Tampereen ammattikorkeakoulu

TCP:n toteutussuunnitelma

TCP-protokolla korkean suorituskyvyn sovelluksissa

Miika Heiska

OPINNÄYTETYÖ
Toukokuu 2021

Tieto- ja viestintätekniikan tutkinto-ohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikan tutkinto-ohjelma
Ohjelmistotekniikka

HEISKA, MIIKA:
TCP:n toteutussuunnitelma
TCP korkean suorituskyvyn sovelluksissa

Opinnäytetyö 68 sivua, joista liitteitä 0 sivua
Toukokuu 2022

Opinnäytetyön tavoitteena oli suunnitella TCP-protokollan toteutus C-kielellä, käyttäen hyväksi DPDK-sovelluskehyksen palveluita. Työssä käsiteltiin yksityiskohtaisesti TCP-protokollan toiminta ja esiteltiin siihen liittyvää protokollapinon käsitettä. Työssä esiteltiin myös lyhyesti DPDK:ta, mutta se ei ollut työn keskeinen sisältö.

Työ syntyi toimeksiantajayrityksen, Nokia Networks and Solutions, aloitteesta, sillä Nokian olemassa olevan testijärjestelmän arkkitehtuuriin sopivaa TCP-toteutusta ei löytynyt.

TCP-protokolla on tietoliikenneprotokolla, joka muodostaa yhteyden kahden eri järjestelmässä pyörivän sovelluksen välille. TCP:n keskeisiä ominaisuuksia ovat muodostetun yhteyden yhteydellisyys ja pyrkimys takaamaan, että lähetetyt bitit saapuvat perille oikeassa järjestyksessä. Lisäksi TCP:n vastuihin kuuluvat vuonhallinta ja virheenkorjaus.

Työssä tarkasteltiin mekanismeja, joilla TCP toteuttaa yllä mainitut tavoitteensa, ja suunniteltiin proseduureja näiden mekanismien toteuttamiseen ohjelmallisesti. Käsiteltäviä seikkoja olivat muun muassa yhteyden muodostus ja purku, viestin lähetys ja vastaanottaminen, sekä kuittaus- ja vuonhallintamekanismit. Suunnitelmien käytännön toteutus oli opinnäytetyön aiherajauksen ulkopuolella ja jätettiin kirjoittajan itsensä toteutettavaksi osana työtehtäviään.

Asiasanat: tcp, transmission control protocol, tiedonsiirtoprotokollat

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT engineering
Software engineering

Miika Heiska:
Plan for TCP implementation
TCP in high performance applications

Bachelor's thesis 68 pages, appendices 0 pages
May 2022

The purpose of this thesis was to plan a C based implementation of Transmission Control Protocol, (TCP), using the services provided by the Data Plane Development Kit (DPDK) framework. The thesis contains a detailed description of the functionality of TCP and planned functions to realise a portion of this functionality. Due to time constraints the implementation of these functions into real, working C code could not be achieved, but the internal logic of these planned functions is documented by the means of flowcharts and pseudocode listings. An overview of concepts such as the protocol stack is provided, as well as a brief overview of DPDK.

The objective of this thesis was provided by Nokia Networks and Solutions. The software module designed here will become a part of a testing and validation framework of System-on-Chip devices, and as the architecture of this framework is specific, none of the many available TCP implementations could be used.

The documentation provided by this thesis will act as a design document of the future C implementation of TCP, that the author will create as part of his duties at the company that commissioned this work.

Key words: tcp, transmission control protocol, communication protocols

SISÄLLYS

ERITYISSANASTO tai LYHENTEET JA TERMIT (valitse jompikumpi)

TCP	Transmission Control Protocol. Yhteysprotokolla, joka takaa lähetetyn datan saapumisen ehjänä ja oikeassa järjestyksessä vastaanottajalle niin hyvin kuin vain mahdollista.
IP	Internet Protocol. Protokolla, joka on vastuussa eri verkoissa sijaitsevien kohteiden välisestä liikenteestä
IDE	Integrated Development Environment. Ohjelma tai joukko ohjelmia, joilla ohjelmistokehitystä suoritetaan. Sisältää muun muassa tekstieditorin, debuggerin ja versionhallinnan
VNC	Virtual Network Computing. Mekanismi, joka mahdollistaa virtuaalikoneiden käytön ohjelmistoja kehitettäessä ja ohjelmia suoritettaessa.
TCP/IP-malli	Yksi protokollapinon tarkasteluun ja kuvaamiseen luotu malli
OSI-malli	Toinen protokollapinon tarkasteluun luotu malli
DPDK	Data Plane Development Kit. Verkkopakettien ja vastaavanlaisten sanomien nopeaan käsittelyyn suunniteltu ohjelmointirajapinta
PDU	Protocol Data Unit. Geneerinen nimitys tiedonsiirtoprotokollan käyttämälle perusyksikölle
Full duplex	Kaksisuuntainen yhteys. Lähettäminen ja vastaanottaminen voi tapahtua samanaikaisesti

Protokolla	Tietty kokoelma tarkkaan määriteltyjä sääntöjä ja viestimuotoja. Määrittelee ”säännöt”, joita kaksi järjestelmää käyttävät keskustellessaan keskenään
IANA	Internet Assigned Numbers Authority. USA:n liittovaltion perustama standardointijärjestö, joka mm. hallinnoi IP-osoitteita
IETF	Internet Engineering Task Force. Voittoa tavoittelematon avoimien standardien järjestö, joka kehittää ja edistää erilaisia internetin protokollia.

1 JOHDANTO

Tietojärjestelmien välinen kommunikointi on erittäin monimutkainen prosessi ja siinä on useita eri vaiheita. Jokaista yhteyttä vaativaa ohjelmaa ei kuitenkaan voi vaatia toteuttamaan omaa tapaansa selviytyä yhteyden muodostamisen haasteista. Onneksi ei tarvitsekaan, sillä TCP, eli Transmission Control Protocol -niminen protokolla tarjoaa sovelluksille kätevästi abstrahoidun tavan keskustela jossakin toisessa laitteessa tai tietojärjestelmässä olevan sovelluksen kanssa.

Tässä työssä tutustutaan perusteellisesti TCP-protokollan toimintaan ja muodostetaan suunnitelma kyseisen protokollan toteuttamiseksi C-kielellä. Työn toimeksiantajalla on tarve räätälöidylle TCP-moduulille, joka saadaan liitettyä osaksi toimeksiantajan olemassa olevaa korkean suorituskyvyn testausjärjestelmää. Kyseessä olevan järjestelmän keskeisessä osassa on DPDK (Data Plane Development Kit), mistä syystä myös tälle kirjastokokonaisuudelle on varattu oma osuutensa tässä opinnäytetyössä.

Kappaleet 2 ja 3 avaavat työnannon taustoja ja perustelevat räätälöidyn TCP-toteutuksen tarvetta tilanteessa, jossa valmiita TCP-toteutuksia on jo olemassa. Lisäksi avataan pintapuolisesti sitä järjestelmäkokonaisuutta, johon valmis toteutus on tarkoitus liittää. Tämän jälkeen kerrotaan lyhyesti toteutuksen kirjoittamiseen käytetystä kehitysympäristöstä.

Kappaleet 5 ja 6 luovat teoreettisen pohjan, jotta lukija ymmärtää ja tuntee tähän työhön liittyvät tiedonsiirron käsitteet ja konseptit. DPDK avataan myöskin pintapuolisesti, jotta lukijalle välittyy käsitys rajapinnan suomista mahdollisuuksista.

Kappale seitsemän käsittelee TCP-protokollaa nimenomaan protokollan tasolla. Kyseistä kappaletta voi pitää eräänlaisena vaatimusmäärittelynä, sillä se kuvaa, mitä toimivan TCP toteutuksen tulee tehdä. Kappale 8 yrittää osaltaan vastata näihin vaatimuksiin ja kuvata miten ne täytettäisiin C-kielellä ohjelmoitaessa.

2 TYÖN TARKOITUS, TOIMEKSIANTO, REUNAEHDOT JA MOTIIVIT

Tämä opinnäytetyö on tehty Nokia Networks and Solutions -yhtiön (tästä eteenpäin “Nokia”) toimeksiannosta. Toimeksiannon lopullinen tavoite on toteuttaa TCP-protokollan implementointi C-kielellä, käyttäen hyväksi DPDK-ohjelmointirajapinnan suomia ominaisuuksia. Tämän kirjallisen tuotoksen painopiste on dokumentoinnissa ja vaatimusmäärittelyssä, sillä käytännön toteutukseen ei riittänyt aikaa.

TCP on laajasti käytetty tiedonsiirtoprotokolla, jonka ominaisuudet ja vaatimukset ovat avoimesti saatavilla. Tästä johtuen valmiita TCP-toteutuksia on saatavilla Linux-käyttöjärjestelmiin rakennettujen toteutusten lisäksi myös avoimen lähdekoodin toteutuksena. Valitettavasti yksikään näistä toteutuksista ei sovi sellaisenaan Nokian kehittämään järjestelmään: Perinteinen TCP-toteutus käyttää käyttöjärjestelmän ytimen eli kernelin palveluita, ja tämä luonnostaan heikentää tehokkuutta. Aikaisemmin tämä ei ollut ongelma, mutta linkkien nopeutuessa perinteisen verkkopinon aiheuttamat hidastukset alkavat näkyä.

Linux käsittelee verkkopaketteja verrattain hitaasti, koska suorituksen täytyy siirtyä kernelin ja käyttäjäavaruuden (user space) välillä. Lisäksi Linuxin verkkopino suunniteltiin yhteensopivaksi mahdollisimman monen eri protokollan kanssa, joten pakettien käsittely sisältää paljon ylimääräistä metadataa. Tämä ylimääräinen data lisää laskennan kompleksisuutta ja täten pakettien käsittelyyn kuluvaa aikaa.

DPDK kiertää nämä ongelmat luomalla suoran kommunikaatioväylän verkkokortin ja käyttäjäavaruuden välille, joten ajallisesti kallista kontekstin vaihtoa ei tarvita. Lisäksi DPDK:ta hyödyntäen voidaan rakentaa protokollaspesifejä toteutuksia, jotka eivät välttämättä ole yhtä laajalti yhteensopivia kuin Linuxin verkkopino, mutta suoriutuvat oman protokollansa pakettien käsittelystä paljon nopeammin.

DPDK:n mukaan ottaminen ei yksinään tehnyt tätä työtä välttämättömäksi, mutta olemassa olevat DPDK:ta hyödyntävät TCP-toteutukset eivät sovi nykyiseen järjestelmäarkkitehtuuriin. Olemassa olevat TCP-toteutukset vaativat myös yhden

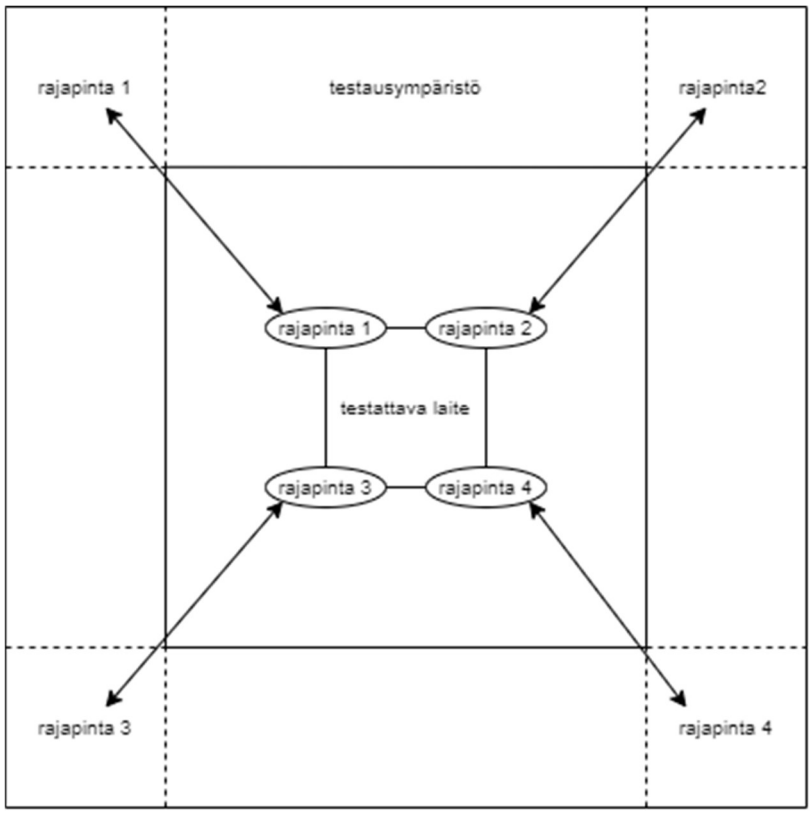
prosessointiytimen käyttämistä pelkästään TCP-yhteyksiä varten, mikä ei ole toivottavaa ytimien rajoitetun määrän vuoksi.

Toimeksiannon tuloksena tuotetun TCP-toteutuksen tarkoitus on toimia DPDK:n mahdollistamilla korkeammilla paketinkäsittelynopeuksilla ilman ylimääräistä ytimien tuhlausta. Lisäksi tämä toteutus on rakennettu varta vasten Nokian rakentaman testausympäristön kanssa yhteensopivaksi, sillä kyseiseen ympäristöön sopivaa TCP-toteutusta ei yksinkertaisesti ole olemassa.

3 TYÖ LAAJEMMASSA KOKONAISUUDESSA

Nokia on jo pidemmän aikaa rakentanut testialustaa, jonka tarkoitus on helpottaa SoC-laitteiden testaamista ennen niiden julkistamista markkinoille. Testaus ja validointi vie valtaosan SoC-laitteiden kehitysajasta, joten parempien testaus- ja validointityökalujen rakentaminen tuo mukanaan selkeää hyötyä. Kun laitteen testaus ja validointi helpottuu, voi siihen allokoituja resursseja siirtää kehitystyöhön tai vaihtoehtoisesti vähentää tuotteen “ideasta hyllylle”-kehityskaareen kuluvaa aikaa ilman, että tuotteen toimintavarmuus tai validoinnin kattavuus kärsii. Vaihtoehtoisesti samalla aikajänteellä on mahdollista testata kattavammin, minkä tuloksena – ainakin toivottavasti – olisi parempia tuotteita.

Nokialla rakennettavassa testiympäristössä pyritään mahdollistamaan kehitettävän laitteiston testaaminen missä kohdassa laitteiston elinkaarta tahansa. Toteutettuna tällainen järjestelmä mahdollistaisi virheiden havaitsemisen jo suunnitteluvaiheessa, ennen kallista ladontaprosessia. Lisäksi tällainen todenmukaiseksi rakennettu testiympäristö mahdollistaa jo aikaisemmin luotujen testitapausten siirtämisen tähän rakennettuun testiympäristöön ja toisaalta myös tässä testiympäristössä luotujen testien – tai oikeassa elämässä kohdattujen ongelmatapausten – siirtämisen muihin testiympäristöihin kohtalaisen vaivattomasti.



KUVA 1: Testiympäristön arkkitehtuuri abstraktilla tasolla

Testiympäristö on suunniteltu sillä periaatteella, että testattava laite – oli se sitten todenmukainen, fyysinen laite, pelkästään ohjelmistotasolla toteutettu virtualisointi tai jotain siitä väliltä – olisi testijärjestelmän keskiössä ja testijärjestelmä tarjoaa sille sen tarvitsemat – tai testitapauksen määritelleen henkilön haluamat – palvelut ja rajapinnat. Testattavan laitteiston näkökulmasta eroa testijärjestelmän tarjoamien palvelujen ja niiden todenmukaisten vastineiden välillä ei ole, ja tämän vuoksi laitteistoa on mahdollista testata valtavassa määrässä eri tilanteita ilman että testaajan tarvitsee huolehtia esimerkiksi siitä, noudattaako testidata täysin todellisen protokollapinon vaatimuksia. Modulaariseksi rakennetut protokollat mahdollistavat mielivaltaisten protokollapinojen rakentamisen tai toisaalta yksittäisen protokollatoteutuksen muokkaamisen muista eristyksissä testitapausta vastaavaksi.

Tämän työn kuvaileman ohjelmiston on tarkoitus asettua saumattomasti osaksi testausympäristössä käytettävien modulaaristen protokollien joukkoa. Modulaarisesti rakennetuilla protokollatoteutuksilla on mahdollista rakentaa mielivaltaisia protokollapinoja mitä tahansa testattavaa laitteistoa tai testitapausta varten. Jotta tämä vaatimus pystytään täyttämään, on TCP-protokollatoteutus rakennettava

alusta asti yhteensopivaksi jo rakennetun testausjärjestelmäarkkitehtuurin kanssa.

4 KEHITYSYMPÄRISTÖSTÄ

TCP-toteutusta kehitetään Nokian laitteilla. Windows-työkannettava on VNC yhteydellä kiinni Nokian serverillä pyörivässä Ubuntu 20.04 virtuaalikoneessa, jonka sisällä kehitystyö tapahtuu. Versionhallintaohjelma Subversion toimii sekä versionhallinta- että varmuuskopiointijärjestelmänä, jotta ajoittain epävakaan virtuaalikoneympäristön tuomia riskejä voidaan lieventää.

Varsinainen kehitys aloitettiin QtCreator-ohjelmalla, joka on alustariippumaton IDE. QtCreator otettiin aluksi käyttöön siksi, että kirjoittajalla oli aikaisempaa kokemusta ohjelman käytöstä. Alkuperäinen suunnitelma oli jatkaa QtCreatorin käyttöä projektin loppuun asti, mutta kun em. ohjelmalla kirjoitettua koodia integroitiin osaksi laajempaa kokonaisuutta, syntyi tarve työkalujen yhdenmukaistamiselle. Näin ollen siirryttiin käyttämään Eclipse-nimistä kehitystyökalua, joka on niin ikään alustariippumaton IDE.

5 PROTOKOLLISTA, PROTOKOLLAPINOSTA JA TIETOKONEIDEN VÄLISESTÄ KOMMUNIKAATIOSTA

Tietokoneiden välisen kommunikaation yksi keskeinen käsite on protokolla. Suomen sivistyssanakirja määrittelee protokollan diplomatiassa noudatettaviksi säännöiksi ja muodoiksi, ja tietoliikenteen ja tietokoneiden kontekstissa merkitys on saman kaltainen. Tietoliikenneprotokolla määrittelee ne säännöt, toimintatavat ja viestien rakenteet, joita kyseistä protokollaa käyttävät keskustelun osapuolet käyttävät kommunikoinnissa.

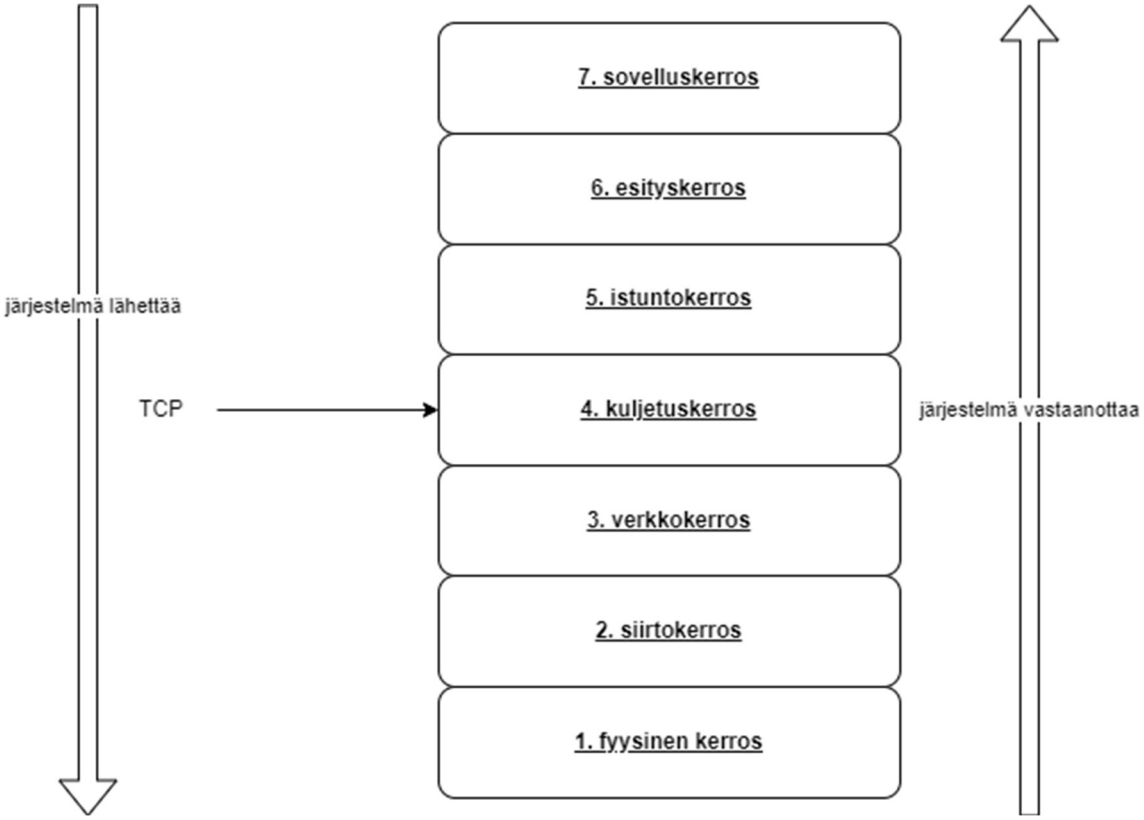
Tietokoneiden välinen kommunikointi on äärimmäisen monimutkaista, ja data, matkatessaan sovellukselta toiselle, kulkee monien eri protokollien välityksellä: Tietokoneen ruudulla esitetty data pitää saada muutettua tietokoneen ymmärtämään muotoon, tavuiksi ja biteiksi. Tämän jälkeen nämä tavut ja bitit täytyy muuntaa siirtotielle sopivaan formaattiin – vaikkapa sähkösignaaleiksi, elektromagneettisiksi aalloiksi tai valopulsseiksi. Oli siirtotie mikä tahansa, sen toisessa päässä olevan kohteen täytyy onnistua tulkitsemaan siirtotiellä tapahtuvat fysikaaliset muutokset oikein, jotta ne voidaan muuntaa biteiksi, tavuiksi ja jälleen ihmisen ymmärrettäväksi dataksi. (Peterson, L., 2011, kappale 1.3.1)

Monimutkaisuutta lisää lähes aina se tosiseikka, että siirtotie on hyvin harvoin varattu vain kahden keskusteluosapuolen käyttöön. Tämän vuoksi reitille tarvitaan reitityslaitteita, joiden tehtävä on välittää viestejä kohteelta toiselle ja mekanismeja, joilla samaa siirtotietä voi käyttää useampi keskustelija samanaikaisesti. Jokaisessa tiedonsiirron vaiheessa on valittavissa lukemattomia erilaisia protokollia. Onneksi koko kokonaisuutta ei tarvitse hallita kerralla, vaan protokollat on jäsennelty protokollapinoiksi, joissa kukin protokolla käyttää alemman kerroksen palveluja, ja tarjoaa palveluja ylemmille kerroksille.

Yleisellä tasolla protokollapinoja mallinnetaan pääasiassa kahdella eri mallilla: OSI-mallilla, ja TCP/IP-mallilla.

5.1 OSI-malli

OSI-malli jakaa verkkoliikenteessä käytetyt toiminnot seitsemään eri kerrokseen, jossa kukin kerros käyttää yhtä alemman kerroksen palveluja, ja antaa palveluja yhtä ylemmän kerroksen käyttöön.



KUVA 2: OSI-malli

Yllä olevassa kuvassa on mallinnettu OSI-malli kokonaisuudessaan ja merkitty tälle työlle keskeisen TCP-protokollan sijoittuminen. Kuvan mukaisesti datan lähtiessä käyttäjältä, se kulkee protokollapinoa ylhäältä alaspäin. Dataa vastaanotettaessa, protokollapinoa puolestaan matkustetaan alhaalta ylöspäin.

Seuraavaksi esitetään lyhyet kuvaukset OSI-mallin eri kerroksista.

5.1.1 Fyysinen kerros

Fyysisen kerroksen tehtävä on huolehtia fyysisen muutoksen tapahtuminen siirtotiellä. Fyysinen kerros on siis fyysinen media – kaapeli, valokuitu, radioaalto – joka huolehtii, että siirtotiellä tapahtuvat fyysiset muutokset tapahtuvat oikein ja ajallaan. Fyysisen kerroksen protokollat määrittelevät esimerkiksi käytettävät jänniterajat, kaapelit, liittimet, modulointitekniikat ja kanavointimenetelmät. (Kumar, S. 2014.)

5.1.2 Siirtokerros

Siirtokerros kokoaa raa'at bitit loogisiksi kokonaisuuksiksi. Sen tehtävä on hallinnoida toisiinsa suoraan kytkettyjen laitteiden välistä yhteyttä, yksilöidä toisiinsa suoraan kytkettynä olevat laitteet, sekä havaita ja korjata siirtotiellä tapahtuneita virheitä. Siirtokerroksen tehtävä on myös tarjota ylemmille kerroksille pääsy siirtotiehen, sekä valvoa milloin siirtotielle on soveliasta lähettää dataa. (Kumar, S. 2014.)

5.1.3 Verkkokerros

Verkkokerros mahdollistaa datan kuljetuksen kahden eri verkossa olevan päätepisteen välillä. Sen vastuulla on verkkopakettien oikea reititys, parhaan reitin valitseminen silloin kun vaihtoehtoja on useita, ja looginen osoitteistus – nykyään yleensä IP-osoitteiden avulla. Verkkokerroksen protokollat koskevat näin ollen erilaisia loogisen osoitteistuksen tapoja, ja proseduureja "parhaan" reitin valitsemiseksi kahden pisteen välillä, miten "paras" missäkin protokollassa määritellään. (Kumar, S. 2014.)

5.1.4 Kuljetuskerros

Kuljetuskerroksen protokollat abstrahoivat alemmat kerrokset näkyvistä ja antavat ylempien protokollien kohdella yhteyttä yksinkertaisemmin pisteestä pisteeseen yhteytenä. Siirtokerroksen protokollien tehtäviin kuuluvat vuonhallinta, segmentointi, virheenkorjaus ja yhteydellisen ja yhteydettömän yhteyden muodostaminen. (Kumar, S. 2014.)

5.1.5 Istuntokerros

Istuntokerroksen tehtävä on huolehtia yhteyden siististä avaamisesta, ylläpidosta ja sulkemisesta. Lisäksi istuntokerros vastaa yhteyden autentikoinnista ja pitää huolen, että vaadittavat tiedot yhteyden tarvitsemiin lupiin ovat lähetettävissä. Lopuksi istuntokerroksen protokollat varmistavat, että yhteyden toisesta päästä saapuvat tiedostot saatetaan oikealle paikalleen. (Kumar, S. 2014.)

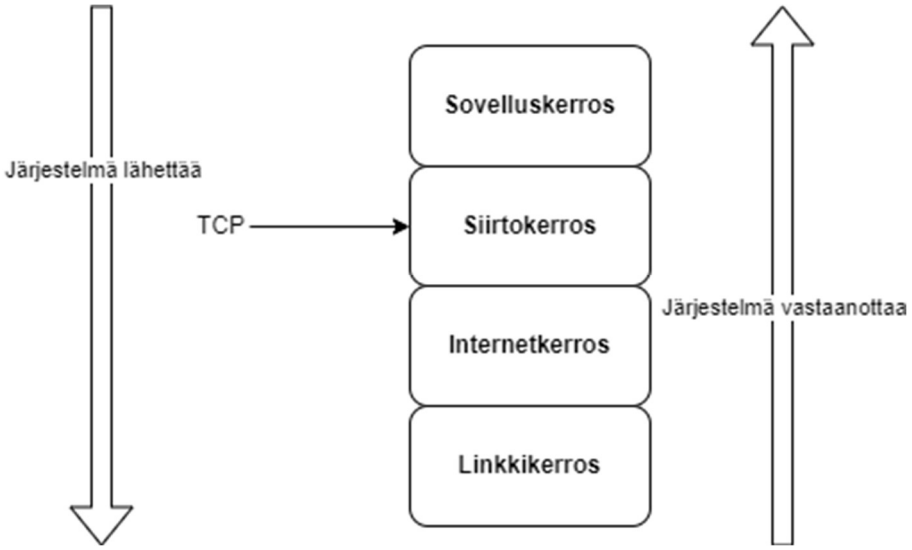
5.1.6 Esityskerros

Esityskerroksella on kolme tehtävää: Käännös, pakkaus, ja salaus. Toisin sanoen esityskerroksen protokollat määrittävät halutun salausalgoritmin, tavan, jolla ihmisen luettava data saatetaan koneluettavaan binäärimuotoon, ja halutun proseduurin lähetettävien tiedostojen tiivistämiseksi pienempään tilaan, jotta tiedonsiirtonopeus kasvaa. (Kumar, S. 2014.)

5.1.7 Sovelluskerros

Sovelluskerroksen protokollat mahdollistavat verkkoyhteyttä vaativien sovellusten toiminnan. Näitä protokollia ovat muun muassa http ja https, Telnet, FTP ja IRC. (Kumar, S. 2014.)

5.2 TCP/IP -malli



KUVA 3: TCP/IP-malli

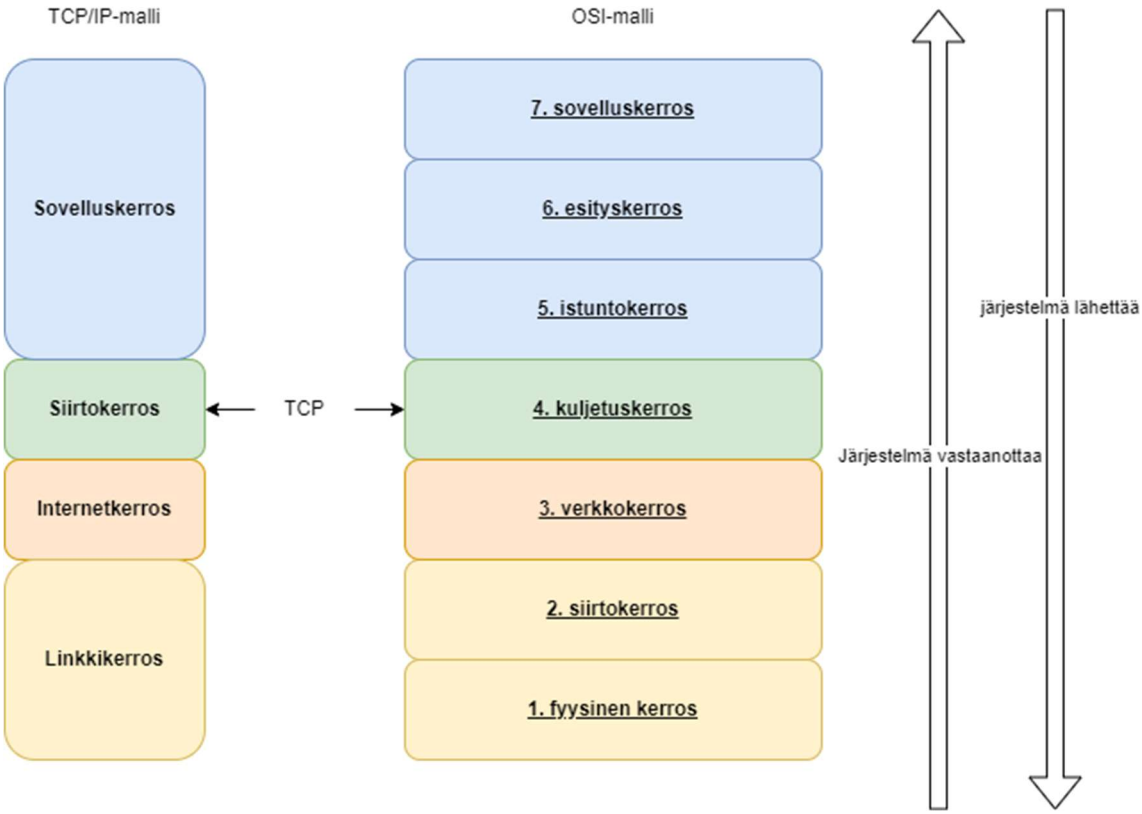
TCP/IP malli jakaa tiedonsiirtoprotokollat seitsemän kerroksen sijaan neljään eri kerrokseen, ja OSI-mallin tapaan kukin protokollakerros käyttää alemman kerroksen palveluja, tarjoten omia palvelujaan kerrosta ylemmäs. TCP/IP-mallin kerrokset ovat linkkikerros, internetkerros, siirtokerros, ja sovelluskerros. (Kozierok, C. 2005.)

Linkkikerros pitää sisällään protokollat, joiden toimialueena on kahden toisiinsa fyysisesti kytketyn laitteen välinen kommunikaatio, mutta sinä ei oteta kantaa fyysisiin yhteyksiin. Internetkerroksen protokollien vastuulla on verkkojen välinen liikenne ja reititys, kun taas siirtokerroksen protokollat varmistavat tiedon virheettömän siirron vuonhallinnalla, ruuhkautumisenestomekanismeilla ja uudelleenlähetyksellä. Lopuksi applikaatiokerroksen protokollat tuovat alempien kerrosten muodostaman yhteyden ja siinä kulkevan datan varsinaisen yhteyttä hyödyntävän sovelluksen käyttöön. (Kozierok, C. 2005.)

OSI-mallista poiketen, TCP/IP-malli ei ota kantaa fyysiseen kerrokseen, vaan olettaa että yhteyden muodostamiseen tarvittava siirtotie on kunnossa. Muutoin TCP/IP-mallin suurin ero OSI-malliin verrattuna on kahden alimman OSI-kerrok-

sen yhdistäminen linkkikerrokseksi, ja istunto-, esitys- ja sovelluskerroksen yhdistäminen sovelluskerrokseksi. TCP/IP-malli ei siis jätä protokollia tai kerroksia pois OSI-malliin verrattuna, vaan yhdistää niitä toisiinsa. (Fortinet. n.d.)

5.3 Mallien hyödyllisyys ja käyttö



KUVA 4: TCP/IP-mallin ja OSI mallin vertautuvuus

OSI-malli on erittäin hyödyllinen analysoidessa tietojärjestelmien välistä kommunikointia, mutta sitä tarkasti seuraavia protokollapinoja ei juurikaan ole toteutettu. Sovellettaessa käytännön tasolle se on usein liian yksityiskohtainen ja käytännön protokollien rajaaminen tarkasti OSI-kerrosten mukaan on hankalaa. TCP/IP-mallin laveammat kerrokset antavat protokolleille hiukan enemmän liikkumavaraa ja käytännön protokollapinot ovatkin helpompia rakentaa näillä laveammilla määritelmillä. (Tanenbaum, A. 2013.)

Tästä huolimatta OSI-mallin hyödyllisyyttä ei tule aliarvioida, sillä se tarjoaa tarkan mallin siitä mitä kahden tietojärjestelmän välisessä kommunikoinnissa tulee

ottaa huomioon ja auttaa ongelmanratkaisussa, sillä diagnostiikka voidaan kohdistaa tarkemmin OSI-mallin avulla.

6 DPDK LYHYESTI

Vaikka DPDK on keskeinen osa tässä työssä esitellyn TCP-toteutuksen vaatimusmäärittelyä, ei tässä työssä kuitenkaan juurikaan avata DPDK:ta. Tämä kappale on tarkoitettu vain ja ainoastaan perusteiden avaamiseksi, jotta asiaan täysin vihkiytymätön lukija ymmärtää DPDK:n käytön tarpeellisuuden.

6.1 DPDK-ohjelmointirajapinta yleisesti

DPDK, eli **D**ata **P**lane **D**evelopment **K**it, on Linux Foundationin hallinnoima Open Source-projekti, joka tarjoaa joukon kirjastoja ja ajureita ohjelmoijan käyttöön. Näiden kirjastojen ja ajurien avulla on mahdollista kommunikoida suoraan laitteen verkkokortin kanssa, ilman verrattain hitaan Linux verkkopinon väliintuloa. (Saarela, J. 2021., sivut 8–9)

6.2 DPDK:n keskeiset ominaisuudet

DPDK:n mukana tulevista kirjastoista noin viisi muodostavat DPDK:n ydintoiminnallisuuden. Nämä kirjastot ovat timer-kirjasto ajastuksen hallintaa varten, muistiin liittyviä seikkoja hallitsevat ring- ja mempool-kirjastot, viestipuskureita hallinnoiva mbuf-kirjasto, sekä EAL, eli **E**nvironment **A**bstraction **L**ayer, joka sitoo edellä mainitut yhteen. Lisäksi DPDK:ssa on oma malloc-kirjasto, joka tarjoaa C-ohjelmointikielen malloc-funktion toiminnallisuudet DPDK:n ominaisuuksilla. (Saarela, J. 2021., sivu 8)

6.2.1 EAL-rajapinta

EAL on geneerinen rajapinta, jolla on suora pääsy ohjelmistoa pyörittävän laitteen “rautaan” – sen muistiin ja I/O laitteisiin. Sen tarkoitus on tarjota paljon erilaisia toimintoja käytettäväksi ja sen vastuulla on moni DPDK:n toimintaan liittyvä keskeinen seikka, kuten esimerkiksi resurssien allokointi, alustukset, prosessien jakaminen ydinten kesken ja itse DPDK:n käynnistys. (Environment Abstraction Layer.)

6.2.2 Timer-kirjasto

Timer-kirjasto mahdollistaa tapahtumien ajastamisen. Ajastimen tarkkuus riippuu siitä, kuinka usein kirjaston rte_timer_manage-funktiota kutsutaan. Haluttaessa tarkempaa ajastusta funktiota voidaan kutsua useammin, mutta tiuhat funktiokutsut heikentävät suorituskykyä. (Timer Library.)

Ero perinteisiin ajastuksiin on se, että DPDK:ta käytettäessä käyttöjärjestelmä ei puutu prosessien suoritukseen vaan prosessi voidaan määrittää pyörimään jatkuvasti tietyllä ytimellä. Tämän ansiosta ajastimen tarkkuus on tarkasti määriteltävissä ja ajastin on myös täysin deterministinen, koska käyttöjärjestelmä ei lähde keskeyttämään prosessia omin lupinensa. (Timer Library.; Zhu, H.(toim.) 2021., kappale 1.3.2.)

6.2.3 Mbuf-kirjasto

Mbuf-kirjastossa on määritelty muistipuskureita, joilla on helppo käsitellä esimerkiksi verkkopaketteja, tai vaikkapa järjestelmän ohjausviestejä. Jokaisessa puskurissa on varattu tila saapuvan paketin metadatalle, jota seuraa vakioitu tila itse paketin datalle. Edellä mainitun tilan suuruus on määriteltävissä, mutta yleensä se vakioidaan kahteen kilotavuun. Muistipuskureita on mahdollista myös ketjuttaa suurien pakettien käsittelyä varten, jolloin vain ketjun ensimmäisessä muistipuskurissa on siirrettävän paketin metadata, sekä ketjussa seuraavan puskurin osoite. (Zhu, H.(toim.) 2021., kappale 6.6.)

Muistipuskurit toimivat pakettien säilytyslaatikoina. Paketin tullessa sisään verkkokortilta, varataan sille sopiva muistipuskuri, johon paketti laitetaan. Paketin lähtiessä tämä muistipuskuri vapautetaan. Puskurien vaatima tila varataan käynnistyksen yhteydessä, joten ajon aikana ei tarvitse varata lisää muistia. (Zhu, H.(toim.) 2021., kappale 6.6.)

6.2.4 Muistinkäsittely

DPDK:n muistinkäsittely hyödyntää muutamia tekniikoita nopeuden ja tehokkuuden lisäämiseksi. Muistin sivutuksessa käytetään suuria sivukokoja ja muistiosoitteina käytetään suoria fyysisiä osoitteita virtuaalisten osoitteiden sijaan. DPDK on myös tietoinen siitä, miten järjestelmän fyysinen muisti on jäsennelty, ja pystyy käyttämään tätä tietoa hyväksi muistiin liittyvissä operaatioissa. Lisäksi muistin allokointiin varattu malloc-funktio on toteutettu DPDK:ssa siten, että se hyödyntää koko DPDK:n tehokkaan muistinkäsittelyn arsenaalia. (Memory in data plane development kit.)

6.2.5 Poll Mode-ajurit

Viimeinen DPDK:n toiminnan keskeinen seikka on PMD, eli Poll Mode -ajuri (**P**oll **M**ode **D**river). Nämä ajurit keskustelevat suoran järjestelmän verkkokortin kanssa käyttäjäavaruudesta, ohittaen kernel-avaruuden kokonaan. (Poll Mode Driver, n.d.)

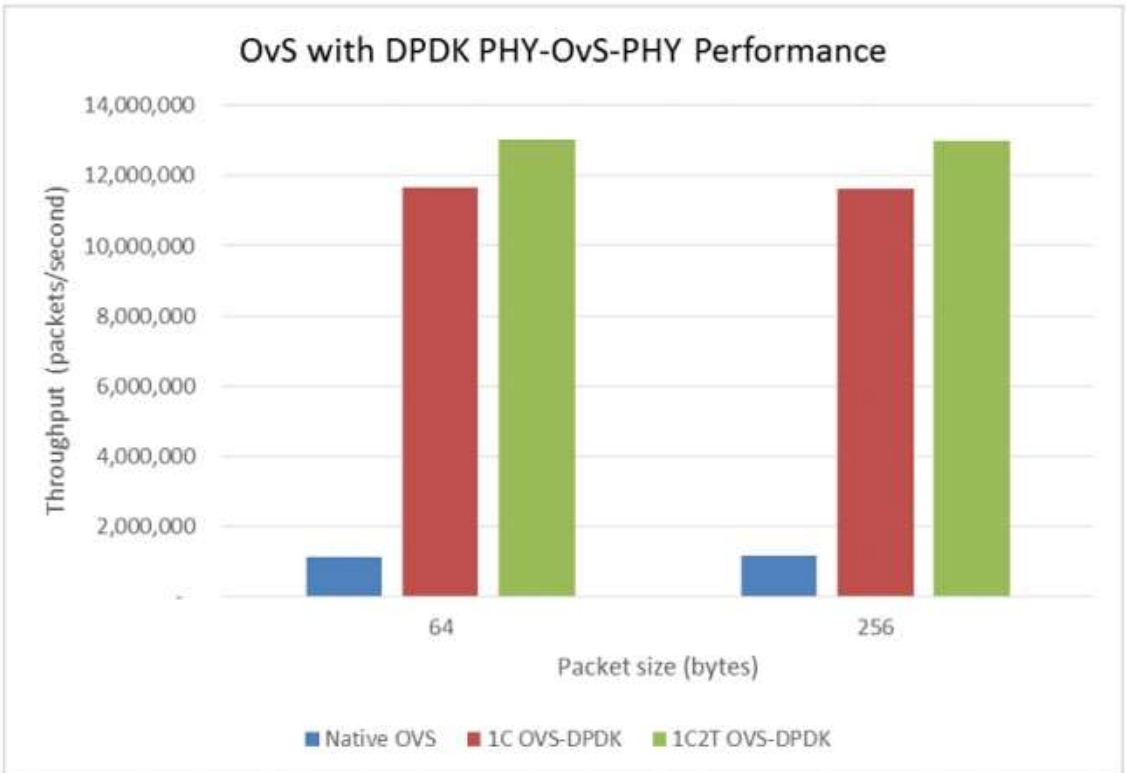
Perinteisessä verkkopakettien käsittelyssä verkkokortin täytyy lähettää keskeytyspyyntö prosessorille verkkopaketin saapuessa verkkokortille. Prosessorin täytyy tämän jälkeen siirtää keskeytetyn prosessin data rekisteriin, suorittaa keskeyttävä prosessi ja hakea tallennettu keskeytetyn prosessin data rekisteristä ennen kuin keskeytettyä prosessia voidaan jatkaa. Tämä kontekstin vaihto on aikaa vievä prosessi, mutta yleensä ympäröivään järjestelmään nähden hyvinkin nopea. Ongelma muodostuu verkkokorttien ja -yhteyksien kehittyessä nopeam-

maksi, jolloin paketteja tulee verkkokortille aina vain useammin ja keskeytyssignaaleja lähetetään yhä tiuhempaan. Lopulta saavutetaan piste, jolloin keskeytysten viemä aika kasvaa merkittäväksi järjestelmän toiminnan kannalta. (Zhu, H.(toim.) 2021., kappale1.3.1.)

PMD:t ratkaisevat tämän ongelman kyselemällä jatkuvasti paketteja suoraan verkkokortilta, käyttäjäavaruuden sisältä. Näin keskeytystä ei tarvita, koska kysely ja mahdollinen datan käsittely ja edelleen lähetys tapahtuvat kaikki käyttäjäavaruudessa, mahdollisesti jopa yhden ja saman prosessin sisällä. (Poll Mode Driver, n.d.)

6.2.6 DPDK:n vaikutus suorituskykyyn

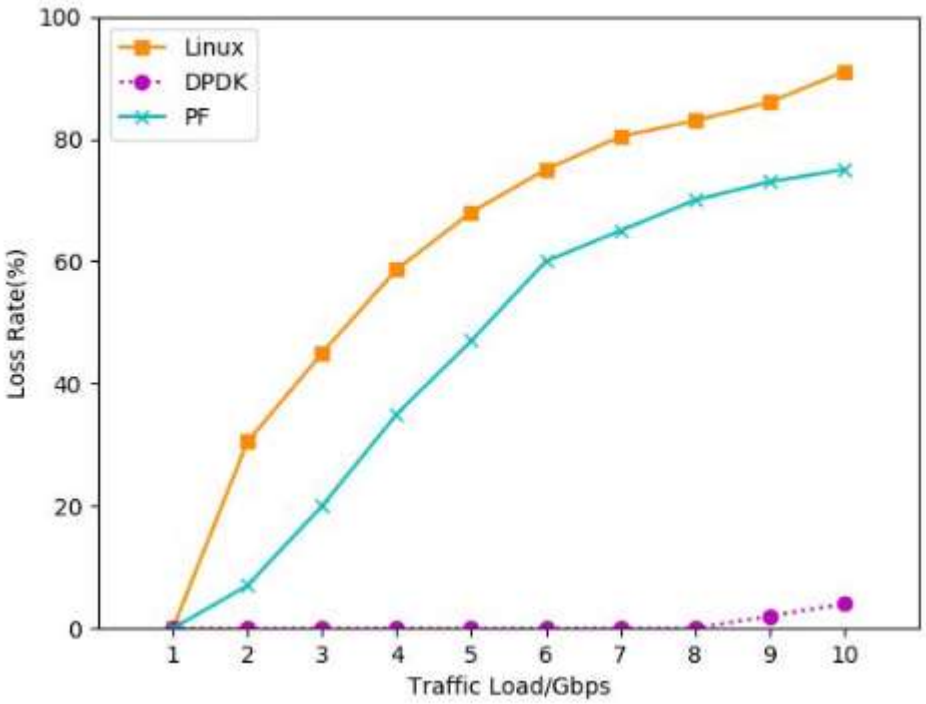
Yllä kuvattujen seikkojen vuoksi DPDK:n hyödyntäminen verkkopakettien käsittelyssä lisää huomattavasti tehokkuutta verkkokorttien ja -yhteyksien nopeuksien kasvaessa. Robin Gillerin vuonna 2016 julkaisema testi OpenVSwitch-verkkokytkimen suorituskyvystä DPDK:n kanssa ja ilman sitä osoittaa yli kymmenkertaisen suoritustehon kasvun, kun DPDK otetaan käyttöön verkkokytkimellä (kuva 5).



KUVA 5: DPDK:n vaikutus kytkimen läpi kulkevien pakettien lukumäärään (Geller, 2016)

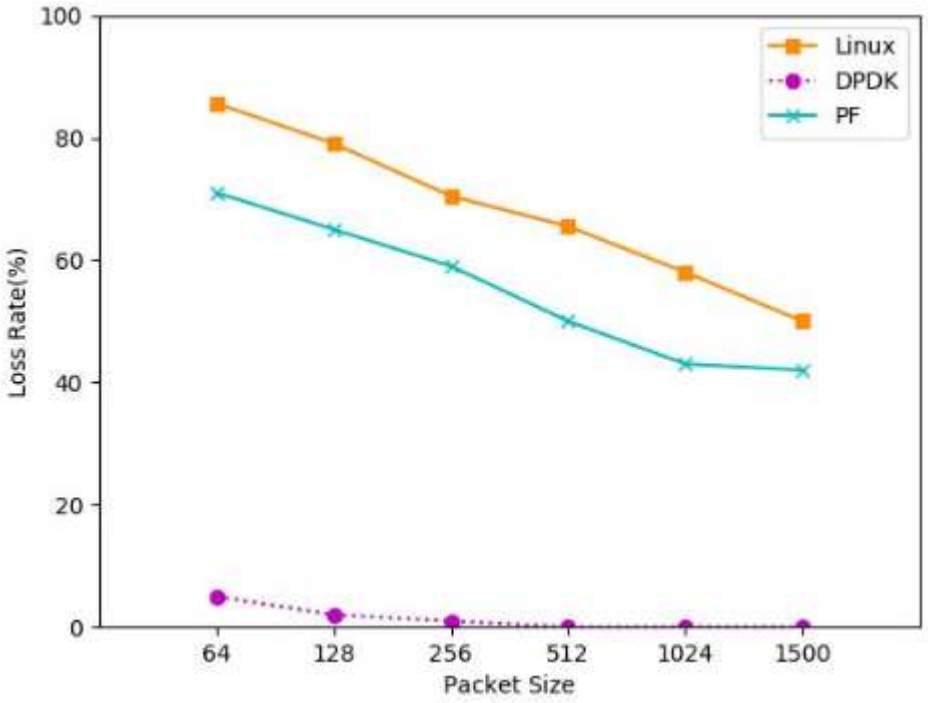
Vuonna 2018 julkaistussa tutkimuksessa (Wenjun Zhu et al) puolestaan tutkittiin DPDK:n suorituskykyä verrattuna perinteiseen Linuxin verkkopinoon. Tutkimuksen mukaan nopeammat verkkoyhteydet lisäävät menetettyjen verkkopakettien osuutta koska järjestelmän verkkopinot eivät pysy nopeamman yhteyden mukana. Tutkimuksessa todettiin huomattava lasku menetettyjen pakettien määrässä sekä suuren pakettivolyymin, että suuren pakettikoon ollessa kyseessä.

Alla olevassa kuvassa 5 on esitetty pakettivolyymin kasvattamisen vaikutus menetettyjen pakettien osuuteen.



KUVA 6: Liikennemäärän vaikutus hävikkiprosenttiin eri järjestelmillä (Zhu, 2021)

Kuvassa 7 puolestaan on esitetty pakettien koon vaikutus samaan suureeseen



KUVA 7: paketin koon vaikutus hävikkiprosenttiin eri järjestelmillä (Zhu, 2021)

Suuri hävikkiprosentti luonnollisesti kääntyy hitaammaksi tiedonsiirtonopeudeksi toistuvien uudelleenlähetysten vuoksi. Näin ollen yllä olevista kuvista nähdään DPDK:n huomattava positiivinen vaikutus tiedonsiirtonopeuteen.

7 TCP-PROTOKOLLA

TCP (Transmission Control Protocol) on full-duplex, yhteydellinen tiedonsiirtoprotokolla, joka sijoittuu OSI mallin siirtokerrokselle (Transport layer, kerros 4) ja TCP/IP mallin verkkokerrokselle. Tässä kappaleessa kerrotaan yksityiskohtaisesti TCP:n tehtävistä ja toiminnasta, sekä avataan TCP:n PDU:n – TCP-segmentin – sisältöä. Kirjallisuudessa TCP-segmentistä käytetään joskus myös nimitystä TCP-paketti, mutta tässä opinnäytetyössä käytetään yksinomaan nimeä segmentti, sekaannusten välttämiseksi.

7.1 TCP yleisesti

TCP-protokollan vastuulla on varmistaa, että data saapuu kokonaisena ja virheettömänä sekä oikeassa järjestyksessä yhteyden vastaanottavalle osapuolelle. Nämä tavoitteet saavutetaan kolmella eri tavalla: Vastaanottajan tulee lähettää lähettäjälle vahvistusviesti vastaanotettuaan datan oikein, datan oikeellisuus itsessään tarkistetaan tarkastussumman avulla, ja tapauksissa, joissa lähetettävä kokonaisuus tarvitsee pilkkoa useampaan segmenttiin, nämä segmentit numeroidaan sekvenssinumeroilla, jotta ne voidaan vastaanottopäässä laittaa oikeaan järjestykseen riippumatta todellisesta saapumisjärjestyksestä. Näin voidaan varmistaa, että TCP:hen luottavat palvelut voivat keskittyä täysin omiin tehtäviinsä ja luottaa siihen, että niille toimitetun datan bitit ovat samassa järjestyksessä kuin lähettäjä on ne alun perin lähettänyt (ja kääntäen: lähettäjä voi luottaa siihen, että lähetetyt bitit saapuvat varmasti perille oikeassa järjestyksessä). (RFC 793.)

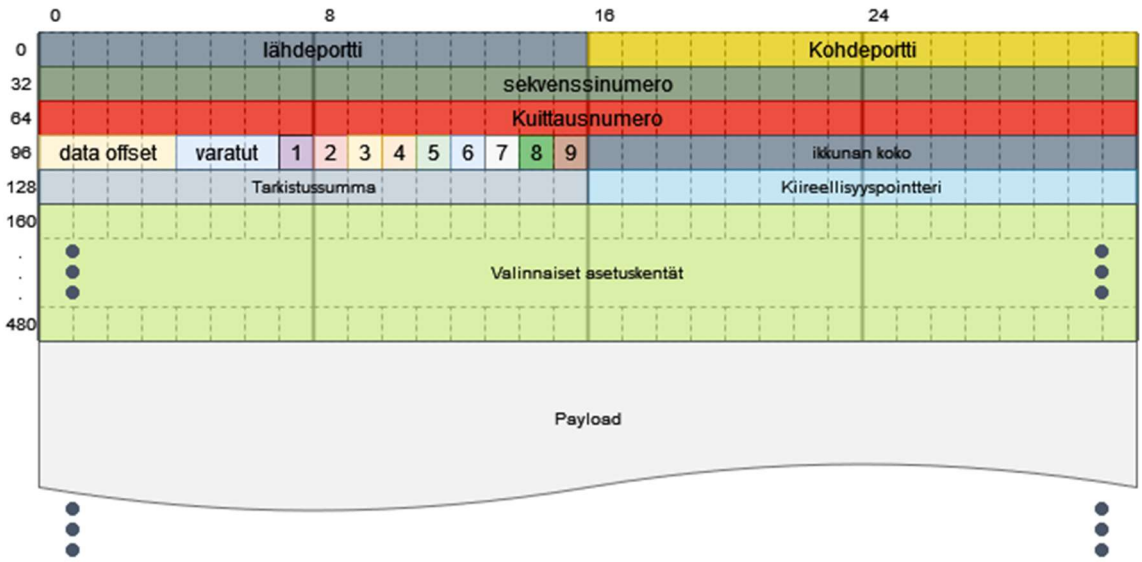
TCP on yhteysprotokolla, joten siihen rakennetut varmistukset suojaavat datan oikeellisuutta vain teknisiä vikoja vastaan. TCP ei estä älykästä kolmatta osapuolta kaappaamasta viestiä ja muokkaamasta sen tarkistussummaa ja sisältöä. TCP ei myöskään käytännössä voi tehdä mitään korjatakseen katkenneita yhteyksiä, mutta yhteydellisen luonteensa vuoksi protokolla mahdollistaa tiedottamisen yhteyksien katkeamisesta ylemmille kerroksille. (Dostalek, L., kappale 9.)

Kuljetuskerroksen protokollana TCP:n vastuualueeseen kuuluvat vuonhallinta, yhteydellisen yhteyden luominen (TCP:n yhteydetön serkku on UDP, **U**ser **D**atagram **P**rotocol), segmentointi ja virheenkorjaus. Tulevissa kappaleissa esitellään mekanismit, joilla TCP huolehtii näistä vastuualueista. (Thiyari, K.)

7.2 TCP-segmentti

TCP protokollan paikka OSI-mallin kerroksella neljä tarkoittaa käytännössä sitä, että TCP protokolla käsittelee IP-paketin dataosuutta, johon on lähetyspäässä lisätty TCP-otsikko. TCP-protokolla riisuu vastaanottamastaan datasta TCP-otsikon osuuden ja antaa jäljelle jäävän datan ylemmille kerroksille. Vastaavasti dataa lähetettäessä, TCP-protokolla vastaanottaa datan ylemmältä kerrokselta, lisää siihen TCP-otsikon ja antaa näin muodostetun TCP-segmentin alaspäin verkkokerrokselle.

TCP-segmentti koostuu TCP-otsikosta ja siirrettävästä datasta (payload). TCP-otsikossa on protokollan kannalta tärkeää informaatiota, joten tässä kappaleessa keskitytään lähinnä TCP-otsikon sisällön kuvaamiseen. (Eddy, W. 2022., kappale 3.1)



KUVA 8: TCP-segmentti

TCP-otsikon ensimmäiset kaksi tavua on varattu lähdeportin numerolle, ja jälkimmäiset puolestaan kohdeportille. Nämä kaksi lukua muodostavat osan yhteyden yksilöimiseen tarvittavasta informaatiosta. Yhteys voidaan yksilöidä täysin, kun tiedetään lähde- ja kohdeporttien lisäksi myös lähteen ja kohteen IP-osoitteet. (Eddy, W. 2022., kappale 3.1; RFC 6093., kappale 4.1.)

Kohdeportin jälkeiset neljä tavua on varattu sekvenssinumerolle. Tämä numero toimii payload-tavujen järjestysnumerona, jotta eri reittejä, eri järjestyksessä saapuneet payload-tavut voidaan kasata vastaanottopäässä eheäksi kokonaisuudeksi. (Eddy, W. 2022., kappale 3.1)

Sekvenssinumerointi ei yleensä ala nollasta, eikä mistään ennalta määrätystä vakiosta, vaan yhteyttä muodostaessa yhteyden kumpikin pää arpoo toisistaan riippumatta oman ensimmäisen sekvenssinumeronsa (initial sequence number, ISN). Sekvenssinumeron on sallittua ylivuotaa nollaksi ja aloittaa laskeminen 32 bittisen lukuavaruuden alusta, ja tämän aiheuttamaa ongelmaa käsitellään myöhemmin tässä teoriaosuudessa. (Eddy, W. 2022., kappale 3.4.1)

Sekvenssinumeron jälkeiset neljä tavua on varattu kuittausnumerolle. Tämä numero kertoo mitä sekvenssinumeroa lähettäjä odottaa, ja näin ollen kuittaa kaikki siihen mennessä lähetetyt tavut vastaanotetuiksi. (Eddy, W. 2022., kappale 3.1)

Neljän bitin pituinen data offset-kenttä kertoo TCP-otsikon kokonaispituuden ilmaistuna 32 bitin, eli neljän tavun, mittaisina sanoina ja sitä seuraa kolme bittiä, jotka on varattu tulevaisuuden käyttötarkoituksia varten. Nämä kolme varattua bittiä tulee asettaa nollaksi. (Eddy, W. 2022., kappale 3.1)

Varattuja bittejä seuraa kahdeksan niin kutsuttua lippubittiä, joiden merkitykset ovat seuraavat:

1. ECN nonse. RFC 3540 määrittelee tämän lipun vahvistamaan eksplisiittisten ruuhkautumisilmoitusten mekanismia, mutta se on kokeellisella tasolla, ja tässä työssä sitä kohdellaan kuten varattua bittiä.
2. Congestion window reduced (CWR). Tämän lipun asettaminen arvoon 1 kertoo viestin vastaanottajalle, että lähettäjä on saanut ja tunnistanut TCP-

segmentin, jonka ECE-lippubitti on ylhäällä ja reagoinut haluamallaan ruuhkautumisenhallintamekanismilla.

3. ECE, eli ECN-Echo: ECN-Echo-lipulla on kaksi merkitystä, riippuen SYN-lipun tilasta. SYN-lipun ollessa ylhäällä, viestin lähettäjä kommunikoi tällä lipulla kyvykkyytensä eksplisiittisiin ruuhkautumisilmoituksiin (ECN, Explicit congestion notification). SYN-lipun ollessa alhaalla, tämän lipun nostaminen tarkoittaa, että lähettäjä on saanut IP kerrokselta ilmoituksen uhkaavasta tai jo meneillään olevasta ruuhkautumisesta.
4. Kiireellisyyslippu. Tämän lipun asettaminen ykköseksi kertoo vastaanottajalle, että kiireellisyyspointteri on merkityksellinen.
5. ACK, eli kuittauslippu. Lipun asettaminen arvoon yksi osoittaa kuittausnumero-kentän olevan merkityksellinen. Tämä lippu tulee olla ylhäällä kaikissa lähetetyissä viesteissä lukuun ottamatta kunkin yhteysosapuolen ensimmäistä viestiä.
6. Push-lippu. Tämä lippu pyytää vastaanottajaa lähettämään payload datan vastaanottavalle sovellukselle niin pian kuin mahdollista. Joissain tapauksissa siirtotehokkuuden lisäämiseksi vastaanottaja saattaa puskuroida tietyn määrän dataa ennen kuin koko puskuri annetaan keralla sovellukselle. Push-lipun tarkoitus on estää tämä toiminnallisuus.
7. Reset-lippu. Tämän lipun asettaminen ykköseksi pyytää vastaanottajaa pudottamaan kyseisen yhteyden välittömästi, ilman tavallista yhteyden purkuun liittyvää proseduuria.
8. SYN-lippu, eli synkronointilippu. Tämän lipun tulee olla ylhäällä vain yhteysosapuolten ensimmäisissä viesteissä, ja se ilmoittaa, että lähettäjä aloittaa uuden sekvenssinumerolaskennan. Näin ollen sekvenssinumero-kentässä on uusi ISN.
9. FIN-lippu, eli lopetuslippu. Lähettäjä lähettää FIN-lipulla varustetun viestin, jos ja vain jos lähettäjällä ei enää ole lähetettäviä payload-tavuja.

(Eddy, W. 2022., kappale 3.1; RFC 6093)

Lippubittejä seuraa kaksitavuinen Window Size -arvo, joka määrittää suurimman mahdollisen tavumäärän mitä lähettäjä on valmis vastaanottamaan, ts. suurimman mahdollisen luvun minkä vastaanottaja voi lisätä edellisen lähettämänsä segmentin sekvenssinumeroon. (Eddy, W. 2022., kappale 3.1)

Window Size -arvoa seuraavat kaksitavuiset tarkistussumma- ja kiireellisyyspointteriarvot. Tarkistussummaa käytetään segmentin virheettömyyden varmistamiseen ja kiireellisyysosoitin osoittaa toteutuksesta riippuen joko viimeisen kiireellisen tavun järjestysnumeron tai viimeistä kiireellistä tavua seuraavan tavun järjestysnumeron. RFC 793 pyrki kommunikoimaan edellä mainituista vaihtoehdoista ensimmäistä, ja myöhemmin RFC 1122 pyrki vahvistamaan tätä tulkintaa, mutta suurin osa TCP-protokollaa käyttävistä sovelluksista oli siirtynyt jo käyttämään jälkimmäistä. (RFC 793., kappale 3.1, sivu 17; RFC 1122., kappale 4.2.2.4, sivu 84; RFC 6093.)

Yllä kuvaillut 20 tavua muodostavat pienimmän mahdollisen TCP-otsikon, ja näin ollen myös pienimmän mahdollisen TCP-segmentin. Koska data offset -kenttä on pituudeltaan vain neljä bittiä, se voi saada arvoja väliltä [0,15]. Kuten aiemmin todettiin, data offset -kentän ”yksikkö” on neljän tavun mittainen sana. Näin ollen TCP-otsikon maksimipituus on 60 tavua.

7.3 TCP:n asetukset

Pakollisten 20 otsikkotavun jälkeen TCP-otsikkoon voi lisätä maksimissaan 10 valinnaista asetusta. Kukin asetus koostuu minimissään yhden tavun mittaisesta kentästä, joka kuvaa käytettävän asetuksen. Asetuksesta riippuen asetuksen numerokoodia seuraa joko yksitavuinen asetuksen pituuden kertova luku, tai edellä mainitun lisäksi tarvittava määrä asetuksen tarvitsemaa dataa. (Eddy, W. 2022., kappale 3.1, sivu 10–11)

Oheisessa taulukossa esitetään IANA:n standardisoimat olevat asetusnumerot ja niiden tarvitsemat parametrit:

Asetusarvo	Selite	Pituus	Data
0	Asetuslistan loppu		
1	Ei operaatiota (NOP)		
2	Segmentin maksimipituus	4	PP
3	Ikkunan skaalaus	3	S
4	SACK sallittu	2	

5	SACK	N	[AAAA,BBBB],[CCCC,DDDD],….
8	Aikaleima	10	[TTTT, EEEE]

TAULUKKO 1: Standardisoidut asetusarvot

Taulukon 1 datasarakkeessa yksittäinen kirjain vastaa yhtä tavua, esimerkiksi ikkunan skaalaus ottaa parametrinä yhden tavun, SACK taas joukon nelitavuisten arvojen muodostamia pareja.

Kahdeksanbittisen lukuavaruuden mahdollistamat muut tunnusluvut on tehty tarpeettomaksi jonkun toisen asetuksen toimesta, tai niitä ei ole standardoitu, niitä käytetään vastoin määriteltyä toimintatapaa tai ne ovat jotenkin muutoin erityisen epävarmoja käyttää. (IANA. 2022.)

7.3.1 Ei operaatiota ja asetuslistan loppu

Taulukossa 1 esitetyistä asetuksista koodit 0 ja 1 ovat melko yksiselitteisiä. Koodia 0 voi halutessaan käyttää merkitsemään optioalueen loppua, ja koodia 1 puolestaan suositellaan käyttämään täytteenä, jotta asetusalue saadaan sopimaan siististi 32:n bitin rajoihin. (Eddy, W. 2022., s. 12)

7.3.2 Segmentin maksimipituus

Segmentin maksimipituus-asetus, englanniksi Maximum segment size, on pituudeltaan neljä tavua, joista asetusnumero ja pituusarvo käyttävät kaksi. Jäljellä olevat kaksi tavua on varattu kokonaisluvulle, joka kertoo suurimman sallitun TCP-segmentin pituuden tavuina ilmaistuna. Tehokkuuden ylläpitämiseksi, tämä arvo asetetaan yleensä tarpeeksi pieneksi, että vältytään IP pakettien pirstaloitumiselta ja sitä seuraavilta tarpeettomilta uudelleenlähetyksiltä. Kumpikin puoli voi yhteyttä muodostaessa asettaa oman arvonsa tähän asetukseen, mikä helpottaa kahden suorituskyvyltään radikaalisti eroavan laitteen välistä kommunikaatiota. Asetusta voi käyttää yhteyden muodostamisvaiheen lisäksi myös yhteyden aikana, joten arvon dynaaminen muutos vastaamaan muuttuvia verkon olosuhteita on mahdollista. (Eddy, W. 2022., kappale 3.7.1)

7.3.3 Ikkunan skaalaus

Ikkunan skaalaus -asetus, englanniksi Window scale, on kolmen tavun mittainen asetus, jonka datakenttä kuvaa kuinka monta hyppyä vasemmalle (left-shift) ikkunan koko -arvon bittiesitystä siirretään. Datakentän arvo voi olla välillä [0,14], ja yhteyden kummankin pään pitää määritellä tämä asetus yhteyden muodostamisen aikana, jotta skaalattu ikkuna otetaan käyttöön. Kumpikin yhteyden pää voi asettaa ikkunan skaalaus -kenttään haluamansa arvon riippumatta siitä mitä yhteyden toinen pää on asettanut, olettaen että ikkunan skaalauksesta on päästy yhteisymmärrykseen yhteyttä muodostettaessa. (Eddy, W. 2022., kappale 2.)

7.3.4 Selektiivinen kuittaus (Selective Acknowledgement, SACK)

Selektiivinen kuittaus (SACK, Selective ACKnowledgement) on keskeinen ominaisuus TCP-protokollan tehokkaassa toiminnassa. Perinteinen TCP-protokollan kuittausmekanismi luottaa kumulatiivisiin kuittauksiin ja tämä mekanismi joutuu ongelmiin, kun lähetetyn segmenttivirran keskeltä putoaa paketteja. (RFC 2018.)

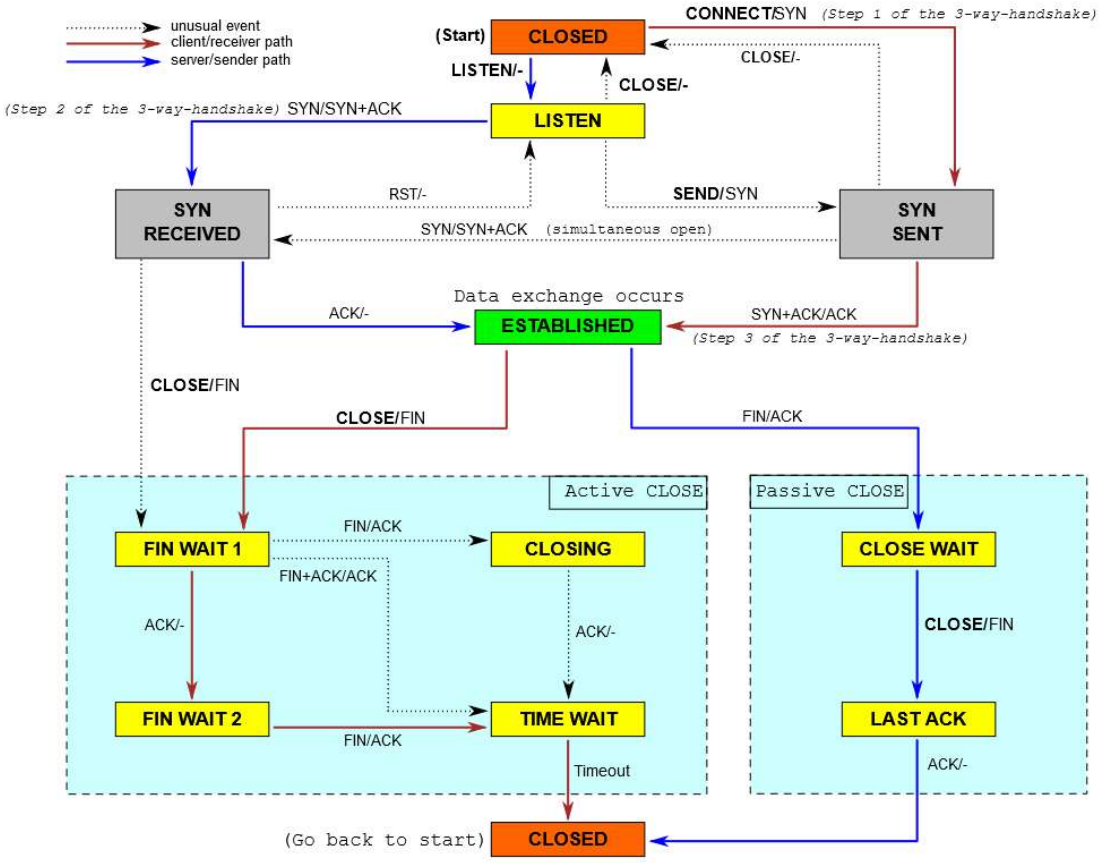
Kuvitteellisessa tilanteessa, jossa lähettäjä esimerkiksi lähettää tavut 1000:sta 10 000:een, 1000:n tavun segmenteissä, toisen segmentin (tavut 2000 - 2999 häviäminen matkalla pakottaa lähettäjän lähettämään kaikki segmentit sekvenssinumerosta 2000 alkaen, vaikka kaikki muut segmentit edellä mainittua lukuun ottamatta saapuisivat perille oikein. Selektiivinen kuittaus antaa vastaanottajalle keinon kuitata epäjatkuvan segmenttien ketjun. Tämän perusteella lähettäjä voi lähettää vain puuttuvat segmentit. (RFC 2018.)

Selektiivisen kuittauksen datakentässä on yhdestä neljään kappaletta lukuvälejä, esitettynä 32 bittisten kokonaislukujen pareina. Parin ”vasen laita” on inklusiivinen, ”oikea laita” taas eksklusiivinen. Yksittäinen lukuväli kertoo yhtäjaksoisesti oikein vastaanotettujen segmenttien sekvenssinumerot. (RFC 2018.)

7.3.5 Aikaleima

Aikaleima-asetus ei yleensä ole sidottu lähettävän tai vastaanottajan kelloon, eikä aikaleiman päivityksessä ole täysin vakiintunutta standardia. RFC 1323:ssa, tämän asetuksen määrittelevässä dokumentissa, ohjeistettiin ainoastaan, että aikaleiman tulee olla verrannollinen oikeaan aikaan nähden. Aikaleimaa voidaan käyttää muun muassa ratkaisemaan sekvenssinumeron ylivuoto-ongelma, josta lisää myöhemmin. Aikaleiman datakentässä on neljätavuinen lähettäjän aikaleima, jota seuraa neljätavuinen kopio viimeksi vastaanotetun segmentin aikaleimasta. (Eddy, W., kappale 3; RFC 1323. Kappale 4.)

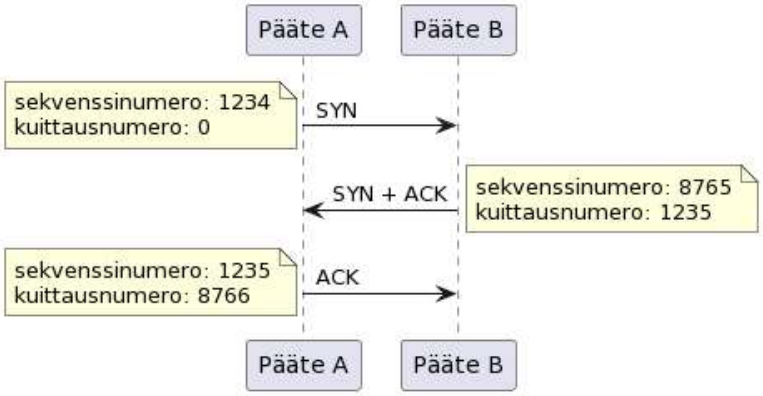
7.4 TCP:n toiminta ja keskeiset ominaisuudet



KUVA 9: TCP:n tilakaavio (Scil100, 2010)

Yllä olevassa kuvassa on esitetty TCP:n eri tilat sekä siirtymä niiden välillä. Tässä kappaleessa kuvataan protokollan toimintaa, keskeisiä ominaisuuksia ja eri tiloja yksittäisen TCP yhteyden aikana.

7.4.1 Yhteyden muodostaminen



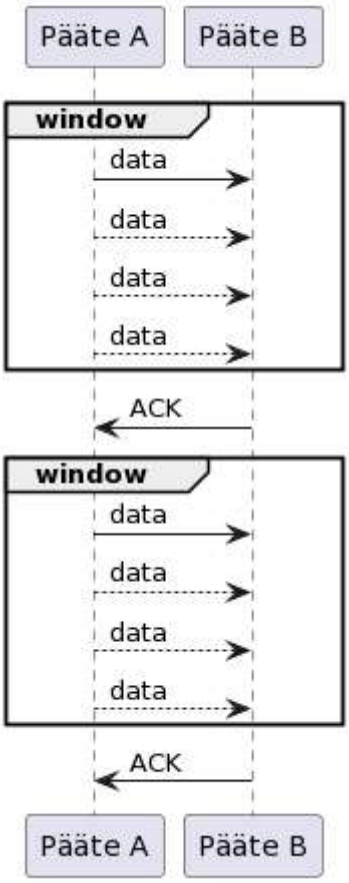
KUVA 10: TCP-yhteyden muodostaminen

TCP yhteys muodostetaan niin sanotulla kolmivaiheisella kättelyllä (three way handshake) jossa yhteyden osapuolet synkronoivat sekvenssinumeronsa, ja tarvittaessa neuvottelevat yhteyden lisäominaisuuksista. Kuvassa 10 on kuvattu tyypillinen TCP-yhteyden muodostaminen:

1. Pääte A lähettää SYN viestin päätteelle B, ja asettaa sekvenssinumeroksi satunnaisluvun *n*. ACK-lippu on 0 ja kuittausnumero niin ikään myöskin 0.
2. Pääte B lähettää SYN-ACK viestin päätteelle A. Viestissä molemmat liput SYN ja ACK ovat ylhäällä, viestin sekvenssinumerokenttään asetetaan satunnaisluku *m*, ja kuittausnumerokenttään arvo *n+1*.
3. Pääte A kuittaa edellisen segmentin vastaanotetuksi lähettämällä ACK viestin päätteelle B. Sen sekvenssinumero on *n+1* ja kuittausnumero *m+1*
4. Yhteys on muodostettu, datan siirto voidaan aloittaa.

(Eddy, W. 2022., kappale 3.5)

7.4.2 Datan lähettäminen



KUVA 11: datan lähetys ja kuittaus

TCP takaa parhaan kykynsä mukaan datan saapumisen virheettömänä vastaanottajalle. Saavuttaakseen tämän, TCP velvoittaa viestin vastaanottajan kuittaamaan vastaanottamansa viestin. Yllä olevassa kuvassa on esitetty datan lähetys päätteeltä A päätteelle B, mutta data voi aivan yhtä hyvin liikkua B:ltä A:lle.

Vaikka jokainen segmentti tulee kuitata, ei lähettäjän tarvitse välttämättä odottaa kuittausta lähettääkseen lisää dataa. Vasta kun vastaanottajan ilmoittama vastaanottoikkuna täyttyy (Window Size, muokattuna Window Scaling asetuksella, jos sellainen on), tulee lähettäjän odottaa aikaisemmin lähetetyt segmentit kuitatuiksi ennen uusien segmenttien lähettämistä. Kun jokainen segmentti kuitataan kumulatiivisesti, voidaan hukatut ja väärässä järjestyksessä saapuneet segmentit havaita.

Kadonneiden ja väärässä järjestyksessä saapuneiden segmenttien havaitsemiseen käytetään kahta tekniikkaa: Uudelleenlähetysajastinta (RTO, retransmission timeout) ja duplikaattikuittauksia (dupAck, duplicate acknowledgement). Kun lähettäjä lähettää segmentin, se käynnistää ajastimen, jonka aika perustuu konservatiiviseen arvioon segmentin edestakaiseen matkaan kuluvasta ajasta. Mikäli lähetystä ei kuitata ajastimen loppuun mennessä, suoritetaan uudelleenlähetys. (RFC 793., sivu 77)

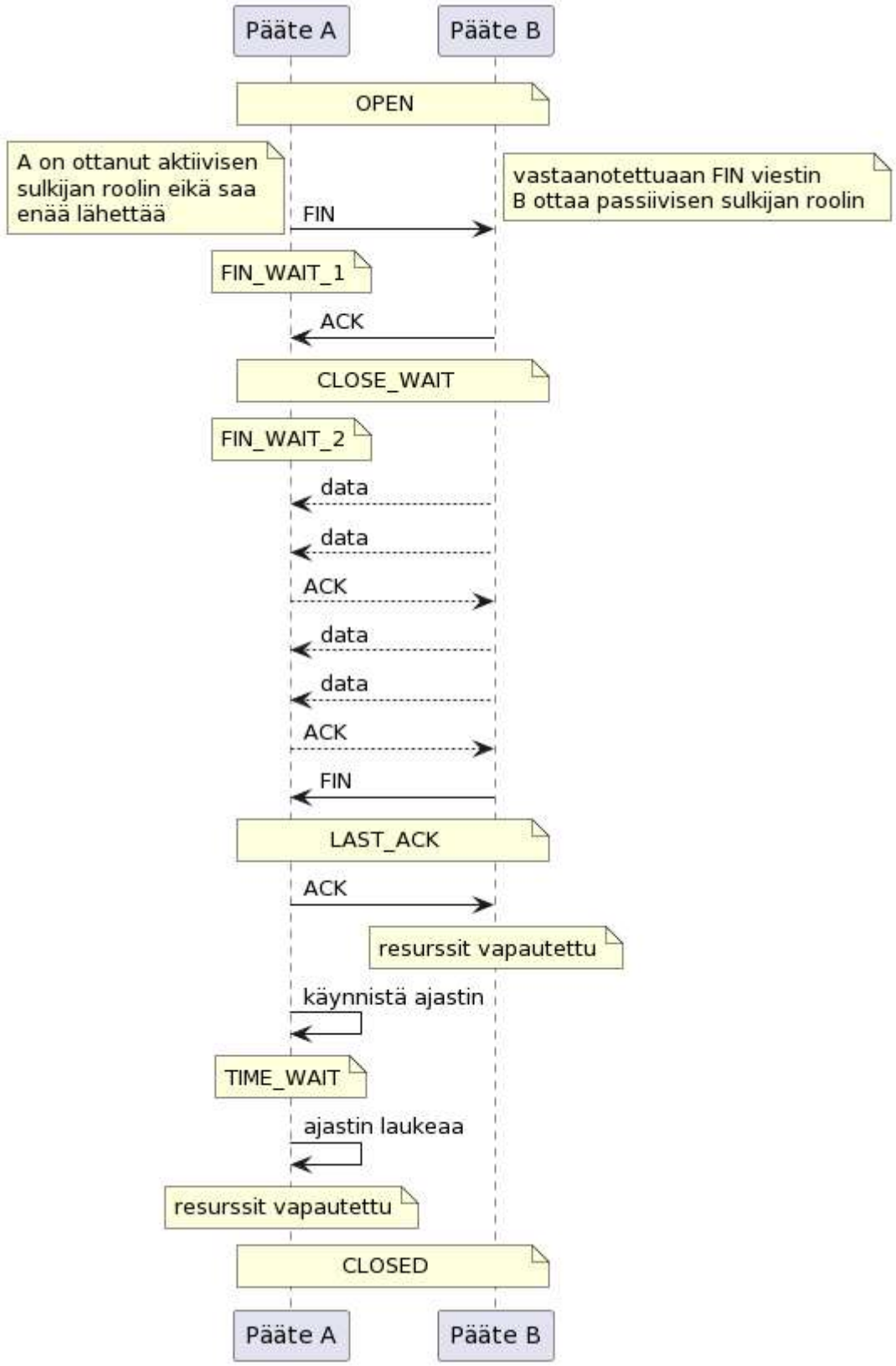
DupAck-mekanismi perustuu kuittausnumeroihin ja sekvenssinumeroiden seuraamiseen. Koska kuittausnumerot ovat kumulatiivisia, lähtökohtaisesti vastaanottaja voi kuitata vain yhtäjaksoisesti oikein vastaanotetut segmentit. Jos segmentit 1, 2 ja 3 saapuvat järjestyksessä perille, 4 ja 5 eksyvät matkalla ja seuraava vastaanotettu segmentti onkin sekvenssinumero 6, vastaanottaja voi kuitata vastaanotetuksi ainoastaan paketit 1, 2 ja 3. Jos lähettäjä huomaa tällaisen käytöksen toistuvan tietyn rajan yli – oletusarvo on kolme duplikattikuittausta – se tulkitsee kaikki viimeisintä kuittausta seuraavat segmentit kadonneiksi ja lähettää ne uudelleen. (RFC 5681, kappale 3.2.)

Datan lähetys ja kuittaus etenee siis seuraavasti:

1. pääte A lähettää haluamansa määrän, kuitenkin maksimissaan vastaanottoikkunan verran dataa. Payload datan määrä on *k* tavua, ja segmentin sekvenssinumero on *n*. Segmentin otsikon ACK lippu on ylhäällä, ja kuittausnumero on *m*
2. Pääte B vastaanottaa datan ja tarkistaa sen oikeellisuuden. Jos tarkistussummat ja sekvenssinumero täsmäävät odotettuihin, vastaanottaja lähettää ACK viestin, jonka sekvenssinumero on *m* ja kuittausnumero puolestaan $n + k + 1$
3. Lähettäjä jatkaa datan lähettämistä, kunnes joko vastaanottoikkuna täyttyy, kuittaus jää tekemättä, vastaanottajalta tulee duplikaattikuittaus tai kaikki data on lähetetty. Vastaanottoikkunan täyttyessä lähettäjä odottaa, kunnes kaikki siihen asti lähetetyt segmentit on kuitattu ennen uusien segmenttien lähettämistä. Jos taas jonkin segmentin kohdalla kuittausta ei

kuulu tietyn ajan kuluessa, tai vastaanottajalta tulee toistuvasti duplikaattikuittauksia, aloitetaan uudelleenlähetys viimeksi kuitattua segmenttiä seuraavasta segmentistä.

7.4.3 Yhteyden purku



KUVA 12: TCP-yhteyden purku

TCP on yhteydellinen tiedonsiirtoprotokolla, joten yhteyttä purettaessa se suljetaan molemmista päistä erikseen noudattaen nelivaiheista kättelyä. Kuvassa 12 on kuvattu yhteyden purun vaiheet.

1. Yhteyden päättämistä haluava osapuoli A lähettää yhteyden toiselle osapuolelle B FIN-viestin merkkinä siitä, että A ei aio enää lähettää payload dataa.
2. B kuittaa saamansa FIN viestin. Tässä vaiheessa A ei voi enää lähettää dataa, mutta B voi halutessaan lähettää.
3. Kun B on lähettänyt kaiken tarvitsemansa, se lähettää A:lle FIN viestin kertoakseen, että kaikki data on lähetetty.
4. A lähettää viimeisen kuittauksen. Tätä viimeistä kuittausta ei kuitata erikseen, vaan A käynnistää ajastimen, jonka aikana se ei hyväksy uusia yhteyksiä. Ajastimen lauettua resurssit vapautetaan ja A voi vastaanottaa uusia yhteyksiä.
5. Kun B vastaanottaa viimeisen kuittauksen, se tulkitsee yhteyden täysin suljetuksi, vapauttaa resurssit ja on täten valmis ottamaan vastaan uusia yhteyksiä.

Kuvassa 12 on myös esitetty yhteyden tilat purkuprosessin eri vaiheissa:

1. OPEN: Avoin yhteys. Molemmat puolet voivat lähettää ja vastaanottaa.
2. FIN_WAIT_1: Ensimmäisen FIN viestin lähettäjä siirtyy tähän tilaan lähetettyään ensimmäisen FIN viestin ja otettuaan täten aktiivisen sulkijan roolin.
3. CLOSE_WAIT: ensimmäinen FIN viesti on kuitattu ja yhteys on enää puoliksi auki.
4. FIN_WAIT_2: A on vastaanottanut kuittauksen FIN viestistään ja tietää että sulkupyyntö on hyväksytty. A voi vielä vastaanottaa dataa päätteeltä B.
5. LAST_ACK: B on lähettänyt kaiken datansa ja lähettää FIN viestin merkiksi siitä, että lisää dataa ei ole tulossa. Tämä viimeinen ACK viesti myös viestii yhteyden lopullisesta katkeamisesta.
6. TIME_WAIT: A lähettää vielä viimeisen ACK viestin kuitatakseen B:n FIN viestin vastaanotetuksi. Tätä viimeistä viestiä ei kuitata, joten sen lähettäjä

käynnistää ajastimen, jonka elossa olon aikana A on vielä valmis uudelleenlähettämään edellä mainitun ACK viestin.

7. CLOSED: Yhteys on täysin suljettu molemmista päistä, ja resurssit on vapautettu uusien yhteyksien käyttöön. A siirtyy tähän tilaan vasta TIME_WAIT tilassa käynnistetyn ajastimen lauettua, B voi puolestaan kohdella yhteyttä täysin katkenneena heti vastattuaan kuittauksen omaan FIN viestiinsä.

(Dostalek, L., kappale 9.3.2)

7.5 TCP:n vastuut

Yllä kuvattiin TCP-protokollan keskeistä toiminnallisuutta: yhteyden muodostusta ja purkua, sekä datan siirtoa. Tässä kappaleessa kuvataan lyhyesti jäljellä olevat vastuualueet – segmentointi, vuonhallinta ja virheenkorjaus – ja mekanismeihin, joilla nämä vastuut täytetään.

7.5.1 Segmentointi ja vuonhallinta

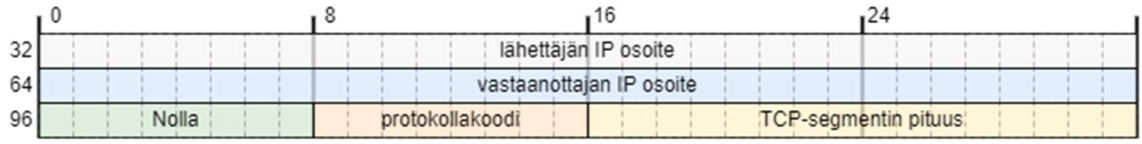
Segmentointi toteutetaan segmenttien sekvenssinumeroilla, jotka helpottavat kokoamaan lähetetyn datan oikeaan järjestykseen vastaanottopäässä. Vuonhallinta toteutetaan Window size -kentällä ja sitä skaalaavalla window scaling -asetuksella. Mikäli paketteja alkaa pudota, voi kumpi pää tahansa muokata näitä arvoja lähettämissään segmenteissä, ja näin tarpeen mukaan hidastaa tai nopeuttaa datavirtaa. (RFC 793., kappale 9.6.)

7.5.2 Virheenkorjaus

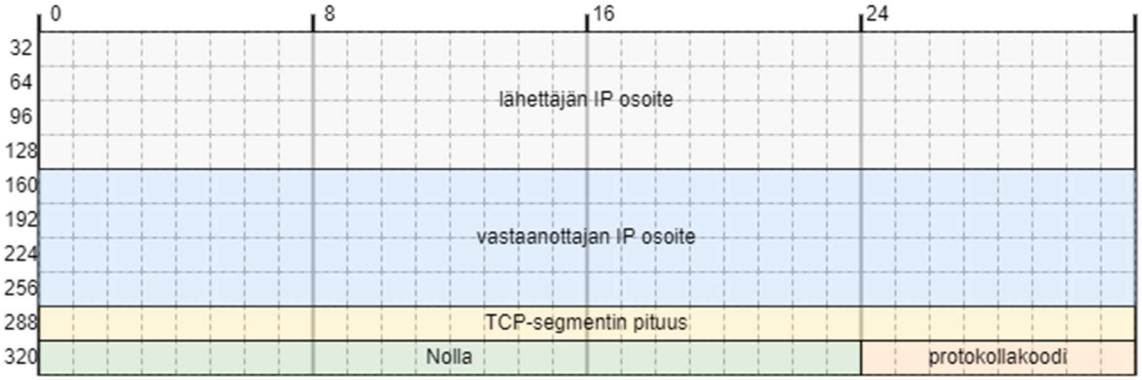
Virheenkorjaus toteutetaan tarkistussumman ja kuittausten yhdistelmänä: Lähettäjä laskee jokaiselle segmentille tarkistussumman ennen lähetystä, ja vastaanottaja tekee tämän laskutoimituksen toistamiseen vastaanotettuaan segmentin. Jos vastaanottajan ja lähettäjän tarkastussummat täsmäävät, segmentti on sel-

viytynyt matkasta virheettä. Jos taas näin ei ole, voidaan lähettää duplikaattikuittaus, tai jättää kuittaus kokonaan lähettämättä, jolloin yhteyden toinen osapuoli tietää lähettää vaurioituneen segmentin uudelleen.

Tarkistussumman laskemisessa käytetään apuna niin kutsuttua pseudo-otsikkoa, joka sisältää joitain kenttiä IP kerroksen otsikosta. Kuvissa 13 ja 14 on esitetty pseudo-otsikot IPv4 sekä IPv6 osoitteille. (Eddy, W. 2022., s. 9; RFC 8200, luku 8.1.)



KUVA 13: pseudo-otsikko IPV4-osoitteille



KUVA 14: pseudo-otsikko IPV6-osoitteille

Tarkistussumman lasku on määritelty IETF:n TCP spesifikaatiossa ykkösen komplementtina pseudo-otsikon ja segmentin muodostavien 16-bittisten sanojen ykkösen komplementtisummasta. Toisin sanoen, kokonaisuus, jossa pseudo-otsikko on asetettu segmentin eteen, pilkotaan ensiksi 16 bitin mittaisiin sanoihin ja nämä sanat summataan yhteen käyttäen ykkösen komplementtiaritmetiikkaa. Lopuksi tästä summasta otetaan ykkösen komplementti. Tarkistussummaa laskettaessa segmentin tarkistussumma-kenttä täytetään nollilla. (Eddy, W. 2022., sivu 9.)

Pseudo-otsikko rakennetaan paikallisesti jokaisen segmentin kohdalla, ja se hylätään tarkistuskoodin laskemisen jälkeen. IP-osoitteet saadaan aiemmalta protokollakerrokselta, ja protokollakoodi on TCP:n kohdalla 6. Segmentin pituuteen

lasketaan molemmat TCP-otsikko ja payload, ja se lasketaan jokaisen käsiteltävän segmentin kohdalla erikseen. (Vinicius, F. 2021.)

7.6 TCP:n haavoittuvuudet ja huomionarvoiset pulmat

TCP-protokollaan kohdistuu useita tietoturvauhkia, ja näistä tulee olla tietoisia toteutusta rakennettaessa. Palvelunesto- ja hajautetut palvelunestohyökkäykset (DoS ja DDoS) ovat näistä yksinkertaisimpia, ja tyypillisiä esimerkkejä näistä ovat SYN-tulva ja RST-hyökkäys.

SYN-tulvassa kohdekoneeseen lähetetään SYN-segmenttejä mutta kolmivaiheista kättelyä ei koskaan suoriteta loppuun. Koska puoliavonaiset yhteydet kuluttavat tietyn määrän säilytystilaa per yhteys, lopulta saavutetaan piste, jossa kohde ei pysty hyväksymään uusia yhteyksiä muistin rajallisuuden vuoksi. RST-hyökkäyksessä puolestaan hyökkääjä kohdistaa RST-viestien tulvan kohdekoneen porttiin, ja estää näin kaiken kommunikaation tämän portin kanssa. (Myers, R, sivu 4-7.)

Palvelunestohyökkäysten lisäksi TCP-yhteyttä vastaan voidaan hyökätä myös yhteyden kaappausyrityksillä, jossa pahantahtoinen toimija on saanut selville sekvenssinumeron, jota kohdejärjestelmä seuraavaksi odottaa. Tällä tiedolla varustautuneena hyökkääjä voi rakentaa haitallista dataa sisältävän segmentin, tai kaapata yhteyden kokonaan itselleen. Myers, R. n.d., sivu 10-12.)

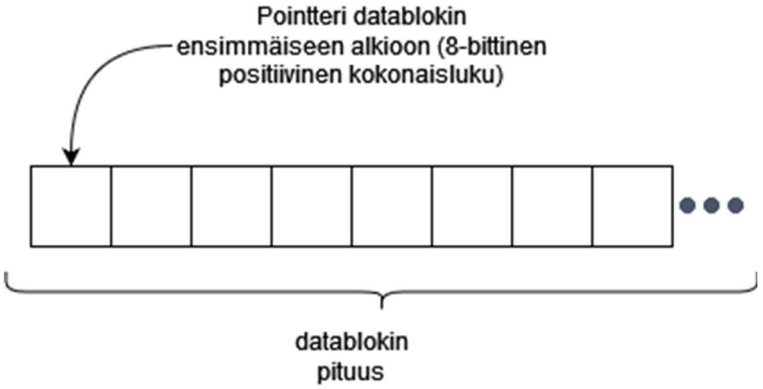
Suorien tietoturvauhkien lisäksi protokollaa toteutettaessa pitää olla tietoinen myös rajallisen lukuavaruuden aiheuttamasta ongelmasta. On olemassa esimerkiksi mahdollisuus, jossa matkalle eksynyt – sittemmin uudelleenlähetetty ja kuitattu – segmentti löytääkin tiensä perille juuri pahimpaan mahdolliseen aikaan, eli silloin kun sekvenssinumero on jo kerran pyörähtänyt ympäri. Tällaisia tapauksia varten voidaan onneksi käyttää aikaleima-asetusta, sillä vaikka kahden segmentin sekvenssinumerot olisivat identtiset, niiden aikaleimat tuskin ovat. (RFC 1323, sivu 4-5, sivu 17-24.)

8 PROTOKOLLAN TOTEUTUKSESTA

Työn alkuperäisenä tavoitteena oli toteuttaa TCP-protokolla C-kielellä, DPDK:n palveluita käyttäen ja yhteensopivaksi Nokian tarpeisiin. Projektin varsinaiseen koodausvaiheeseen siirryttäessä huomattiin kuitenkin, että kokonaisuus oli vielä oletettua laajuuttakin laajempi ja monimutkaisempi. Näin ollen täyttä toteutusta ei ollut millään tasolla mahdollista saavuttaa tälle työlle varatun ajan puitteissa.

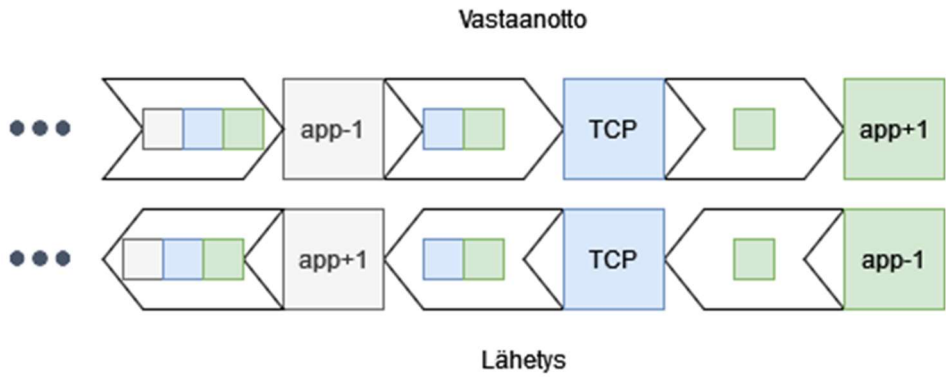
Tästä johtuen tämä kappale käsittelee TCP-protokollan käytännön toteutusta vuokaavioiden, pseudokoodin ja muiden ohjelmoinnin suunnitteluun tarkoitettujen työkalujen avulla. Hyvin pieni osa tämän kappaleen sisällöstä on jo toteutettu käytännössä, mutta niitä ei tässä työssä tarkastella. Käytännön toteutusten integroiminen olemassa olevaan koodikantaan, sekä niiden validointi, on niin aikaa vievä prosessi, että sille ei yksinkertaisesti ole aikaa.

8.1 Toteutus yleisesti



KUVA 15: datablokki

Pohjimmiltaan järjestelmässä liikutellaan yllä olevan kaltaisia datablokkeja. Datablokit toteutetaan struktuureilla, jotka pitävät sisällään pointterin datablokin ensimmäiseen alkioon, tiedon blokin pituudesta, sekvenssinumeron sekä Push- tai kiireellisyyslipun arvon. Kukin datablokki on yhden TCP-segmentin payload osuus. Datavuo on esitelty kuvassa 16.

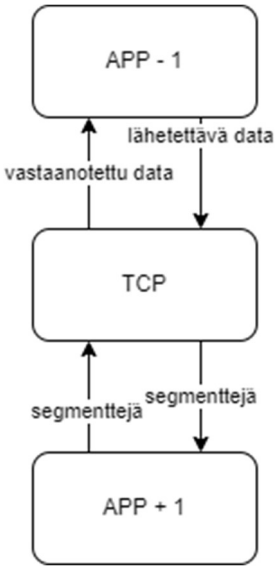


KUVA 16: Datan liikkuminen järjestelmässä karkealla tasolla.

Kuvassa esitetyt app - 1 ja app + 1 -komponentit voivat olla periaatteessa mitä tahansa ohjelmallisesti toteutettua, kunhan ne toteuttavat applikaatioiden väliset rajapinnat. Värilliset laatikot nuolien sisällä kuvaavat siirrettävän datan pituutta sen läpäistessä eri applikaatioita, ja applikaatioita voi luonnollisesti olla enemmän kuin kuvassa esitetyt kolme.

Kun vuon jokainen ”pysäkki” on pohjimmiltaan vain ohjelmallisesti toteutettu kokonaisuus, voidaan rakentaa toimivien protokollapinojen lisäksi myös tarvittaessa täysin teoreettisia protokolla- tai applikaatioketjuja, esimerkiksi sellainen, jossa TCP syöttää dataa applikaatiokerroksen sijaan ethernet-protokollalle.

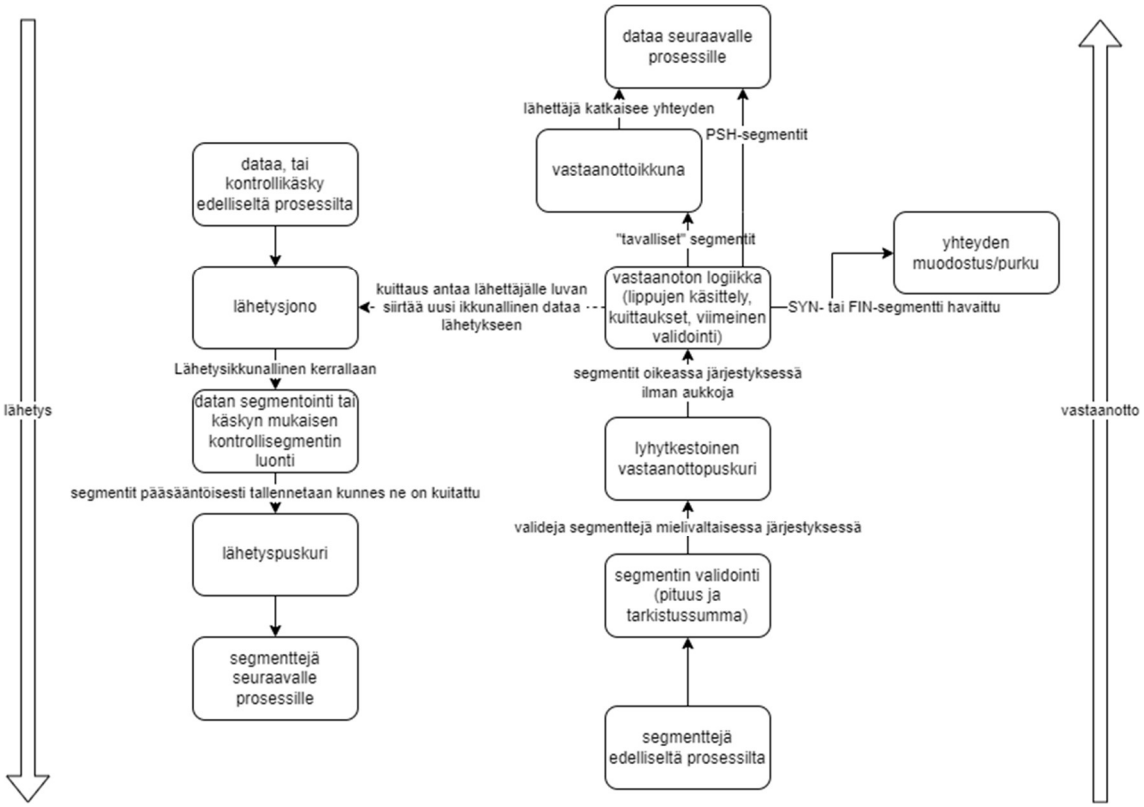
Alla olevissa kuvissa nähdään datavuo tarkemmin TCP:n näkökulmasta tarkasteltuna.



KUVA 17: Datavuo TCP:n näkökulmasta.

Kuvasta 17 nähdään, että TCP käsittelee yhtäältä kokonaisia, sille lähetettäväksi luovutettuja datakokonaisuuksia, sekä toisaalta myös segmenttejä, joita se sekä lähettää eteenpäin, että vastaanottaa.

Alla olevassa kuvassa 18 esitetään lähempi tarkastelu edellä mainitun kuvan 17 TCP-laatikkoon.



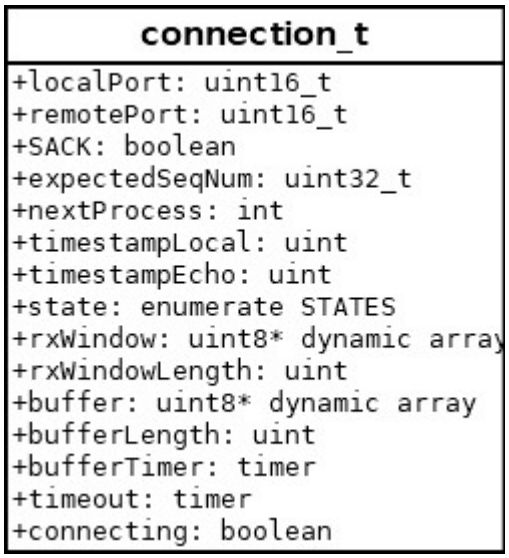
KUVA 18: Datavuo TCP:n sisällä

Kuvasta 18 nähdään TCP-kokonaisuuden yleinen toimintaperiaate: Lähettävä prosessi antaa haluamansa määrän dataa TCP:n kuljetettavaksi, datasta erotetaan lähetysikkunan suuruinen osuus lähetettäväksi, ja tämä ikkunallinen dataa pilkotaan vielä pienemmiksi, TCP-segmentteihin sopivaksi palasiksi. Nämä palaset siirretään lähetyspuskuriin, jossa ne odottavat lähetyksen jälkeen, kunnes ne kuitataan. Kuittauksen jälkeen kuitatut segmentit poistetaan puskurista ja – pois lukien tilanne, jossa uudelleenlähetystä tarvitaan – TCP:n huomaan annetusta datasta otetaan seuraava ikkunallinen lähetykseen.

Vastaanottopuolella TCP vastaanottaa segmenttejä mielivaltaisessa järjestyksessä. Nämä segmentit validoidaan ja asetetaan järjestykseen. Segmentit kuitataan ja niiden payload asetetaan lippumuuttujien tilasta riippuen joko odottamaan lähetyksen valmistumista, tai syötetään suoraan vastaanottavalle prosessille.

TCP-protokollatoteutuksen rajapintafunktioihin asetetaan pointteri datablokin ensimmäiseen alkioon, sekä struktuuri, jossa on kokonaisuuden kannalta tärkeää metadataa, kuten esimerkiksi kyseistä applikaatiota (tässä tapauksessa TCP-protokollaa) kiinnostavan datan kokonaispituus. Rajapintafunktiot suunnitellaan siten, että ne voidaan asettaa suorittavassa prosessissa peräkkäin: Kunkin applikaation sisällä muokataan rajapintafunktioon saatua datakokonaisuutta ennen kuin se siirtyy ”putkessa” eteenpäin seuraavan applikaation rajapintafunktioon. Lopulta putken yhdestä päästä syötetty data tulee ulos putken toisesta päästä, ja protokollapinon kokonaisvaikutus dataan on nähtävissä.

8.2 Yhteyksien hallinta



KUVA 19: yhteyden tiedot säilövä struktuuri

Kaikki yksittäiseen yhteyteen liittyvä data tallennetaan struktuuriin, jonka suunniteltu sisältö on esitetty kuvassa 19. Nämä struktuurit puolestaan tallennetaan assosiatiiviseen taulukkoon, jonka avaimena toimii porttinumeroista muodostettu tunnuskoodi. Viestin saapuessa voidaan nopeasti määrittää mihin kyseisen segmentin payload täytyy suunnata, onko segmentin sekvenssinumero oikein ja mitä

viestin vastaanottamiseen vaikuttavia asetuksia on voimassa. Lähettävän puolen on helppo tarkistaa struktuurista esimerkiksi yhteyden toisen pään mainostama vastaanottoikkunan koko. Yhteyksiä on lisäksi helppo avata ja sulkea lisäämällä tai poistamalla niitä taulukosta.

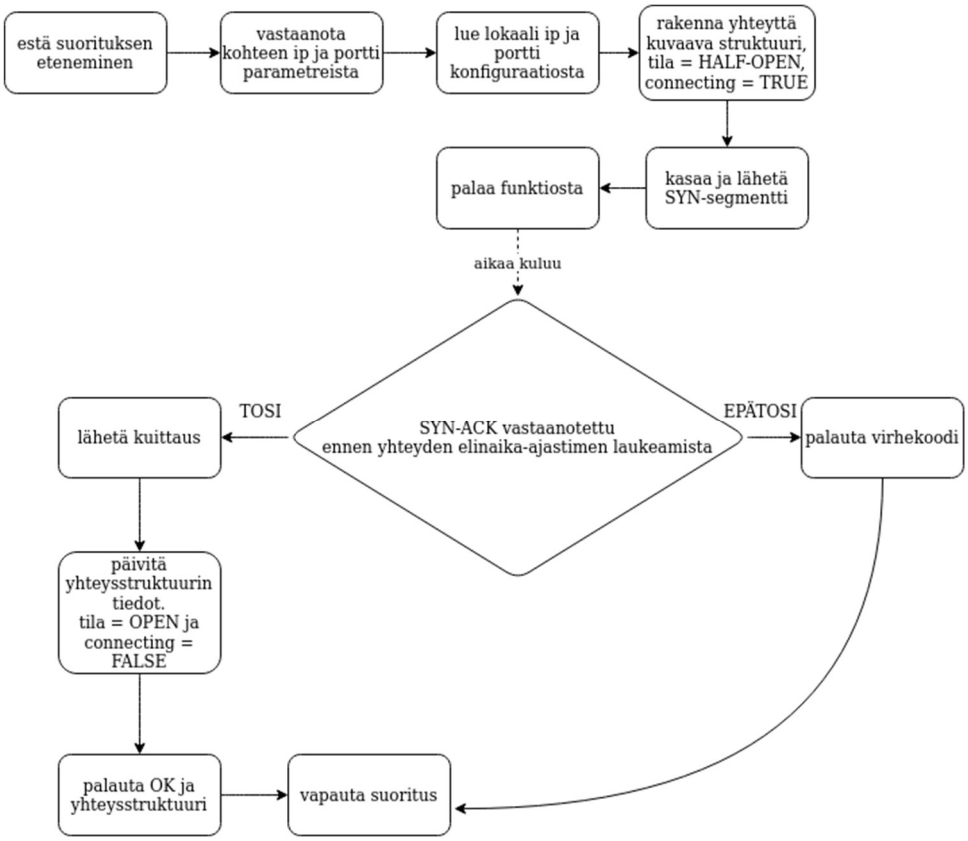
State-muuttujan arvot vastaavat pääosin kappaleessa 7.4.3 eriteltyjä yhteyden tiloja. Kun ensimmäinen SYN-paketti lähetetään, lähettäjä lisää yhteyttä vastaavan struktuurin taulukkoon ja asettaa tilaksi HALF-OPEN. Yhteys siirtyy tilaan OPEN, kun kolmivaiheinen kättely on suoritettu, ja HALF-CLOSE vastaa tilannetta, jossa aktiivinen sulkija on lähettänyt FIN-segmentin, mutta passiivinen sulkija ei. Enumeroidulla muuttujalla ei ole erillistä CLOSED-tilaa, vaan tapauksissa, jossa yhteys on täysin katkaistu – esimerkiksi sulkuproseduurin aikana tai kun RST segmentti on lähetetty tai vastaanotettu – yhteyttä edustava alkio yksinkertaisesti poistetaan taulukosta.

8.3 Yhteyden muodostaminen

Yhteyden muodostumisen toteutus seuraa kolmivaiheisen kättelyn algoritmia. Rajapintafunktioon annetaan yhdistettävän kohteen IP-osoite sekä portti, johon halutaan yhdistää. Yhteyden puuttuvat tiedot, eli lokaali IP-osoite ja portti saadaan jotain muuta kautta. Tämä mekanismi ei tämän työn kirjoitusvaiheessa ollut aivan selvää. Yhteyttä muodostava funktio pyritään saamaan toimimaan siten, että se palauttaa yhteyttä edustavan struktuurin. Tätä palautettua struktuuria voi sen jälkeen käyttää hyväksi lähetys- ja yhteyden purkufunktion parametreina.

Yhteyden muodostavan funktion tulee myös toimia blokkaavana sitä kutsuvan prosessin näkökulmasta. On tärkeää, että TCP:tä hyödyntävä prosessi ei etene yhdistämisen ohi ennen kuin yhdistysprosessista on saatu jokin lopputulos, oli se sitten muodostettu yhteys tai virhekoodi. Tarkat mekanismit tämän tavoitteen saavuttamiseksi ovat vielä jokseenkin hämärän peitossa, mutta Nokian kollegan kanssa käytyjen keskustelujen perusteella ongelma voidaan kiertää sijoittamalla TCP-proseduurien ja TCP:tä hyödyntävien applikaatioiden väliin rajapinta, joka piilottaa TCP:n sisäiset funktiot applikaatiolta. Tarkka toteutus on kirjoitushetkellä epäselvä.

Alla olevassa kuvassa esitetään yhteyden muodostamiseen käytetty proseduuri siinä määrin, kun sitä on kirjoitushetkellä suunniteltu.



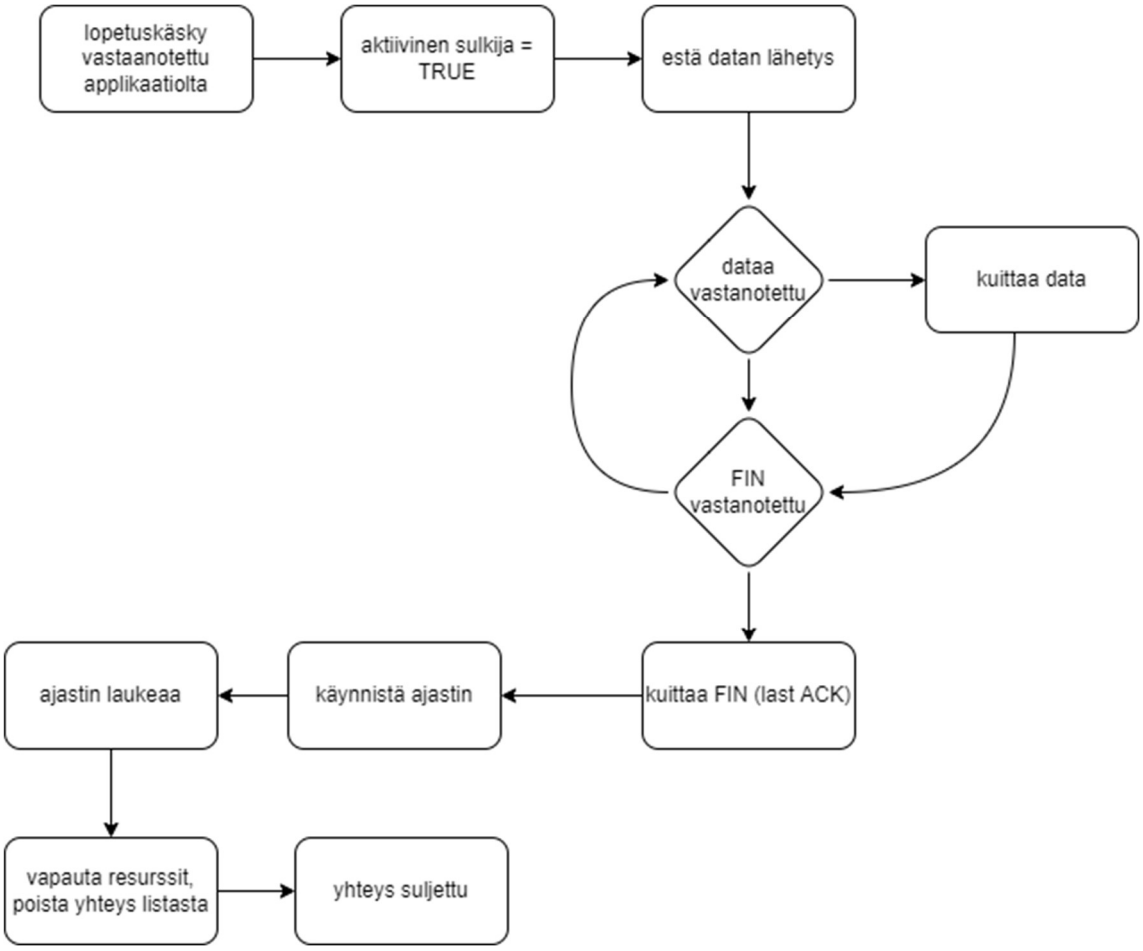
KUVA 20: suunniteltu yhteydenmuodostusproseduuri

Virhetilanteessa funktion palauttama struktuuri ei edusta toimivaa yhteyttä ja tästä informoidaan funktion kutsujaa palauttamalla virhekoodi funktion paluuarvona. Yhdistämisen onnistuessa asiasta kerrotaan niin ikään paluuarvon avulla. Kummassakin tapauksessa struktuuri otetaan viitemuuttujana funktioon.

8.4 Yhteyden purku

Yhteyden purkava proseduuri muuttuu hiukan sen perusteella kumpi yhteysosapuoli lähettää ensimmäisen FIN-segmentin. Tästä aktiivisen sulkijan roolista tulee pitää kirjaa yhteysstruktuurin sisällä olevalla muuttujalla.

Aluksi esitellään aktiivisen sulkijan proseduuri.



KUVA 21: yhteyden purku

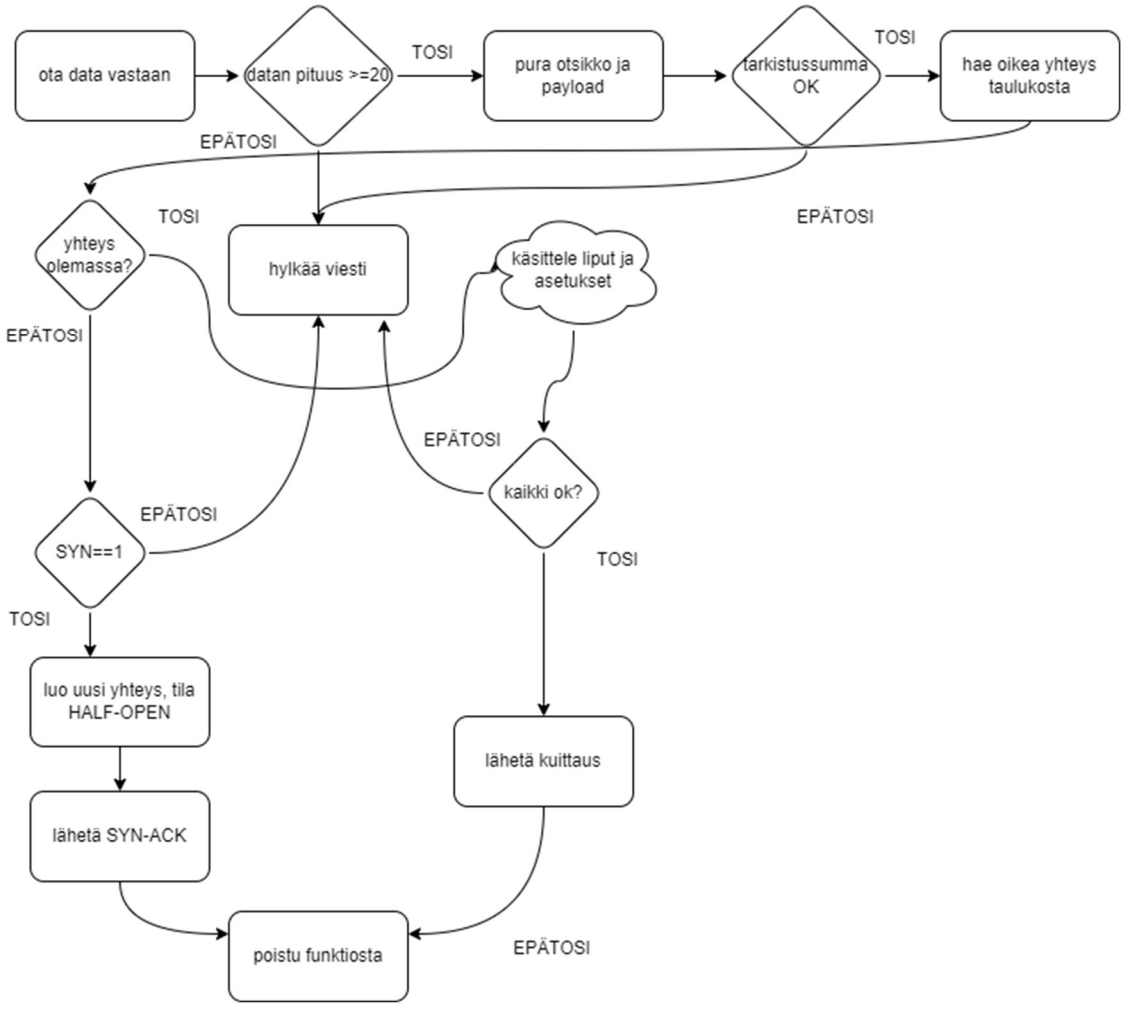
Kuvasta 21 nähdään että proseduuri seuraa kappaleessa 7.4.3 määriteltyä sulkuproseduuria. Kun sulkukomento saadaan applikaatiolta, on sen jälkeen tärkeää estää datan lähettäminen. Tämä voidaan tehdä asettamalla tarkistus mahdollisimman lähelle applikaatiokerrosta

8.5 Viestien vastaanottaminen

Viestien vastaanottamisen voi pelkistää muutamaan vaiheeseen:

1. Vastaanota pointteri dataan
2. Erota toisistaan TCP-otsikko ja payload
3. Suorita otsikon lippujen ja mahdollisten asetusten määrittelemät toimenpiteet
4. Lähetä payload-osuus eteenpäin
5. Palaa funktiosta.

Nämä vaiheet ovat kuitenkin vain äärimmäisen karkea esitys vaadittavista toimenpiteistä. Alla esitetyissä kuvissa 22 on kuvattu vastaanottoprosessin eteneminen hiukan yksityiskohtaisemmalla tasolla, joskin tämäkin kuvaus jättää vielä paljon yksityiskohtia pois.



KUVA 22: Datan vastaanotto karkealla tasolla

```
processFlagsAndOptions(payload, header, connection)
{
    if(!connecting)
    {
        if(SYN==0 && ACK==1)
        {
            if (sequenceNumber == expectedSequenceNumber)
            {
                sequenceNumberOK=true;
            }
            if(sequenceNumberOK)
            {
                if(RST=1)
                {
                    dropConnection(connection);
                }
                if(FIN==1)
                {
                    if(connection.closer==NONE)
                    {
                        connection.closer=PASSIVE;
                    }
                    else if(connection.closer==ACTIVE)
                    {
                        startCloseTimer(connection);
                    }
                    acknowledge(connection);
                    return;
                }
                if(URG==1)
                {
                    pushDataToNextApplication(separateUrgentData(header, payload, connection));
                }
                if(PSH==1)
                {
                    pushDataToNextApplication(payload, connection);
                }
                if(ECE==1)
                {
                    initiateCongestionControl(connection);
                }
                pushDataToRecieveWindow(connection, payload);
                acknowledge(connection);
                return;
            }
            if(RST=1 || RST==1 || FIN==1)
            {
                return;
            }

            if(URG==1)
            {
                pushDataToBuffer(separateUrgentData(header, payload, connection));
            }
            if(PSH==1)
            {
                pushDataToBuffer(payload, connection);
            }
        }
    }
}
```

KUVA 23: Lippujen käsittely pseudokoodina

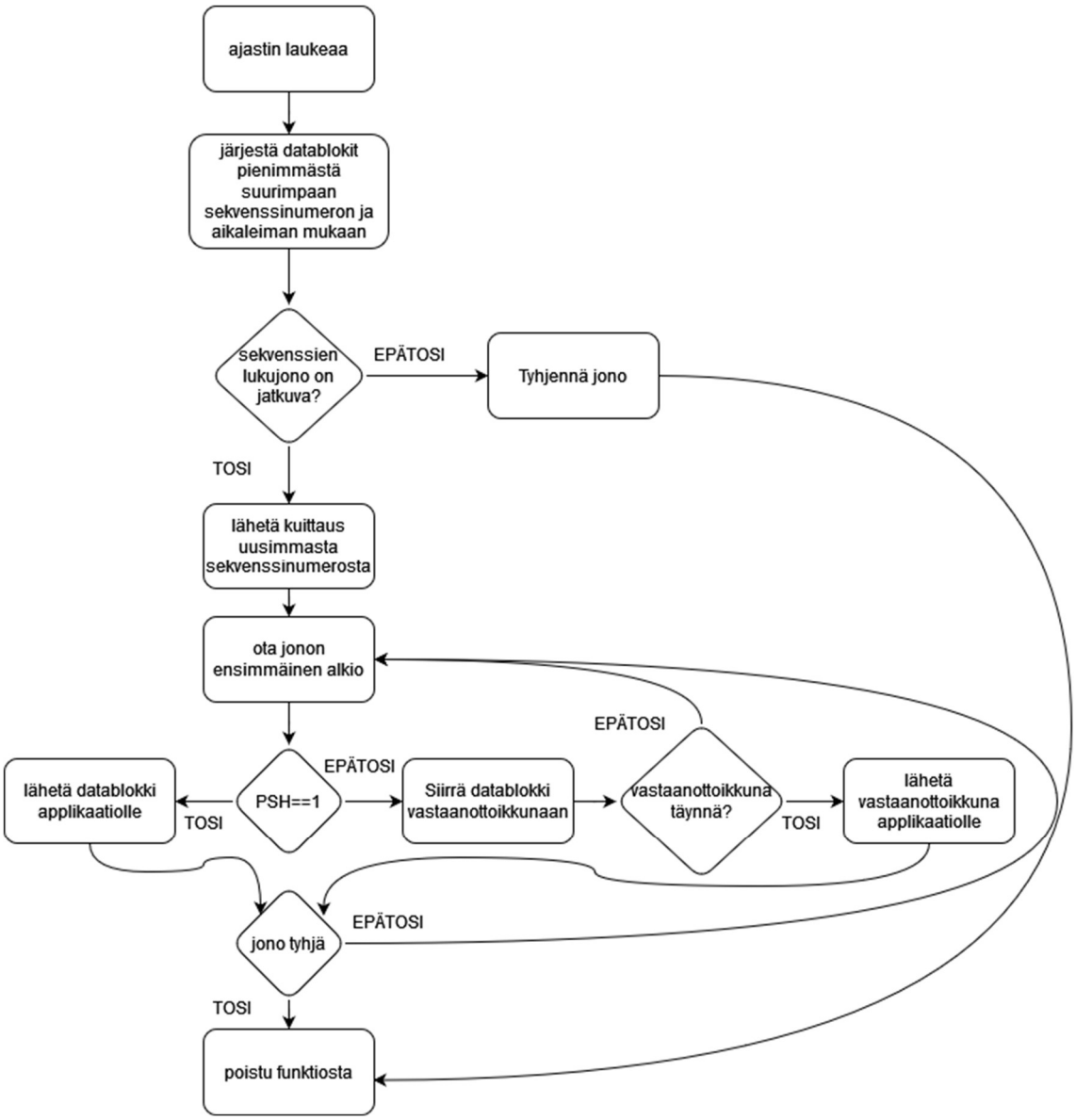
Yllä olevassa pseudokoodilistauksessa (kuva N) on esitetty ohjelman toiminta silloin kun yhteys on jo muodostettu. Pohjimmiltaan tämä ohjelma etenee melko yksinkertaisesti: Ensiksi tarkistetaan, täsmääkö vastaanotetun segmentin sekvenssinumero odotettuun sekvenssinumeroarvoon, ja vastaus tähän kysymykseen jakaa ohjelman suorituksen kahteen haaraan. Jos sekvenssinumero täsmää, segmentin payload työnnetään ketjun seuraavalle prosessille, tai säilötään vastaanottoikkunaan koko kokonaisuuden edelleen lähetystä varten, riippuen

PSH ja URG lippujen tilasta. Mikäli sekvenssinumero ei täsmää, siirretään payload sen sijaan lyhytkestoiseen puskuriin (buffer). Tämän puskurin tarkoitus on antaa hiukan armonaikaa siltä varalta, että kaikki segmentit saapuvat perille jotakuinkin samoihin aikoihin, mutta väärässä järjestyksessä.

Sekä vastaanottoikkuna että lyhytaikainen puskuri ovat dynaamisia taulukoita, jotka sisältävät alkioinaan datablokkeja, ja joiden toteutuksessa käytetään struktuureja. Näin voidaan helpommin pitää kirjaa taulukoiden pituudesta ja tarvittaessa asettaa taulukolle maksimipituus, joka on muutettavissa ajon aikana. Vastaanottoikkunan ilmoitettu maksimikoko määrittelee maksimimäärän dataa, jota paikallinen TCP-prosessi suostuu käsittelemään, ja sitä voidaan dynaamisesti muokata riippuen tilanteesta. Tällaisia tilanteita voivat olla esimerkiksi segmenttien katoaminen, jolloin vastaanottoikkunaa pienennetään tiheämpien kuittausten toivossa.

Lyhytaikaisella puskurilla ei ole ilmoitettua maksimikokoa, mutta käytännön tasolla sen koko ei voi kasvaa loputtomiin. Tämä kokorajoite johtuu siitä, että kun ensimmäinen datablokki asetetaan puskuriin, käynnistyy kyseistä puskuria koskeva, muutaman sadan millisekunnin mittainen ajastin, jonka elossa olon aikana puskuriin hyväksytään uusia alkioita. Nämä uudet alkiot eivät käynnistä uutta ajastinta ja vain yksi ajastin saa olla käynnissä kerrallaan.

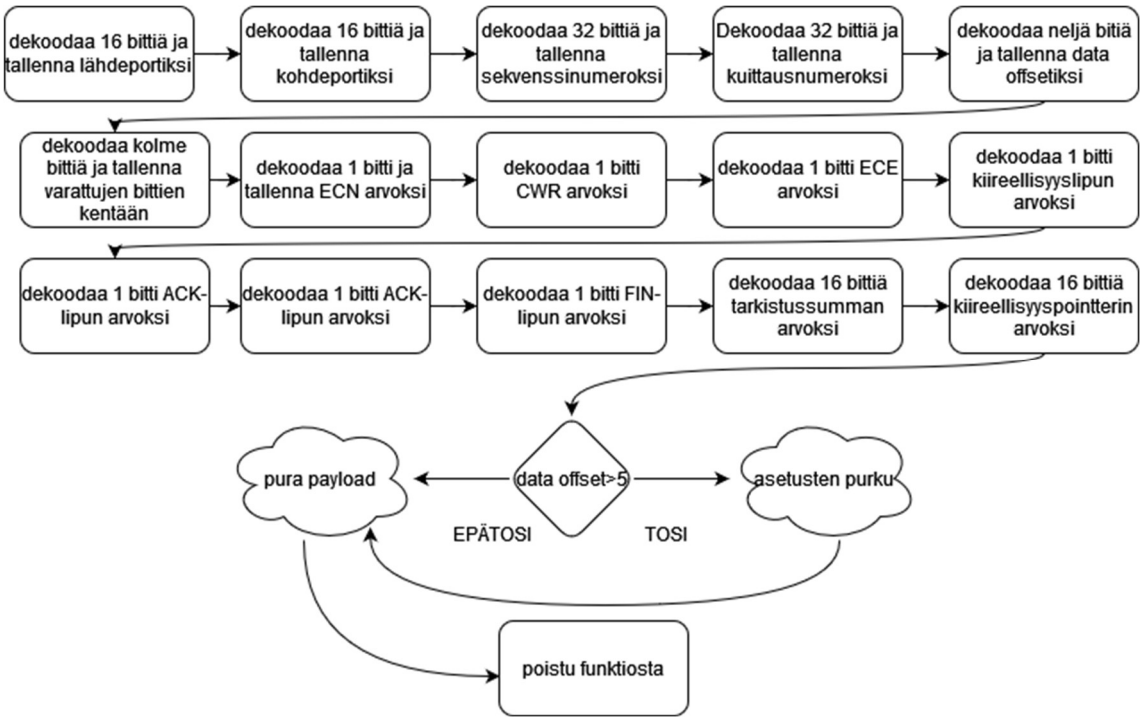
Kuvassa 24 on kuvattu mitä tapahtuu ajastimen lauettua.



Kuva 24: puskurin operaatiot

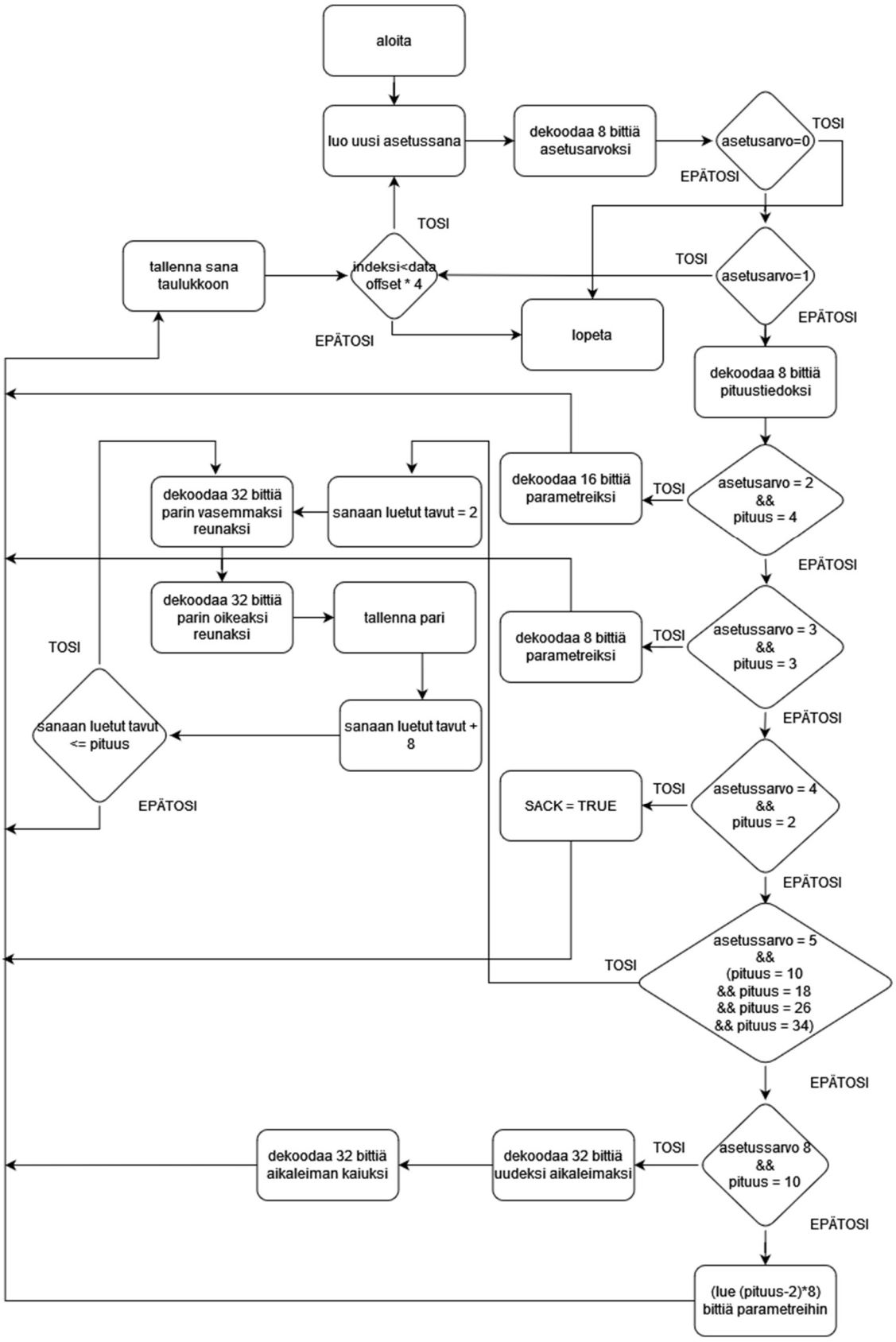
Ajastimen lauettua järjestetään datablokit niiden sekvenssinumeroiden mukaan, ja tarkistetaan, onko sekvenssinumerojono jatkuva. Kummassakin tulee luonnollisesti muistaa 32-bittisen lukualueen ylivuotomahdollisuus. Kun nämä kaksi seikkaa on varmistettu, voidaan silmukoida puskuri läpi, ja syöttää sen alkiot PSH-lipun perusteella joko suoraan applikaatiolle tai kyseisen applikaation vastaanottoikkunaan. Jos jonosta puuttuu sekvenssinumeroita ajastimen laukeamisen ja jonon järjestämisen jälkeen, voidaan koko lähetyksen todeta menneen pieleen ja tyhjentää tämä lyhytaikainen puskuri.

Kuvassa 25 on kuvattu TCP-otsikon purkamiseen käytetty proseduuri.



KUVA 25: TCP-otsikon erottaminen datasta

TCP-otsikko säilötään struktuurissa, jossa on bitfield-kentät jokaista kappaleessa 7.2 eritellyn TCP-otsikon kenttää kohden. Valinnaiset asetukset säilötään puolestaan ”sanoina”. Kukin sana on struktuuri, joka sisältää asetuskoodin sekä pituuden, ja kahdeksanpaikkaisen taulukon mahdollisia parametreja varten.



KUVA 26: Asetusten erottaminen TCP-otsikosta

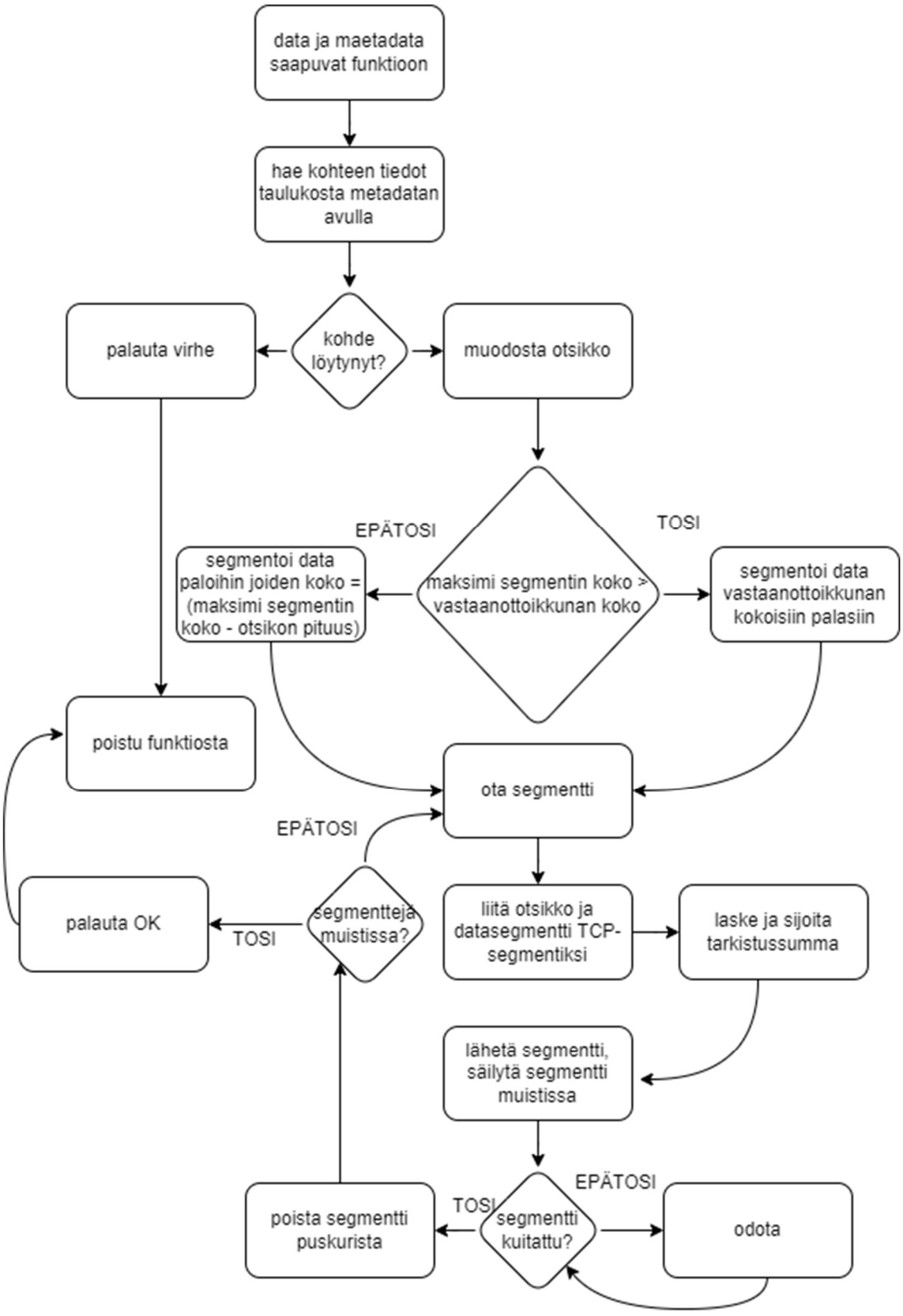
Kuvassa 26 kuvattu funktio näyttää sekavalta, mutta on pohjimmillaan yksinkertainen: data offset kenttää hyödyntämällä saadaan selville otsikon kokonaispituus, joten funktiossa voidaan silmukoida otsikko loppuun asti hyvin helposti asetus kerrallaan. Asetuskoodista taas voidaan päätellä mitä asetuksen pituuden tulee olla, ja pituudesta tiedetään parametrien pituus. Jokainen sana säilötään omaan struktuuriinsa, jotka puolestaan kootaan yhteen dynaamiseen taulukkoon ja tallennetaan osaksi TCP-otsikon säilövää struktuuria. Alla olevassa taulukossa nähdään kunkin asetuskoodin vaikutus tietorakenteisiin ja/tai datan käsittelyyn.

Koodi	Vaikutus
0	Ei vaikutusta
1	Ei vaikutusta
2	Aseta yhteysstruktuurin maximumOutgoingSegment datajäseneen haluttu arvo
3	Laske uusi ikkunan koko ja sijoita se yhteysstruktuuriin. Uusi arvo lasketaan left shiftaamalla window size -kentän arvoa tämän asetuksen parametrien verran.
4	Aseta yhteysstruktuurin SACK kenttä arvoon TOSI
5	Lisää parametrit yhteysstruktuurin sackValues-kenttään
8	Aseta asetuksen parametrien ensimmäiset neljä tavua yhteysstruktuurin timestampEcho kenttään

TAULUKKO 2: asetuskoodit ja niiden vaikutus toteutuksessa

8.6 Datan lähetyksestä

Datan lähetys toimii pitkälti samoin kuin vastaanotto, ainakin arkkitehtuurin näkökulmasta. TCP-kokonaisuuden lähetysrajapintafunktioon saapuu parametreina lähetettävä data, samoin kuin metadata, jossa on lähettämiseen ja osa otsikon kokoamiseen tarvittavista tiedoista. Alla on esitetty datan lähetysproseduuri pääpiirteittäin.



KUVA 27: datan lähetys

Dataa lähetettäessä on tärkeää, että lähetetyt segmentit pidetään muistissa kuittaamista varten. Vasta kun yksittäinen segmentti on kuitattu, voidaan se – ja sitä edeltävät segmentit – poistaa muistista.

Lähettäjän tulee pitää kirjaa lähetetyn datan määrästä siinä tapauksessa, jos maksimi segmentin koko jää vastaanottoikkunaa pienemmäksi. Tällaisessa tapauksessa jokaiselle segmentille ei tarvitse odottaa erikseen kuittausta, vaan lähettäjä voi lähettää segmenttejä ketjussa vastaanottoikkunan täyttymiseen asti ja vasta sitten pysähtyä odottamaan kuittausta.

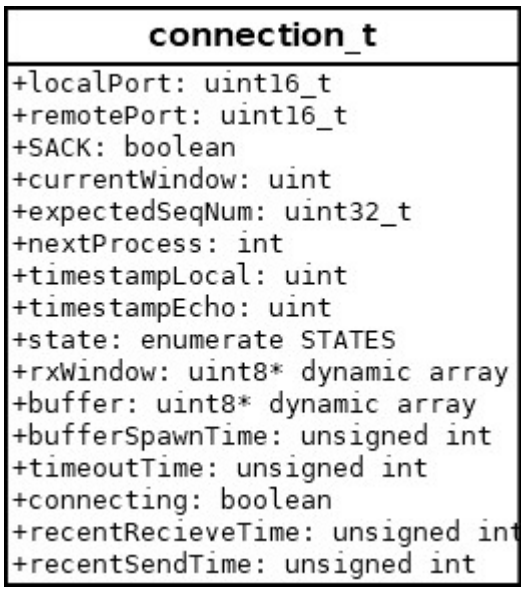
Lähetyspuolen rajapintafunktioita rakennettaessa pyritään mahdollisimman suureen samankaltaisuuteen olemassa olevien TCP-toteutusten kanssa. Tämä tarkoittaa sitä, että lähetysfunktion argumenttilista pyritään pitämään mahdollisimman lyhyenä. Toteutuksessa tarkoitus on päästä tilanteeseen, jossa lähetettävän datan ja datan pituuden lisäksi, funktio tarvitsee vain yhteyttä edustavan struktuurin toimiakseen oikein.

8.7 Ajastimien toteutuksesta

Protokollan toteutusta koskevia kappaleita tarkastelemalla huomataan, että se vaatii useita ajastimia toimiakseen spesifikaation mukaisesti: Yhteysosapuolten täytyy pitää kirjaa lyhytaikaisen puskurin elinajasta, koko yhteyden elinajasta, sekä uudelleen lähetyksen ajastimesta. Erilliset ajastimet kuitenkin syövät prosessointiaikaa, ja tämän työn tarkoituksena on nimenomaan suunnitella protokollan toteutus siten, että siihen käytetty prosessointiaika saadaan minimoitua.

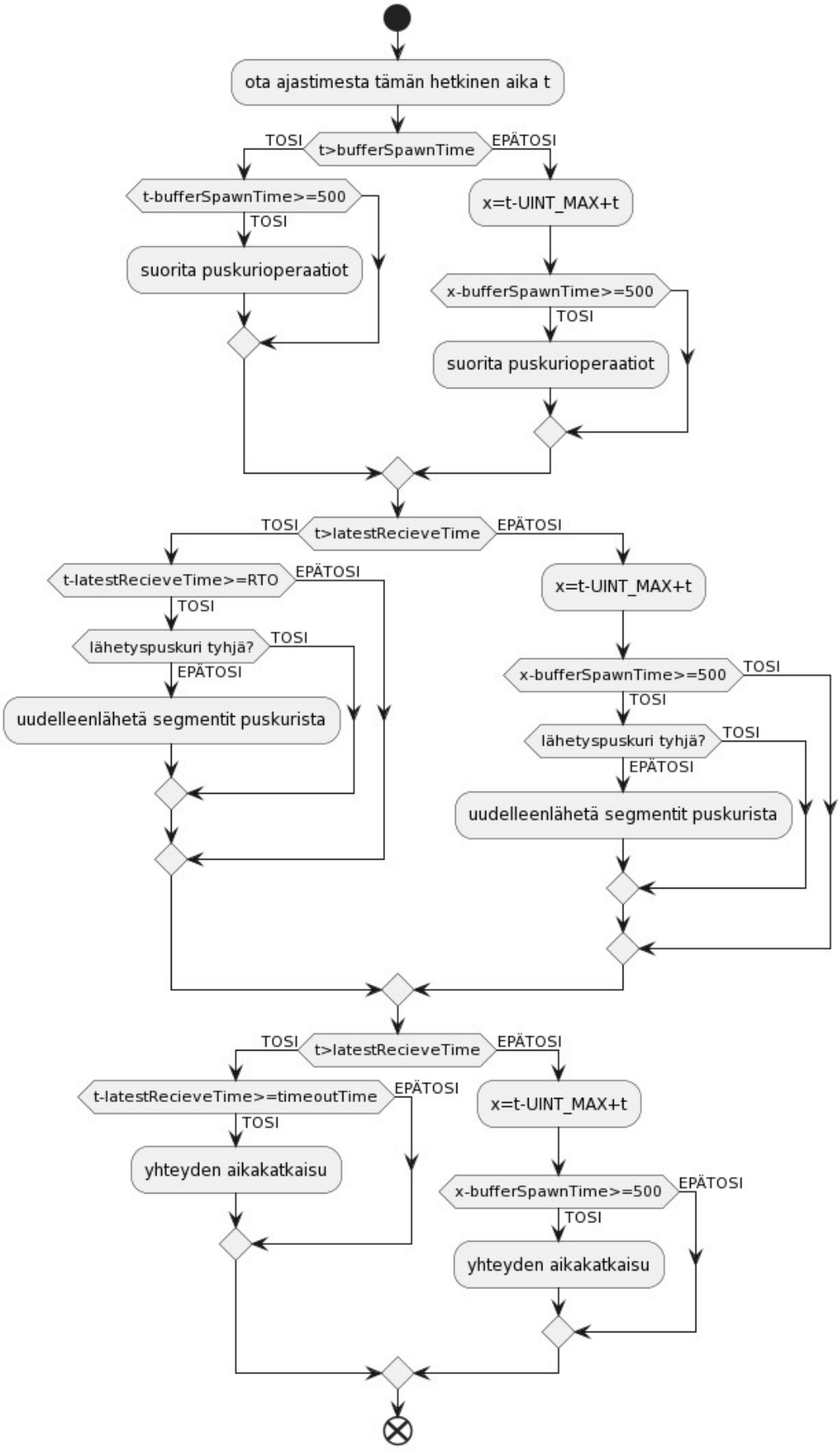
Tavoitteen saavuttamiseksi erilliset ajastimet yhdistetään yhdeksi, tiheään laukeavaksi ajastimeksi, joka käynnistää määritellyn proseduurin ajan kuluttua loppuun ja käynnistyy tämän proseduurin suorituksen päätyttyä uudestaan.

Tämä uusi suunnittelu vaatii päivityksen kuvassa 19 esiteltyyn yhteysstruktuuriin:



KUVA 28 uuteen ajastinajatteluun sopiva yhteysstruktuuri.

Ajastimeen liitetty proseduuri puolestaan on esitelty kuvassa 29.



KUVA 29: ajastimeen liitetty proseduuri

Kuvissa 29 mainittu RTO, eli **r**etransmission **t**ime**o**ut lasketaan kertomalla segmentin edestakaiseen matkaan (RTT, **r**ound **t**rip **t**ime) kuluva aika kahdella, ja ottamalla 16:sta tällaisesta arvosta keskiarvo. Keskiarvon muodostavat alkiot asetetaan vakiokoiseen rengaspuskuriin, jonka kooksi valitaan 16, sillä kahden potenssit ovat tietokoneelle helppoja jakajia. RTT saadaan yksinkertaisesti tarkistamalla aika segmentin lähetyshetkellä, ja uudestaan silloin kun siihen on saatu hyväksyttävä vastaus.

Toteuttamalla TCP-protokollan vaatimat ajastimet yllä kuvatun kaltaisesti, voidaan kaikki ajastamista vaativat toimenpiteet sitoa yhteen kelloon ja näin tehostaa ohjelmiston toimintaa. On kuitenkin oltava hyvin tietoinen yllä kuvatun proseduurin laskentavaatimuksista, ettei esimerkiksi kymmenen millisekunnin tiheydellä laukeava ajastin ehdi laukeamaan toista kertaa proseduurin suorituksen aikana, vaikka tämä proseduuri suoritettaisiin mielivaltaiselle määrälle yhteyksiä.

8.8 Toteutuksen muita yksityiskohtia

Kappaleessa 8 on tähän asti kuvailtu tarkasti suunniteltuja proseduureja TCP:n keskeisten ominaisuuksien toteuttamiseksi. Tässä kappaleessa käydään lyhyesti läpi joitain suunniteltuja tapoja sellaisten teoriaosuudessa mainittujen yksityiskohtien toteuttamiseen, joihin ei tähän mennessä ole otettu kantaa.

Tarkistussumman laskemisessa käytetään apuna kuvan 30 funktiota, joka palauttaa kahden kokonaisluvun ykkösen komplementtisumman:

```
uint16_t binarySum(uint16_t a, uint16_t b)
{
    uint16_t carry = 0;
    while(b!=0)
    {
        carry = a&b;
        a=a^b;
        b=carry<<1;
    }
    return a;
}
```

KUVA 30: kahden kokonaisluvun ykkösen komplementtisumma

Kun pseudokoodin ja segmentin kaikki 16-bittiset sanat on summattu, lopputuloksesta otetaan ykkösen komplementti bitwise NOT -operaattorilla. Laskentaan tarvittavat IP osoitteet saadaan metadatana, varsinaisen TCP-kokonaisuuden ulkopuolelta, joten erityistä proseduuria IP osoitteiden pyytämiseen ei tarvitse toteuttaa. Muutoin tarkistussumman laskemiseen käytettävä pseudo-otsikko voidaan rakentaa TCP-ohjelmiston sisäisesti.

SACK:in toteutusta ei tässä vaiheessa ole paljoa suunniteltu, mutta olemassa olevat suunnitelmat mahdollistavat sen lisäämisen melko pienin lisäyksin: Yhteysstruktuurissa on jo muuttuja sille käyttääkö yhteys SACK:ia vai ei, ja SACK:n aktivoiminen tarkoittaisi käytännössä puskurioperaatioiden (kuva N) muuttamista sellaiseksi, että epäjatkuvuutta havaittaessa lähetetäänkin kuittaus oikeilla SACK:in arvoilla, jotta lähettäjä osaa uudelleen lähettää halutut segmentit.

Aikaleima-asetuksen tarkasta toimintaperiaatteesta ei vielä ole täyttä varmuutta, mutta kaikkein yksinkertaisin ja kivuttomin ratkaisu olisi todennäköisesti liittää aikaleiman päivitys ajastinoperaatioiden (kuva 29) jatkoksi.

8.9 DPDK:n käytöstä

DPDK:n käytännön soveltaminen ei kirjoitushetkellä ole selvää, mutta on syytä olettaa, että kirjastojen käytännön merkitys selkeytyy ajettaessa tässä työssä eriteltyä suunnitelmaa käytäntöön.

Toteutettava TCP-kokonaisuus integroidaan järjestelmään missä DPDK:ta hyödynnetään jo valmiiksi laaja-alaisesti, ja DPDK:n ominaisuuksista tärkeimmiksi toteutuksen kannalta nähdään tässä vaiheessa sen mahdollistama nopea muistinkäsittely, suora yhteys verkkokortilta TCP-toteutukseen ja ytimien yksityiskohtainen hallinta. Viimeksi mainittu mahdollistaa koko järjestelmän deterministisen toiminnan, ja sitä kautta tarkemmat ajastimet, ytimien varaamisen oikeisiin tarkoituksiin ja suoritusturvallisuuden, ilman pelkoa käyttöjärjestelmän väliintulosta.

9 POHDINTA

Tämä opinnäytetyö käsitteli TCP-protokollan toimintaa, ja sen toteutuksessa huomioonotettavia seikkoja. Kuten jo aikaisemmin todettiin, aikarajoitteiden vuoksi käytännön toteutukselle ei juurikaan jäänyt aikaa, ja se vähä mitä käytännössä ehdittiin kirjoittaa ei ollut kovin keskeistä tämän työn aiheen kannalta.

Näistä seikoista huolimatta työtä tehdessä kirjoittajalla heräsi suuri mielenkiinto protokollia ja niiden toimintaa ja ohjelmointia kohtaan. Tarkka, bittikohtainen ohjelmointi segmentin sisällä on kiehtova ohjelmoinnin alalaji, jota tämän opinnäytetyön siivittämänä voidaan toivottavasti jatkaa.

Työn aikarajoitteet johtivat hankalaan tilanteeseen, jossa varsinaista käytännön todistetta TCP:n ja DPDK:n yhteen liittämisen hyödyistä ei voitu tässä dokumentissa todentaa. Lisäksi DPDK:n integroiminen kokonaisuuteen on vielä lapsenkengissään, jopa kokeneempien kollegoiden mielissä. Nämä seikat todennäköisesti selvenevät huomattavasti, kun suunniteltuja proseduureja aletaan kirjoittamaan käytännössä, mutta nykyisellään käytännön toteutuksen puuttuminen on selkeä puute. Tästä huolimatta toive on, että tästä dokumentista on apua käytännön ohjelmointia jatkettaessa.

Työn tekemisen aikana opittiin paljon TCP:n toimintamekanismeista, ja opittiin hyödyntämään RFC dokumentteja. Näistä varsinkin jälkimmäinen oli erittäin hyödyllistä, sillä ne auttavat kirjoittamaan TCP-moduulin mahdollisimman pitkälle standardien mukaiseksi. Tämän lisäksi todettiin ohjelmointityön moninaisuus: Vaikka opinnäytetyön tekemisen aikana varsinainen koodin kirjoittaminen jäi minimiin, oli tarvittavien funktioiden muodostaminen yllättävän haastavaa huolimatta siitä, että kyseessä olivat kuitenkin ”vain” vuokaavioesitykset.

On tarpeen huomauttaa, että tämä opinnäytetyö ei pidä sisällään koko TCP-protokollan spesifikaatiota. Käytetyissä lähteissä, varsinkin RFC:iden ollessa kyseessä, oli paljon yksityiskohtaisempia selontekoja kuin mitä tähän työhön olisi ollut järkevää kirjoittaa, mutta näiden dokumenttien seuraaminen tulee olemaan ensiarvoisen tärkeää käytännön toteutusta suoritettaessa.

Lähteinä työssä käytettiin runsaasti internetresursseja, joista RFC:t muodostuivat keskeisimmäksi. Lisäksi käytössä oli jonkin verran kirjallisuutta, jota hyödynnettiin e-kirja-formaatissa.

Kehitystyön seuraava askel on luonnollisesti saattaa tässä dokumentissa yksilöidyt – kuin myös tästä dokumentista pois jätetyt – toteutukset käytäntöön. Odotettavissa on myös DPDK-osuuden laajentuminen ja lukemattomien ajatus- ja logiikkavirheiden paljastuminen koodatessa. Työtä jatketaan vielä pitkään, sillä TCP-protokolla on monimutkainen kokonaisuus, jonka toteuttaminen oikein vaatii perusteellista huolellisuutta.

LÄHTEET

Burakov, A. 2019. Memory in Data Plane Development Kit Part 1: General Concepts. Intel. Verkkosivu. Viitattu 06.04.2022. https://www.intel.com/content/www/us/en/developer/articles/technical/memory-in-dpdk-part-1-general-concepts.html

Dostalek, L., Kabelova, Alena. 2006. Understanding TCP/IP: a clear and comprehensive guide to TCP/IP protocols. E-kirja. Vaatii käyttöoikeuden. Birmingham, U.K: PACKT Publishing. https://ebookcentral.proquest.com/lib/tampere/detail.action?docID=944992&pq-origsite=primo

Eddy, W. 2022. Transmission Control Protocol (TCP) Specification. IETF. Verkkosivu. Viitattu 07.04.2022. https://datatracker.ietf.org/doc/draft-ietf-tcpm-rfc793bis/

Environment Abstraction Layer. DPDK. n.d. Viitattu 06.04.2022. http://doc.dpdk.org/guides/prog_guide/env_abstraction_layer.html

Fortinet. n.d. TCP/IP model vs OSI-model. Verkkosivu. Viitattu 11.04.2022. https://www.fortinet.com/resources/cyberglossary/tcp-ip-model-vs-osi-model

Giller, R. 2016. Open vSwitch with DPDK Overview. Intel. Verkkosivu. Viitattu 06.04.2022. https://software.intel.com/content/www/us/en/develop/articles/open-vswitch-with-dpdk-overview.html

IANA. 2022. Transmission Control Protocol (TCP) Parameters. Verkkosivu. viitattu 11.04.2022 https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml

Kozierok, C. 2005 The TCP/IP guide: a comprehensive, illustrated Internet protocols reference. San Francisco: No Starch Press. viitattu 11.04.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/tcp-ip-guide/9781593270476/cover.html

Kumar, S., Dalal, S., Dixit, V. 2014. THE OSI MODEL: OVERVIEW ON THE SEVEN LAYERS OF COMPUTER NETWORKS. Julkaisun osa (PDF). Viitattu 13.04.2022. International Journal of Computer Science and Information Technology Research 2(3). https://www.researchpublish.com/upload/book/THE%20OSI%20MODEL%20OVERVIEW%20ON%20THE%20SEVEN%20LAYERS-607.pdf

Memory in data plane development kit. n.d. Intel. Verkkosivu. luettu 06.04.2022. https://www.intel.com/content/www/us/en/developer/articles/technical/memory-in-dpdk-part-1-general-concepts.html.

Myers, R. n.d. Attacks on TCP/IP Protocols. The university of tennessee chattanooga. CPSC4620: Computer Network Security. Viitattu 11.04.2022. https://www.utc.edu/sites/default/files/2021-04/course-paper-5620-attacktcpip.pdf

Peterson, L., Davie, B. 2011. Computer networks: A Systems Approach. E-Kirja. Vaatii käyttöoikeuden. San Francisco: Elsevier Science & Technology. https://learning.oreilly.com/library/view/computer-networks-5th/9780123850591/?sso_link=yes&sso_link_from=tampere-university

Poll Mode Driver. DPDK. n.d. Viitattu 14.04.2022. https://doc.dpdk.org/guides/prog_guide/poll_mode_drv.html

RFC 793. Postel, J. (toim). 1981. Transmission Control Protocol. Information Sciences Institute. University of Southern California. Standardidokumentti. Viitattu 06.04.2022. https://datatracker.ietf.org/doc/pdf/rfc793

RFC 1122: Braden, R. (toim). Requirements for Internet Hosts -- Communication Layers. IETF. Spesifikaatiodokumentti (PDF). Viitattu. 07.04.2022. https://www.rfc-editor.org/pdfrfc/rfc1122.txt.pdf

RFC 1323. Jacobson, V., Braden, R., Borman, D. TCP Extensions for High Performance. IETF. Spesifikaatiodokumentti (PDF). Viitattu 11.04.2022. https://www.rfc-editor.org/rfc/rfc1323.html

RFC 2018: Mathis, M., Mahdavi, J., Floyd, S., Romanow, A. TCP Selective Acknowledgment Options. IETF. Spesifikaatiodokumentti (PDF). Viitattu 11.04.2022. https://datatracker.ietf.org/doc/pdf/rfc2018

RFC 5681: Allman, M., Paxson, V., Blanton, E. 2009. TCP Congestion Control. IETF. Spesifikaatiodokumentti (PDF). Viitattu 11.04.2022. https://www.rfc-editor.org/rfc/rfc5681

RFC 6093: Gont, F., Yourtchenko, A. 2011. On the Implementation of the TCP Urgent Mechanism. IETF. Spesifikaatiodokumentti (PDF).
Viitattu 07.04.2022. https://www.rfc-editor.org/pdfrfc/rfc6093.txt.pdf

RFC 8200: Deering, S., Hinden, R. 2017. Internet Protocol, Version 6 (IPv6) Specification. IETF. Spesifikaatiodokumentti (PDF). Viitattu 11.04.2022. https://datatracker.ietf.org/doc/pdf/rfc8200

Saarela, J. 2021. Muistinhallinta DPDK-pohjaisessa sovelluksessa. Opinnäytetyö. Tieto- ja viestintätekniikan tutkinto-ohjelma. Tampereen ammattikorkeakoulu. Opinnäytetyö. Viitattu 11.04.2022. https://urn.fi/URN:NBN:fi:amk-202105158817

Scil100. 2010. TCP state diagram. Wikimedia commons. Kuva. https://commons.wikimedia.org/w/index.php?curid=30810617

Tanenbaum, A., Wetherall, D. 2013. Computer networks. Pearson Education. Viitattu 11.04.2022. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/computer-networks-fifth/9780133485936/xhtml/Cover.html

Thiyari, K., Pandey, H. 2022. Layers of OSI-model. GeeksForGeeks. Verkkosivu. Viitattu 06.04.2022. https://www.geeksforgeeks.org/layers-of-osi-model/

Timer Library. DPDK. n.d. Viitattu 06.04.2022. https://doc.dpdk.org/guides/prog_guide/timer_lib.html

Vinicius, F. 2021. The pseudo header in TCP. Verkkosivu. Viitattu 11.04.2022. https://www.baeldung.com/cs/pseudo-header-tcp

Wenjun, Z., Peng, L., Baozhou, L., He, X., Yujie, Z. 2018. Research and Implementation of High Performance Traffic Processing Based on Intel DPDK. Konferenssijulkaisu. Vaatii käyttöoikeuden. Luettu 06.04.2022. https://ieeexplore-ieee-org.libproxy.tuni.fi/stamp/stamp.jsp?tp=&arnumber=8701793

Zhu, H.(toim.) 2021. Data Plane Development Kit (DPDK) : a software optimization guide to the user space-based network applications. E-kirja. Boca Raton: CRC Press. Viitattu 06.04.2022. Vaatii käyttöoikeuden. https://web-s-ebscohost-com.libproxy.tuni.fi/ehost/ebookviewer/ebook/bmxlYmtfXzI1NzMxMjNfX0FO0?sid=cd9c1200-76eb-4e15-aa80-8d0a6b80da45@redis&vid=0&format=EB&lpid=lp_Cover-2&rid=0