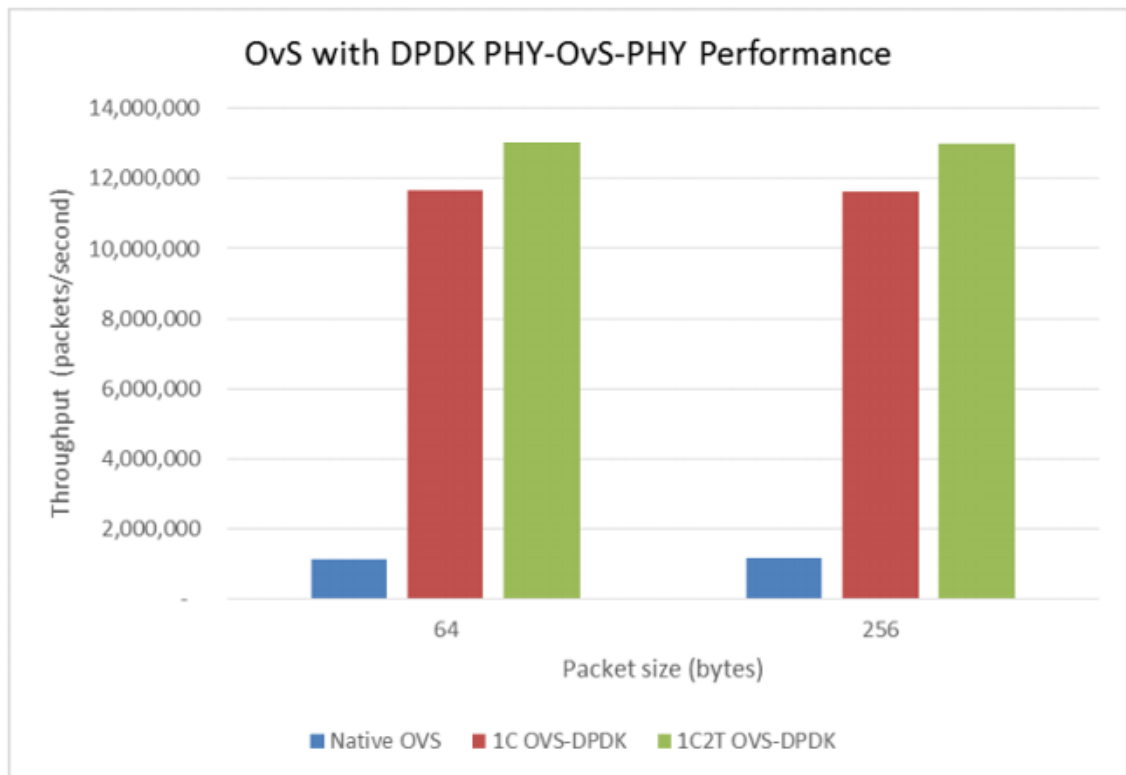
#### **4.7 DPDK:n suorituskyky**

DPDK ohittaa Linux-ytimen ja luo suoraan yhteyden verkkokortin ja käyttäjätason välille (kuvio 9). Näin vastaanotettavat paketit voidaan tuoda sellaisenaan käyt-täjäavaruuteen ja esimerkiksi jatkaa niiden prosessointia tehtävään määritellyillä prosessoriytimillä. Tätä toimintakyvyn eroa DPDK:n ja tavallisen Linux-ytimen verkkopinon välillä on testattu, ja antaa hyvän vaikutelman DPDK:n suoritusky-vystä.



KUVIO 9. DPDK:n asema laitteistossa

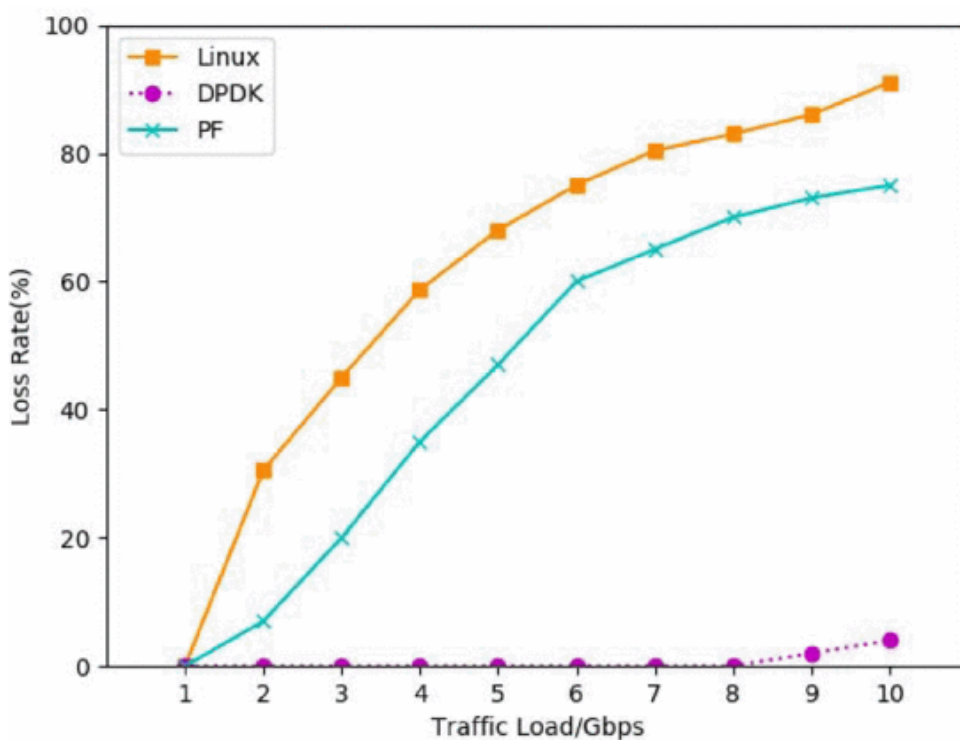
DPDK:n tuomaa hyötyä testattiin Intelin toimesta (Giller, 2016) siten, että siirrettiin 64 ja 256 tavun paketteja OvS:n natiivin datapolun kautta, sekä OvS:n DPDK-datapolun kautta ensin yhdellä ytimellä ja tämän jälkeen yhdellä ytimellä mutta kahta säiettä käyttäen (kuvio 10).



KUVIO 10. OvS:n suorituskky natiivina ja DPDK:n kanssa (Giller 2016)

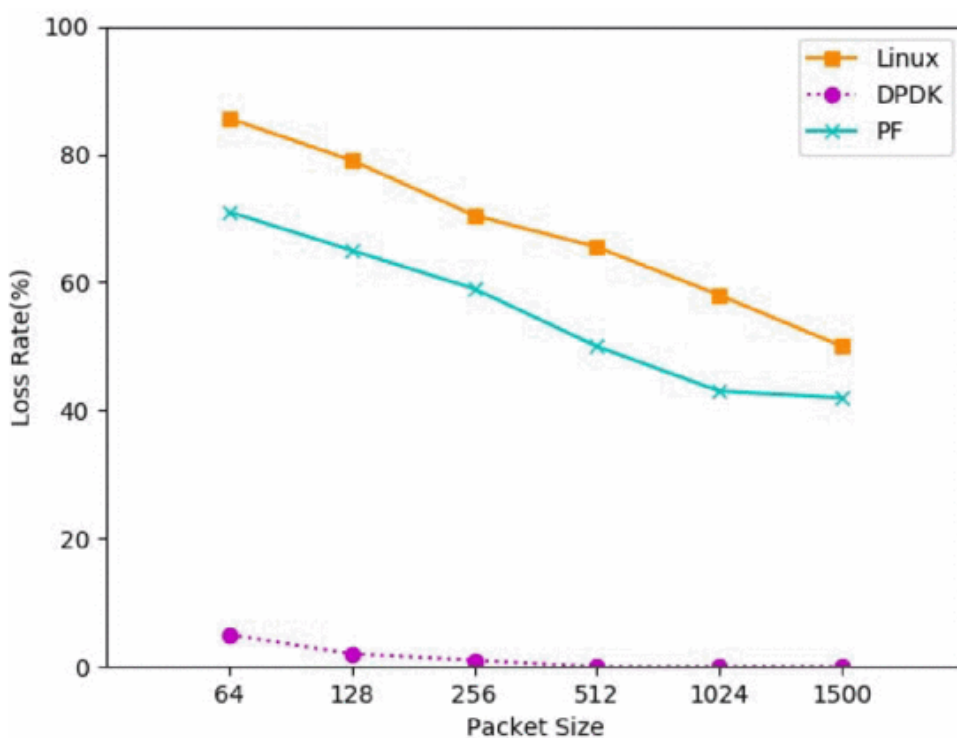
DPDK-datapolkua käyttäen paketteja meni sekunnissa noin 10 kertainen määrä yhdellä ytimellä ja noin 12 kertainen määrä yhdellä ytimellä ja kahdella säikeellä, kun verrattiin tuohon Open vSwitchin natiiviin datapolkuun. (Giller 2016.)

Wenjun (2018, 5) ryhmineen tutki artikkelissaan DPDK:n suorituskkyä. He suorittivat testejä, joissa mitattiin DPDK:n kykyä kaapata paketteja lähettämällä eri kokoisia paketteja eri nopeuksilla, ja mittaamalla kuinka paljon paketteja menetetttiin. DPDK:ta verrattiin Linux-ytimen verkkopinon ja PF\_RING:in suorituskkyyn. Ensimmäisessä testissä testattiin vakiokokoisilla paketeilla alkaen nopeudesta 1 gigabittiä sekunnissa ja päättyen 10 gigabittiin sekunnissa (kuvio 11).



KUVIO 11. Pakettien menetys eri nopeuksilla (Wenjun, 2018)

Toisessa testissä mitattiin pakettien menetyksen määrää muuttamalla pakettien kokoa (kuvio 12). DPDK selviytyi tästäkin testistä selvästi pelkkää Linuxia ja PF\_RING:iä paremmin.



KUVIO 12. Pakettien menetys eri kokoisilla paketeilla (Wenjun, 2018)

Näissä esitellyissä testeissä ei testata sen laajempaa systeemiä syvällisemmin mutta ne antavat hyvän kuvan DPDK:n kyvyistä verrattuna joihinkin muihin ratkaisuihin ja näin ollen tarjoavat syyn siihen, miksi DPDK:ta kannattaa käyttää sovelluksissa, joissa tarvitaan nopeaa pakettiprosessointia.

DPDK:n tarve kiteytettynä siis syntyy siitä, että kun verkkotoimintoja on ruvettu virtualisoimaan, ei tavallisten käyttöjärjestelmien tapa käsitellä verkkoliikenteen paketteja enää ole ollut riittävä verkkokorttien kasvaneiden nopeuksien vuoksi. DPDK ohittaa käyttöjärjestelmissä tapahtuvan pakettikäsittelyn ja tuo paketit verkosta muokkaamattomana sovellustasolle käsiteltäväksi.

## 5 Tutkittu käyttäjäsovellus

### 5.1 Käyttäjäsovelluksen muistinhallinta

Tutustutaan vielä tarkemmin aiemmin mainittuun kerrokseen käyttäjäsovelluksen ja DPDK:n rajapinnan välissä (kuvio 13). Tässä työssä tutkitaan valmiin sovelluksen muistinkäyttöä.



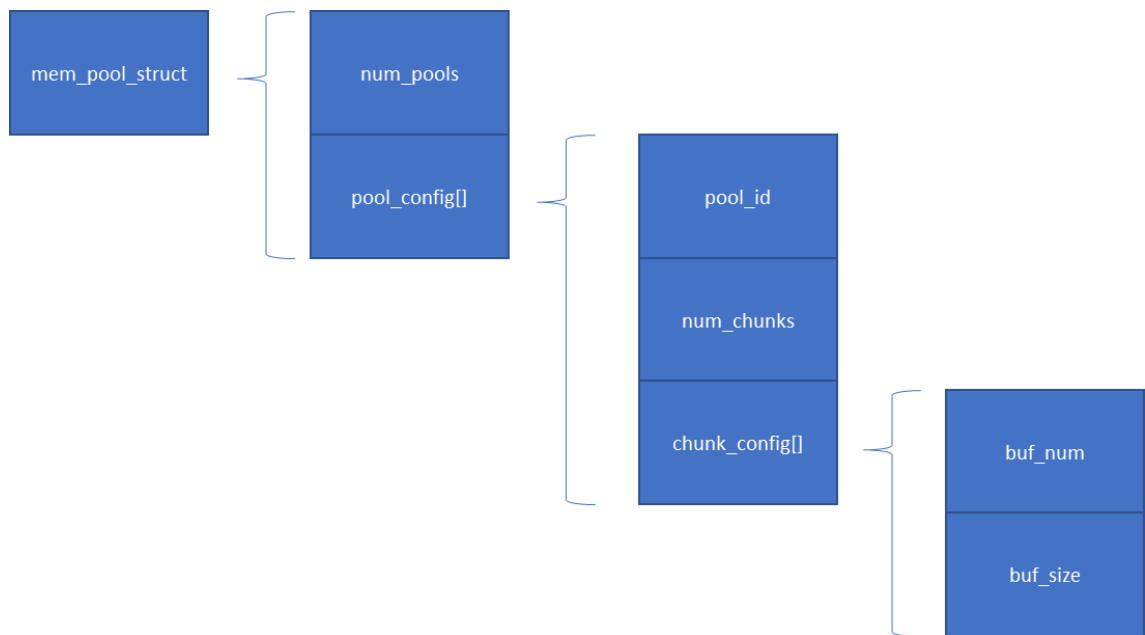
KUVIO 13. Havainnollistava kuva sovelluksen rakenteesta

Tämä DPDK:n ja sovelluksen välissä oleva sovituserros sisältää funktioita muistinhallintaan, ajastuksiin, ja muihin DPDK:n tarjoamiin ominaisuuksiin.

Muistia tutkitussa sovelluksessa käytetään siten, että sovellusta käynnistettäessä haluttu käytettävä muisti konfiguroidaan. Tämä konfigurointi tapahtuu siten, että muistialtaiden haluttu määrä määritellään ohjelmakoodissa ja sitten jaetaan tunnisteen perusteella eri tarkoituksiin kuten yleiseen käyttöön, ajastimille, prosessien väliselle viestinnälle sekä prosessien omaa muistia varten.

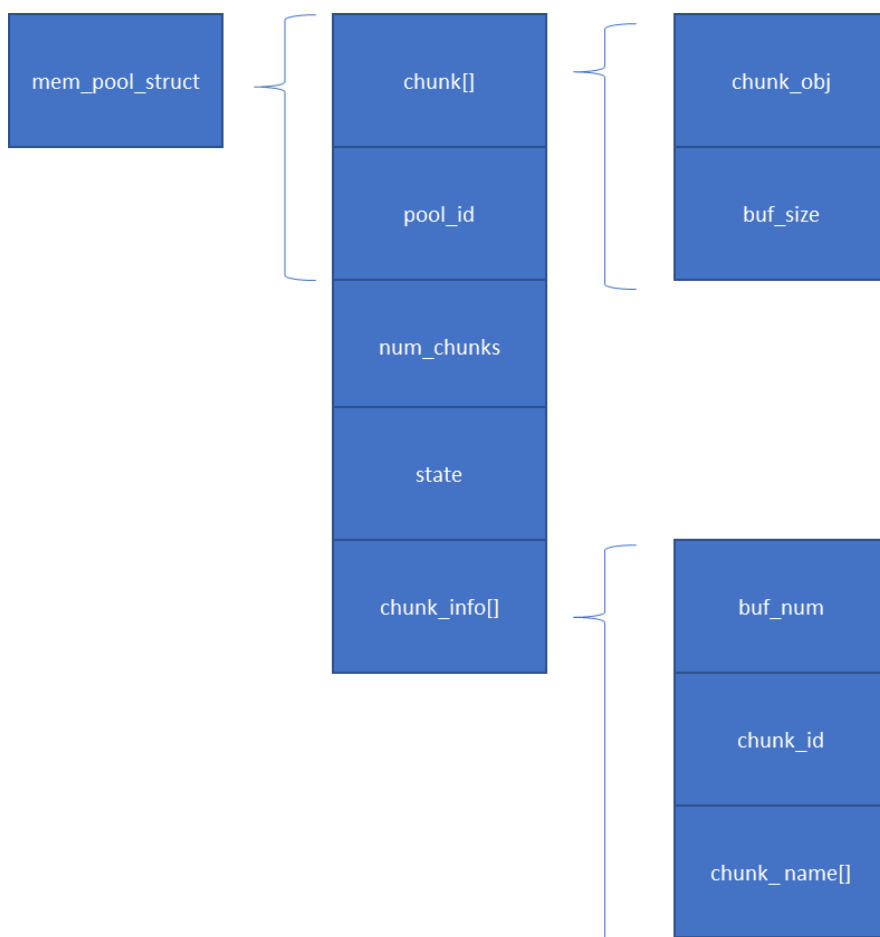
Sovelluksen käynnistyessä tehdään alustukset, ja tämä tehdään myös muistille. Muistille on määritelty struktuuri (kuvio 14) joka sisältää haluttujen muistialtaiden

määrän, altainen konfiguroinnit eli pool\_configin. Pool\_config sisältää pool\_id:n, halutun lohkojen (chunk) määrän, sekä chunk\_configin. Chunk\_config sisältää puskurien määrän ja puskurien koot. Esimerkiksi chunk\_config[0] alla voidaan antaa puskureiden määräksi vaikka 64 ja puskurikooksi 128.



KUVIO 14. Muistialtaiden konfigurointiin tarkoitettu rakenne

Aluksi alustettu konfiguraatietieto toimitetaan eteenpäin käsiteltäväksi muistia varaavalle toiminnolle joka muistialtaita varatessaan asettaa tiedot erilliseen muistialtaiden tiedot sisältävään rakenteeseen (kuvio 15).



KUVIO 15. Rakenne, johon varatun muistin osoitteet ja tiedot asetetaan

Tämä varatun muistin tiedot sisältävä rakenne saa sisältönsä muistia varaavista funktioista, joiden toiminta käydään läpi seuraavaksi.

## 5.2 Muistin varaus

Systeemin käynnistämisen jälkeen aloitetaan eri osa-alueiden alustus, ja tässä keskitytään yhteen niistä eli muistin alustamiseen (kuvio 16).



KUVIO 16. Muistin varaus

Kun sovellus käynnistetään, muiden alustusten joukossa alustetaan myös muistin. Tämä alkaa sillä, että kutsutaan funktiota `bf_init`. `Bf_init`in alussa kutsutaan funktiota `bf_set_mem_config` joka alustaa aiemmin kuvatun konfigurointirakenteen sillä tavalla, miten on lähdekoodiin määritelty. Kun konfiguraatio on valmis, siirrytään `create_mem`-funktioon, joka saa parametrinaan konfigurointirakenteen. Funktiossa `create_mem` kutsutaan funktiota `alloc_name_block` joka taas puolestaan kutsuu DPDK:n funktiota `rte_memzone_lookup`. `Rte_memzone_lookup` etsii sille parametrina annetun nimen perusteella, että löytyykö sillä nimellä jo valmiiksi varattua muistialuetta, ja jos löytyy, palauttaa sen osoittimen tähän muistialueeseen (DPDK API `memzone`, n.d.)

Jos nimeä vastaavaa aluetta ei löydy, edetään funktioon `rte_memzone_reserve_aligned`. Se varaa muistialueen funktiolle annetun nimen alle sen pituisena kuin määritelty ja palauttaa osoittimen tähän muistialueeseen. Funktio al-

`loc_name_block` palauttaa muistialueen sinne mistä sitä kutsuttiin ja saatu muistialueen osoite asetetaan aiemmin kuvailtuun rakenteeseen (kuvio 15) johon talletetaan konfiguraation perusteella varatun muistin tiedot.

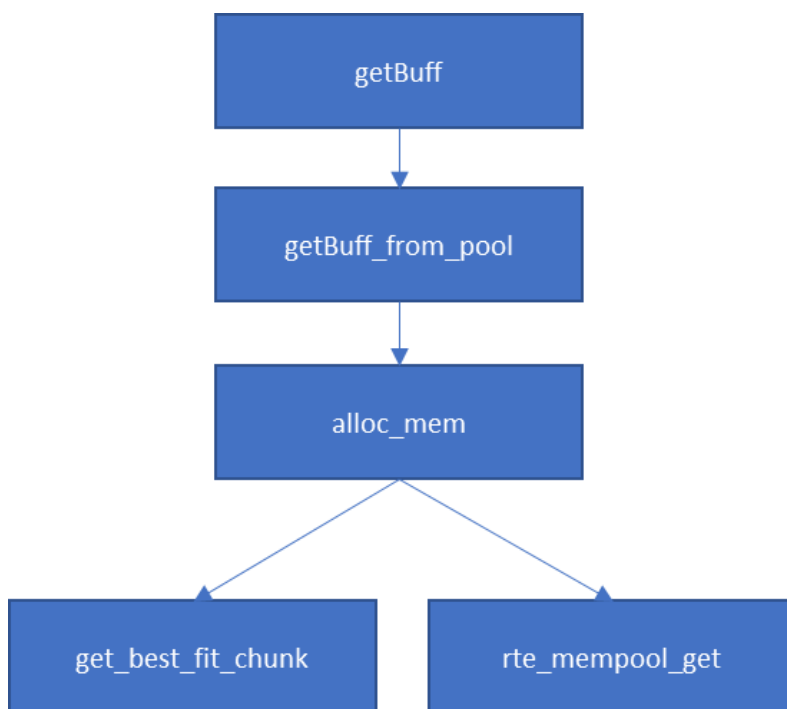
Seuraavaksi funktiossa `create_mem` kutsutaan funktiota `create_pool` niin monta kertaa kuin on konfigurointirakenteessa muistialtaiden määräksi `num_pools` muuttujassa määritetty.

`Create_pool` funktiossa kutsutaan funktiota `sort_chunk_by_size` joka lajittelee lohkot kokonsa perusteella. Lajittelun jälkeen käytetään `create_chunk`-funktiota. `Create_chunk` kutsuu edelleen `mempool_create`-funktiota, joka taas kutsuu DPDK:n funktiota `rte_mempool_create`, ja saa parametreinaan muun muassa muistialtaalle annettavan nimen, haluttujen puskureiden määrän, sekä yksittäisen puskurin koon. `Rte_mempool_create` palauttaa tämän varatun muistialtaan osoittimen, joka asetetaan konfiguraation perusteella varatun muistin tiedot sisältävään rakenteeseen. Lopuksi vielä tarkastetaan muistialtaassa olevien elementtien määrä funktiolla `rte_mempool_in_use_count`. Sen palauttama tieto vapaiden elementtien määrästä sijoitetaan muistitiedot sisältävään rakenteeseen.

Sovelluksen käynnistyessä aloitetaan eri osien alustus, ja tässä selostetaan yhden osan, muistin, alustus. Aluksi ennalta määritetty varattavan muistin rakenne ja määrä asetetaan kuvion 14 mukaiseen konfiguraatorakenteeseen. Tämän jälkeen konfiguroinnin perusteella muisti varataan ja tieto varatusta muistista asetetaan sille tiedolle tarkoitettuun kuvion 15 mukaiseen rakenteeseen.

### 5.3 Varatun muistin käyttöönotto

Muistia varattiin eri asioille, eli yleiseen käyttöön, ajastuskäyttöön, prosessien välisen viestinnän tarpeisiin sekä prosessien tarpeisiin. Tätä varattua muistia haetaan tarpeen mukaan käyttöön näitä tarkoituksia varten ja se tapahtuu kuvion 17 mukaisella tavalla.



KUVIO 17. Muistin käyttöönotto

Muistin hakeminen alkaa getBuff-funktiosta, jota jokin sovellus kutsuu. GetBuff on tarkoitettu yleisen muistin varaamiseen, ja toimittaa parametrit eteenpäin funktiolle getbuff\_from\_pool. Tämä funktio saa parametreinaan getBuff-funktion parametrit ja sen lisäksi tiedon siitä, että haettava muisti otetaan yleisestä muistialtaasta.

Funktion alloc\_mem tarkoitus on saamiensa parametrien perusteella etsiä sopivan kokoinen lohko (chunk). Näitä lohkoja varattiin vaihteleva määrä eri kokoisia, jotta voitaisiin mahdollisimman tarkasti jakaa muistia sovelluksen eri osille. Tämä etsintä tapahtuu funktiolla get\_best\_fit\_chunk joka hakee sopivan kokoisen lohkon ID:n pool\_id:n sisältämien chunk ja num\_chunk sekä halutun koon perusteella. Funktiossa get\_best\_fit\_chunk käytetään DPDK:n funktioita rte\_prefetch1 ja rte\_prefetch2.

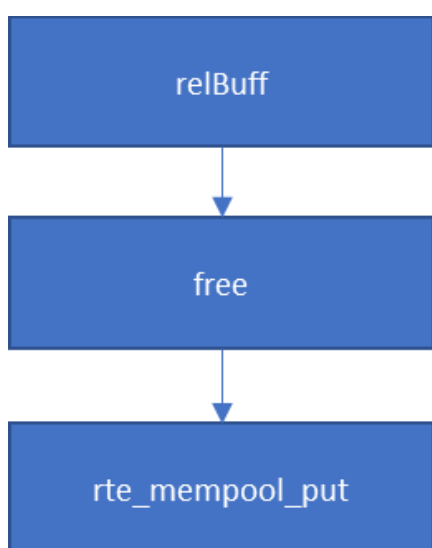
Rte\_prefetch-funktiot hakevat datan muistista prosessorin välimuistiin ennen kuin sitä oikeasti tarvitaan ja tällä on mahdollista saavuttaa hyötyjä nopeudessa, kun data on jo valmiina sitä tarvittaessa (DPDK API reference prefetch, n.d.)

Edelleen ollaan `alloc_mem`-funktiossa ja tiedetään `chunk_id`. Tämän jälkeen `chunk_id`:tä verrataan pool-struktista saatuun chunkien määrään ja käyttäen `rte_mempool_get`-funktia DPDK:n kirjastoista, haetaan osoittimeen haetun lohkon muistiosoite.

`Rte_mempool_get`-funktio saa parametreinaan osoittimen, joka osoittaa muistialtaaseen, ja osoittimen, johon asetetaan täytettävän objektin osoite (DPDK API reference mempool, n.d.).

## 5.4 Muistin vapautus

Käyttöön otettu muisti voidaan myös vapauttaa. Se tehdään siten että funktiota `relBuff` kutsutaan siitä prosessista, josta varattu muistipuskuri vapautetaan. Funktio `relBuff` kutsuu funktiota `free` joka puolestaan kutsuu DPDK:n funktiota `rte_mempool_put` (kuvio 18).



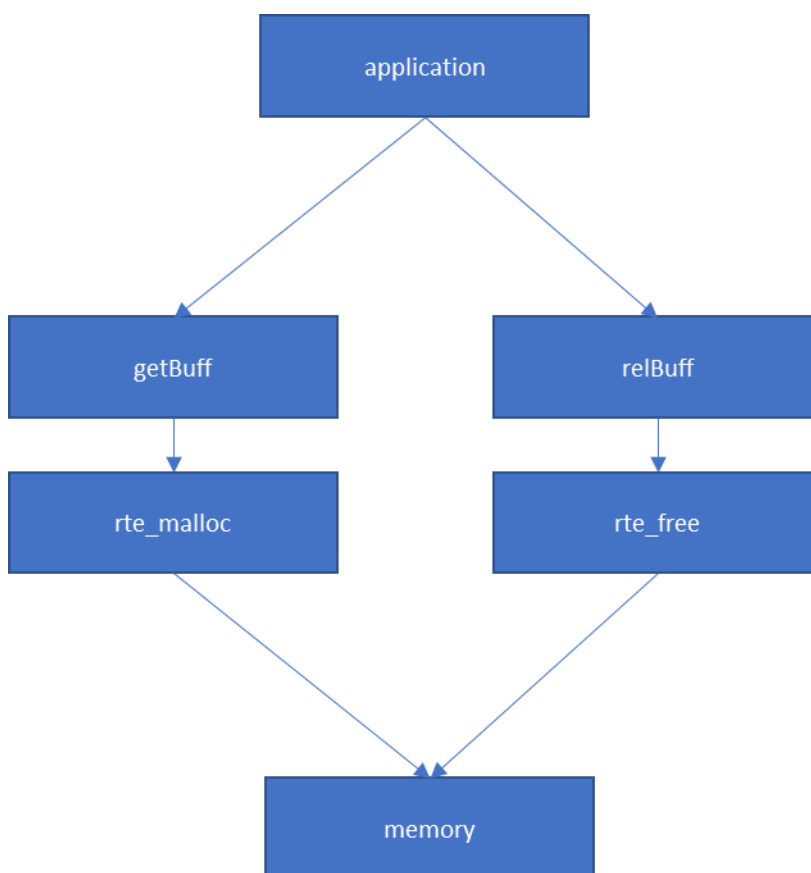
KUVIO 18. Muistin vapautus

`Rte_mempool_put`-funktia käytetään muistialtaan objektien palauttamiseen muistialtaaseensa. Sen parametrit ovat halutun muistialtaan osoite sekä palautettavan objektin osoite. (DPDK API reference mempool n.d.)

## 5.5 Muutokset muistin toimintaan

Kappaleessa 4.5.3 todettiin että DPDK:n muistinkäyttö on muuttunut päivitysten 17.11 ja 18.11 välillä dynaamisemmaksi ja tätä koetettiin testata luvussa 5 esitellyn valmiin sovelluksen puitteissa. Aiemmin mainitut funktiot `rte_memzone_reserve_aligned` ja `mempool_create_aligned` ovat sellaisia, että niitä voidaan dynaamisten toimintojen ansiosta käyttää DPDK:n alustamisen jälkeenkin.

Huomattiin kuitenkin, että konfigurointirakenteen perusteella varatuista muisteista, vain prosesseille sisäiseksi dataksi varattava osa oli sellainen, joka kannatti varata dynaamisesti ja tämä tehtiin käyttäen DPDK:n `rte_malloc`-funktia. Tämä sen vuoksi että sovelluksen käynnistyessä prosessin sisäisen muisti varattiin vain kerran ja se säilyi koko prosessin elinkaaren ajan. Kun käytettiin funktiota `rte_malloc` niin piti myös luoda erillinen `relBuff`-funktio tätä prosessin muistin vapautusta varten (kuvio 19), sillä alkuperäinen `relBuff` oli sellainen funktio, joka oli tehty sovelluksen alussa konfiguraatorakenteen perusteella varatun muistin vapautusta varten.



KUVIO 19. Muistin käytön muutoksia prosessidatan osalta

Muita osia kuten yleiseen käyttöön tarkoitettuja muistialtaita käytettiin erilaisten viestien siirtoon, ja näissä tapauksissa todettiin, että ei ole järkevää varata muistia siten, että käytettäisiin suoraan esimerkiksi DPDK:n `mempool_create`-funktiota jokaisella kerralla, kun jokin viesti tarvitsee puskuria. Tämä johtuu siitä, että `rte_malloc`-funktion käyttäminen jokaisen yksittäisen viestin tarpeeseen ei ole kannattavaa suorituskyvyn vuoksi. Lisäksi ei ole kannattavaa varata kokonaista muistiallasta esimerkiksi yhtä elementtiä varten per jokainen viesti. Tämän vuoksi muistiallas olisi pitänyt varata viestittävän prosessin alussa valmiiksi viestien siirtoa varten, mutta tämä on lähes sama asia kuin olemassa oleva toiminto, jossa varattava muisti konfiguroidaan ja varataan koko järjestelmän käynnistymisen yhteydessä, ja tätä varattua muistia sitten haetaan ja allokoidaan tarpeellisen kokoisina paloina tätä viestien siirtoa varten.

Yhteenvetona voidaan todeta, että dynaamiselle muistin varaukselle löytyi käyttöä harvakseltaan tai kerran tapahtuville datan luonneille, mutta jatkuvasti, ja lähes satunnaisesti tapahtuvalle viestinnälle ei nähty järkeväksi tuoda tätä dynaamista muistin varausta lähelle prosessin muistin varausfunktioita.

## 6 POHDINTA

Tässä opinnäytetyössä tutustuttiin laajasti Data Plane Development Kit -ohjelmistokehykseen, sen kirjastoihin ja sen mukana tulleisiin Poll Mode -ajureihin. Lisäksi keskityttiin enemmän DPDK:n muistinkäyttöön ja tutkittiin DPDK:ta käyttävän tutkitun sovelluksen tapoja käyttää muistia ja miten se yhdistyy DPDK:n rajapintoihin.

Työtä tehdessä käsitys DPDK:n toiminnasta kasvoi huomattavasti sillä kirjoittaessa tuli tutustuttua DPDK:n dokumentaatioon, esimerkksiovelluksiin ja ohjelmointirajapintaan. Myös yleistieto liittyen verkkotoimintoihin kasvoi, sillä tutustuttiin myös virtualisointiin ja niiden toimintojen toteuttamisen historiaan. Saatiin myös parempi käsitys työpaikalla käytettävän sovelluksen toiminnasta ainakin muistin osalta.

Käyttäjäsovellusta tutkittaessa, todettiin ettei ainakaan nopeassa viestinnässä muistinkäsittelyä kannattanut muuttaa. Esimerkiksi yhden kerran varattavalle prosessin muistille dynaaminen varaus voi olla ihan järkevä vaihtoehto, mutta jatkuvasti liikkuvien viestien muistintarvetta ajatellen oli parempi, että muisti varattiin valmiiksi jo sovelluksen alussa ja sitä varattua muistia sitten otetaan tarvittaessa käyttöön. Nopeutta ajatellen ei kannattanut käyttää malloc-tyylistä varaamista, ja jos viestittävän prosessin alussa olisi varattu muistiallas viestintää varten, olisi siinäkin tapauksessa tarvinnut tietää kuinka paljon muistia varata. Tämä olisi kuitenkin lähes sama tapa, miten muistinvaraus sovelluksessa tapahtuu jo nyt. Työssä kuvaillut mbuf-puskurit olisivat myös olleet mielenkiintoinen mahdollisuus esimerkiksi viestien kuljetukseen sovelluksessa, mutta ne eivät olleen tässä työssä tarkemmin tutkittava kohde.

Tämä opinnäytetyö toimi hyvänä porttina tutustuttaessa DPDK:n tarpeeseen ja ominaisuuksiin. Työ lähtee laajasta selostuksesta suppenemaan kohti tarkempia yksityiskohtia lopulta päätyen jo mainitun käyttäjäsovelluksen muistinhallintaan ja ajatuksiin sen kehittämisestä. Näin ollen työ onnistui hyvin.

Työn ansiosta ymmärrys verkkotoimintojen historiasta, DPDK:sta ja sen tarpeesta sekä yleisesti tietokoneista ja järjestelmistä kasvoi. Työt DPDK:n parissa jatkuvat koska DPDK on laaja kokonaisuus, josta tässä työssä tutkittiin vain muistia.

## LÄHTEET

Burakov, A. 2019. Memory in Data Plane Development Kit Part 3: 17.11 and Earlier Releases. Luettu 22.3.2021. <https://software.intel.com/content/www/us/en/develop/articles/memory-in-dpdk-part-3-1711-and-earlier-releases.html>

Burakov, A. 2019. Memory in Data Plane Development Kit Part 4: 18.11 and Beyond. Luettu 22.3.2021. <https://software.intel.com/content/www/us/en/develop/articles/memory-in-dpdk-part-4-1811-and-beyond.html>

Cerovic, D. 2018. Fast Packet Processing: Survey. Luettu 19.4.2021. Vaatii käyttöoikeuden. <https://ieeexplore.ieee.org/document/8398225>

Doherty, J. 2016, SDN and NFV Simplified. Indiana: Pearson Education.

DPDK about, n.d. DPDK.org Luettu 19.4.2021. <https://www.dpdk.org/about/>

DPDK API reference mbuf, n.d. DPDK.org. Luettu 11.4.2021. [https://doc.dpdk.org/api/rte\\_mbuf\\_8h.html](https://doc.dpdk.org/api/rte_mbuf_8h.html)

DPDK API reference memzone. n.d. DPDK.org. Luettu 14.4.2021. [https://doc.dpdk.org/api/rte\\_memzone\\_8h.html#aa772ef84e73ac173503c16f243766993](https://doc.dpdk.org/api/rte_memzone_8h.html#aa772ef84e73ac173503c16f243766993)

DPDK API reference mempool. n.d. DPDK.org. Luettu 23.3.2021. [https://doc.dpdk.org/api/rte\\_mempool\\_8h.html](https://doc.dpdk.org/api/rte_mempool_8h.html)

DPDK API reference prefetch. n.d. DPDK.org. Luettu 21.4.2021. [https://doc.dpdk.org/api/rte\\_prefetch\\_8h.html](https://doc.dpdk.org/api/rte_prefetch_8h.html)

DPDK blog release 20.11, 2020, DPDK.org Luettu 14.4.2021. <https://www.dpdk.org/blog/2020/11/30/dpdk-issues-20-11-most-robust-dpdk-release-ever/>

DPDK EAL. n.d. DPDK.org. Luettu 15.3.2021. [http://doc.dpdk.org/guides/prog\\_guide/env\\_abstraction\\_layer.html](http://doc.dpdk.org/guides/prog_guide/env_abstraction_layer.html)

DPDK mempool. n.d. DPDK.org. Luettu 20.3.2021. [http://doc.dpdk.org/guides/prog\\_guide/mempool\\_lib.html#mempool-library](http://doc.dpdk.org/guides/prog_guide/mempool_lib.html#mempool-library)

DPDK mbuf. n.d. DPDK.org. Luettu 10.4.2021. [https://doc.dpdk.org/guides/prog\\_guide/mbuf\\_lib.html](https://doc.dpdk.org/guides/prog_guide/mbuf_lib.html)

DPDK Overview, n.d. DPDK.org. Luettu 17.2.2021. [https://doc.dpdk.org/guides/prog\\_guide/overview.html](https://doc.dpdk.org/guides/prog_guide/overview.html)

DPDK PMD. n.d. DPDK.org. Luettu 13.3.2021. [https://doc.dpdk.org/guides/prog\\_guide/poll\\_mode\\_drv.html](https://doc.dpdk.org/guides/prog_guide/poll_mode_drv.html)

DPDK ring. n.d. DPDK.org. Luettu 20.3.2021.  
[https://doc.dpdk.org/guides/prog\\_guide/ring\\_lib.html](https://doc.dpdk.org/guides/prog_guide/ring_lib.html)

DPDK supported. n.d. DPDK.org. Luettu 30.4.2021. <http://core.dpdk.org/supported/>

DPDK system requirements. n.d. DPDK.org. Luettu 20.3.2021.  
[https://doc.dpdk.org/guides/linux\\_gsg/sys\\_reqs.html](https://doc.dpdk.org/guides/linux_gsg/sys_reqs.html)

DPDK Timer. n.d. DPDK.org. Luettu 11.4.2021.  
[https://doc.dpdk.org/guides/prog\\_guide/timer\\_lib.html](https://doc.dpdk.org/guides/prog_guide/timer_lib.html)

Emmerich, P., Gallenmuller, S., Raumer, D., Wohlfart, F., Carle, G. 2015. MoonGen: A Scriptable High-Speed Packet Generator. Luettu 12.4.2021.  
[https://www.net.in.tum.de/fileadmin/bibtex/publications/papers/MoonGen\\_IMC2015.pdf](https://www.net.in.tum.de/fileadmin/bibtex/publications/papers/MoonGen_IMC2015.pdf)

Giller, R. 2016. Open vSwitch\* with DPDK Overview kuva. Luettu 5.4.2021.  
<https://software.intel.com/content/www/us/en/develop/articles/open-vswitch-with-dpdk-overview.html>

Giller, R. 2016. Open vSwitch\* with DPDK Overview. Luettu 28.3.2021.  
<https://software.intel.com/content/www/us/en/develop/articles/open-vswitch-with-dpdk-overview.html>

Heqing, Z. 2021. Data Plane Development Kit (DPDK). Florida: CRC Press

Herbert, T., Bruijn, W. n.d. Scaling in the Linux Networking Stack. Kernel.org. Luettu 20.4.2021. <https://www.kernel.org/doc/Documentation/networking/scaling.txt>

IBM Virtio. 2010. IBM. Luettu 13.3.2021. <https://developer.ibm.com/technologies/linux/articles/l-virtio/>

IBM Jumboframes. n.d. IBM. Luettu 21.4.2021. <https://www.ibm.com/docs/en/linux-on-systems?topic=io-jumbo-frames>

MoonGen Github. Kuva MoonGenin toiminnasta. Luettu 12.4.2021. <https://github.com/emmericp/MoonGen>

Muistin toimintaa. 2020. Geeksforgeeks. Luettu 20.4.2021.  
<https://www.geeksforgeeks.org/whats-difference-between-cpu-cache-and-tlb/>

OpenDataPlane FAQ. n.d. ODP. Luettu 13.4.2021. <https://opendataplane.org/index.php/resources/faq/>

Overview of Packet Reception. n.d. Red Hat. Luettu 13.3.2021. [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/6/html/performance\\_tuning\\_guide/s-network-packet-reception](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/s-network-packet-reception)

PF\_RING documentation. n.d. Ntop. Luettu 20.4.2021.  
[https://www.ntop.org/guides/pf\\_ring/index.html](https://www.ntop.org/guides/pf_ring/index.html)

Rizzo, L. 2012. netmap: a novel framework for fast packet I/O. Luettu 20.4.2021. <https://www.usenix.org/system/files/conference/atc12/atc12-final186.pdf>

Wenjun, Z., Peng, L., Baozhou, L., He, X., Yujie, Z. 2018. Research and Implementation of High Performance Traffic Processing Based on Intel DPDK. Luettu 20.4.2021. <https://ieeexplore.ieee.org/document/8701793>

Ying, Z. 2018. Network Function Virtualization. New Jersey: John Wiley & Sons. <https://learning.oreilly.com/library/view/network-function-virtualization/9781119390602/c01.xhtml>