Tampereen yliopisto

Antti-William Irva

TCP/IP-POHJAINEN SOKETTIOHJELMOINTI JAVALLA

Kandidaatin tutkielma
Informaatioteknologian ja viestinnän tiedekunta
Tarkastaja: Lehtori Juha Vihervaara
Toukokuu 2023

TIIVISTELMÄ

Antti-William Irva: TCP/IP-pohjainen sokettiohjelmointi Javalla
Kandidaatin tutkielma
Tampereen yliopisto
Tieto- ja sähkötekniikka
Toukokuu 2023

Tässä tutkielmassa perehdytään TCP/IP-pohjaiseen sokettiohjelmointiin Java-ohjelmointikieltä hyödyntäen. Työssä esitellään lyhyesti C++ ja Python -kielet sekä näiden toimintaa hajautetussa järjestelmässä, mutta keskiössä on Javalla tapahtuva sokettiohjelmointi. Lisäksi tärkeänä aiheena käsitellään TCP-, UDP-, että multicast-tyyppisten sokettien toteutus sokettiohjelmoinnin avulla. Tutkielmassa käsitellään myös lyhyesti tietoturvaa lähdekoodin ja Java Socket Security Extensionin (JSSE) kautta sekä perehdytään tämän käytettävyysongelmiin. Modernit soketteja hyödyntävät teknologiat, kuten web-sovelluksien toteuttamiseen tarkoitettu WebSocket-protokolla rajattiin tämän työn ulkopuolelle.

Tutkielmassa aineistoina toimivat aiheeseen liittyvä kirjallisuus, kuten erilaiset oppaat, artikkelit ja konferenssijulkaisut. Lisäksi keskeisenä tiedonlähteenä toimi Java-ohjelmointikielen dokumentaatio sekä eri protokollien RFC-dokumentit. Aiheeseen liittyvän aineiston esittelyn ja analysoinnin lisäksi tutkielmaan luotiin yksinkertaisia ohjelmointiesimerkkejä, jotka esitellään ohjelmakoodin sekä ohjelmien suorituksista otettujen kuvakaappausten avulla. Näin lukijalle pyrittiin esittämään, miten sokettiohjelmointia voidaan käytännössä toteuttaa ja miten erityyppisten yhteyksien käyttö eroaa toisistaan.

Työssä havaittiin, että sokettien käyttäminen Javalla on suhteellisen helppoa ja niiden hyödyntäminen onnistuu myös vähemmän kokeneelta ohjelmoijalta. Työssä huomattiin myös, että sokettien hyödyntäminen eri yhteystyypeillä on hyvin samankaltaista, ja näin ollen esimerkiksi TCP-sokettien käytön oppimisen jälkeen on helppo siirtyä UDP-soketteihin. Tietoturvan osalta kuitenkin havaittiin, että asianmukainen yhteyksien salaaminen vaatii aiempaa osaamista SSL- tai TLS-protokollista.

Tutkielman lopputuloksena todettiin, että sokettiohjelmointiin Java on hyvä ja helposti lähestyttävä ohjelmointikieli, mutta myös muilla kielillä on tarjottavana helppokäyttöiset kirjastot, minkä vuoksi käyttäjän kannattaa valita käyttötarkoitukseen sopiva kieli useampien muuttujien perusteella. Lisäksi huomattiin tietoturvan soveltamisen haasteet sokettien yhteydessä, ja tätä ehdotettiin jatkoselvitysaiheeksi. Toiseksi varteenotettavaksi tutkimusaiheeksi nostettiin WebSocket-protokolla, jota tässä työssä ei käsitelty.

Avainsanat: sokettiohjelmointi, TCP/IP-protokollapino, Java, tiedonsiirto.

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

ARPANET	Advanced Research Projects Agency Network
API	Application Programming Interface
DTLS	Datagram Transport Layer Security
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
IPTV	Internet Protocol Television
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
JSSE	Java Secure Socket Extension
QOTD	Quote Of The Day
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol

1. JOHDANTO

TCP/IP-protokollapino on tärkeä osa internetissä tapahtuvaa tiedonsiirtoa ja mahdollistaa tiedonsiirron kahden laitteen välillä internetin yli. Jotta sovellukset voivat hyödyntää edellä mainitun protokollapinon tarjoamia palveluita, tarvitaan soketteja ja tarkemmin sokettiohjelmointirajapintoja. Yleisesti käytetty vertaus kuvaamaan soketteja on kahden ihmisen välinen kommunikointi matkapuhelimen välityksellä. Tällaisessa kommunikoinnissa matkapuhelimet toimivat puhelinyhteyden päätepisteinä, joiden avulla henkilöt voivat keskustella keskenään. Soketit toimivat internetissä tapahtuvassa tiedonvälityksessä samankaltaisesti: puhelinten sijaan keskenään keskustelevat tietokoneet ottavat yhteyttä toisiinsa sokettien avulla.

Sokettien historia juontuu jo 1960-luvulle ARPANET:n aikoihin, kun eri valmistajilta tulevien koneiden haluttiin kommunikoivan keskenään. Termi soketti puolestaan on mainittu vuonna 1971 julkaistussa RFC 147 -dokumentissa (Winett, 1971). Nykyaikaisten TCP/IP-pohjaiseen tiedonsiirtoon perustuvien sokettien synty voidaan sijoittaa vuoteen 1982, jolloin UNIX-käyttöjärjestelmän versiot 4.1BSD ja 4.2BSD julkaistiin (Gay, 2021).

Tässä tutkielmassa kuvataan sokettiohjelmoinnin kooditoteutuksia erityisesti Java-ohjelmointikielen näkökulmasta, mutta myös muita ohjelmointikieliä esitellään lyhyesti. Lisäksi sivutaan myös tietoturvaan liittyviä asioita. Sokettiohjelmoinnista voi olla haastavaa löytää tietoa suomeksi, ja työn tarkoituksena onkin tarkastella, miksi sokettiohjelmointia tehdään, miten se tapahtuu käytännön tasolla ja esitellä tuoretta tietoa liittyen sokettiohjelmointirajapintojen käyttämiseen Javalla.

Javalla tapahtuva verkko-ohjelmointi valikoitui tutkielman aiheeksi, sillä Javalle on olemassa olevat kirjastot sokettiohjelmointia varten, ja se on ensimmäinen ohjelmointikieli, joka on suunniteltu verkkosovellukset huomioiden alusta alkaen (Harold, 2013). Lisäksi koodiesimerkeissä Java voi olla helpommin omaksuttavaa kuin esimerkiksi C:llä kirjoitettu koodi, joskin lukijalla voi olla omia preferenssejään ohjelmointikieliin liittyen. Osittain tämän vuoksi tutkielmassa on haluttu tuoda esiin myös se, että sokettiohjelmointi on mahdollista myös muilla kielillä. Esimerkiksi C:lle ja C++:lle on kattavat kirjastot sokettien hyödyntämiseen. Lähteinä tässä työssä käytetään eri ohjelmointikielien dokumentaatioita, aiheesta tehtyjä kirjoja sekä kirjallisuusselvityksiä ja

alan artikkeleita. Tarkoituksena on koota tietoa aiheesta kattavasti eri lähteistä ja havainnollistaa esitettyä tietoa koodiesimerkein. Tutkielmassa vertaillaan myös Javalla tapahtuvaa ohjelmointia muihin siinä esitettyihin ohjelmointikieliin.

Luvussa 2 esitellään sokettiohjelmointia yleisesti sekä paneudutaan C- ja Python-kieliin ja niiden erityisominaisuuksiin sokettirajapintoja käytettäessä. Lisäksi perehdytään eri ohjelmointikielien toimintaan hajautetussa järjestelmässä. Lopuksi käydään lyhyesti läpi Javan perusominaisuuksia, minkä jälkeen siirrytään lukuun 3, jossa syvennytään tarkemmin sokettiohjelmointiin Javalla ja sen tarjoamilla kirjastoilla. Luvussa 3 kerrotaan aiheesta yleisesti sekä syvennytään 3:een eri tapaan, joilla soketteja voidaan toteuttaa. Tämän jälkeen käsitellään sokettien lähettämän datan salaamista sekä sokettiohjelmien tietoturvaa lähdekoodin osalta. Lopuksi tarkastellaan Java sokettien suorituskykyä, minkä jälkeen tehdään yhteenveto.

2. VERKKOPOHJAINEN SOKETTIOHJELMOINTI

2.1 Yleistä sokettiohjelmoinnista

Asiakaspalvelintyyppiset ohjelmat ovat nykyisin yksi yleisempiä tapoja toteuttaa palveluja verkossa. Tämän tyyppisissä sovelluksissa asiakas eli esimerkiksi tavallinen tietokoneen käyttäjä voi käyttää palvelimen luomia käyttöliittymiä verkkoyhteyden kautta. Tällaisen palvelun mahdollistaminen vaatii luonnollisesti datan siirtoa palvelimen ja asiakkaan välillä, ja soketit toimivat tällaisen verkon yli tapahtuvan viestinnän päätepisteinä. (Gokhale & Mandir, 2021)

Soketti API:t eli ohjelmointirajapinnat mahdollistavat yhteyden muodostamisen kahden tällaisen päätepisteen välillä. Soketit eivät ole kiinnitettyjä vain yhteen asiakkaaseen, vaan ne voivat palvella useampaa yhtäaikaisesti. Tavallisesti siirtoyhteyskerroksella käytetään joko TCP- tai UDP-protokollaa. Verkkopohjaisessa sokettiohjelmoinnissa soketti on yhdistetty tiettyyn TCP- tai UDP-porttiin, jonka toiminnan se mahdollistaa. Toki soketteja käytetään myös saman järjestelmän sisällä, kuten samalla tietokoneella tapahtuvien prosessien välillä. Esimerkiksi Unix domain -soketti mahdollistaa tällaisen prosessien välisen tiedonsiirron. Jotta eri puolella internetiä olevien laitteiden välille saadaan muodostettua yhteys, tarvitaan portin lisäksi IP-osoite. IP-osoitteen avulla päästään siis laitteelle, jonka kanssa halutaan kommunikoida, ja portin avulla spesifioidaan itse palvelu. On myös huomioitava, että sokettia luodessa tarvitaan myös paikallinen IP-osoite sekä porttinumero, jotta järjestelmä tietää, mihin porttiin käytettävältä palvelulta tuleva data ohjataan. Osoitteiden ja porttinumeroiden lisäksi on vielä tärkeää määritellä yhteystyyppi, sillä TCP- ja UDP-tiedonsiirrot käyttävät eri soketteja. (Van Winkle, 2019) Yleisemmille palveluille on oletusarvoisesti varattu tiettyjä porttinumeroita. Näitä portteja kutsutaan niin sanotuiksi ”well known” -porteiksi. Tästä esimerkkinä selaimet ja verkkopalvelimet käyttävät http-protokollaa toimittaessaan html-sivuja, ja tälle http-protokollalle on varattu portti 80.

On myös syytä erottaa toisistaan WebSocket-protokolla ja sokettiohjelmointi yleisesti. WebSocket-protokollan tarkoitus on mahdollistaa kahden päätepisteen reaaliaikainen tiedonvälitys molempiin suuntiin. WebSocket-protokolla on suunniteltu toimimaan selainpohjaisesti. Se toimii samankaltaisesti http-protokollan kanssa kuitenkin niin, että useita TCP-pohjaisia http-yhteyksiä ei tarvita vaan samaa yhteyttä käytetään molempiin suuntiin. (Fette & Melnikov, 2011) On kuitenkin muistettava se, että WebSocket-protokollan toimiessa TCP-yhteyden avulla, tarvitaan tämän TCP-yhteyden

päätepisteissä tavallisia soketteja. Tässä tutkielmassa kuitenkin pysyttäydytään perinteisten sokettiyhteyksien piirissä.

Soketin kooditoteutuksen kirjoittaminen on mahdollista useammalla kielellä, joista muutamaa esittelen myöhemmin. Kieltä valittaessa ei kuitenkaan tarvitse murehtia siitä, mitä kieltä mahdollisesti yhteyden toisessa päässä olevan käyttäjän soketeissa on käytetty. Datan lähettäminen ja vastaanottaminen onnistuu, vaikka asiakkaan soketti olisi toteutettu Javalla ja palvelimen C:llä. (Maata, et al., 2017) Mitä kieltä koodin kirjoittamisessa käytetäänkään, hyödynnetään toiminnoissa ohjelmointirajapintaa. API eli ohjelmointirajapinta on ikään kuin pyyntöjen käsittelijä, jonka avulla erilaisia toimintoja, kuten pyyntöjä, voidaan suorittaa. Rajapintoja on lukuisia erilaisia, ja ne tarjoavat hyvinkin erilaisia tietoja ja funktioita. Tässä tutkielmassa hyödynnetyt rajapinnat tarjoavat toimintoja sokettien tarpeisiin esimerkiksi yhteyksien avaamiseen ja sulkemiseen.

2.2 C-kieli

C on Dennis Ritchien kehittämä ja vuonna 1972 julkaistu käännettävä ohjelmointikieli. C on suosittu kieli niin sanotussa järjestelmäohjelmoinnissa, ja suositut käyttöjärjestelmät, kuten Linux ja Windows, ovat suureksi osaksi kirjoitettu käyttäen C:tä. Kieltä käytetään myös sulautettujen järjestelmien ohjelmointiin eli esimerkiksi elektroniikassa. (Dmitrović, 2021, luku 1.) Edellä mainittujen asioiden ja ”rautaläheisyytensä” vuoksi C on oiva kieli myös sokettiohjelmointiin. Etenkin koneenläheisessä toteutuksessa, jossa halutaan mahdollisimman tehokasta suorituskykyä ja hyödyntää esimerkiksi rinnakkaisuutta, C tarjoaa hyviä työkaluja. Näiden ominaisuuksien ansiosta C:tä käytetään myös sokettiohjelmoinnissa, ja sokettiohjelmointirajapinnat kehitettiin alun perin käyttäen kyseistä kieltä.

C:llä ohjelmointi kuitenkin luo myös omaa mutkikkuuttansa itse ohjelmointiin, mistä esimerkkinä toimivat Winsock- ja Berkeley-rajapinnat. Winsock on rajapinta, joka määrittelee, miten TCP/IP-protokollapinon toiminnot toteutuvat Windows-käyttöjärjestelmässä. Tätä rajapintaa hyödynnetään myös C:llä tapahtuvassa sokettiohjelmoinnissa Windowsille. Berkeley on puolestaan samankaltainen rajapinta, mutta se on tarkoitettu puolestaan unix-pohjaisille käyttöjärjestelmille. Winsock rajapinnalle täytyy myös tehdä alustus- ja siivousvaiheet ennen kuin kyseistä rajapintaa voidaan hyödyntää. Berkeley-soketeille näitä vaiheita ei vaadita. (Dmitrović, 2021, luku 1.) On siis muistettava, että tehokkuudella on aina hintansa mikä voi luoda ohjelmakoodin kirjoittajalle ongelmia ratkaistavaksi. Lisäksi koodin mutkistuessa sen ylläpidettävyys kärsii, ja isommissa projekteissa se voi muodostua haasteeksi.

C:llä sokettien hyödyntäminen onnistuu POSIX-standardia noudattavissa käyttöjärjestelmissä sys/socket.h-otsikkotiedoston avulla. Selvennyksenä, POSIX on standardi, jota noudattavat pääasiassa Unix-pohjaiset käyttöjärjestelmät. Otsikkotiedosto on puolestaan tiedosto, joka sisältää määrittelyt esimerkiksi vakioarvoille, makroille sekä globaaleille muuttujille (JavaTpoint, 2021). Otiskkotiedostot voivat olla käyttjän luomia tai järjestelmän ennalta määrittämiä kuten edellä mainittu socket.h. Kyseisen otsikkotiedoston avulla on mahdollista hyödyntää SOCK_STREAM ja SOCK_DGRAM määrittelyjä TCP- ja UDP-sokettien luomiseen. Ensin mainittua voidaan käyttää TCP:lle ja toisena mainittua taas UDP:lle.

Windows-käyttöjärjestelmä ei noudata POSIX-standradia eikä sille voi näin ollen myöskään hyödyntää socket.h-tiedostoa. Windowsilla soketteja voi hyödyntää winsock2-tiedoston avulla. Tämän lisäksi ohjelman on WSAStartup-funktiolla asennettava winsock WSADATA tietorakenteen avulla. Tämän jälkeen sokettien määrittäminen onnistuu AF_INET ja SOCK_STREAM sekä SOCKET_DGRAM määrittelyjen avulla kuten Unix-käyttöjärjestelmilläkin. (Microsoft, 2021)

2.3 Python

Python on Guido Van Rossumin kehittämä ohjelmointikieli, joka julkaistiin vuonna 1990. Python on yksi yleisimmistä nykyisin käytetyistä ohjelmointikielistä. Pythonista on myös useita toteutuksia, joista yksi on CPython, joka pohjautuu edellä esiteltyyn C-kieleen. Python eroaa C:stä ja Javasta esimerkiksi siinä, että se on tulkattava kieli, eli sitä ei käännetä konekielelle ennen ajoa. Lisäksi tyypitys on dynaamista, eli muuttujien tyyppiä ei määritellä etukäteen, vaan ajon aikana. Nämä ominaisuudet tekevät Pythonista helposti lähestyttävän ja ymmärrettävän, mutta kirjoitettujen ohjelmien tehokkuus kärsii. Javan tavoin Python tukee olio-ohjelmointia. (Sharma, et al., 2021) Python on myös hyvä tapa lähestyä sokettiohjelmointia yksinkertaisuutensa vuoksi. Yksinkertaisia sokettiohjelmia luodessa Python tarjoaa hyvät kirjastot ja kirjoittajan tarvitsee huolehtia vähemmän yksityiskohdista. Seuraavana esitetään koodiesimerkki yksinkertaisesta sokettiolion luonnista Python-kielellä.

```
import socket

HOST = socket.gethostbyname(socket.gethostname())
PORT = 9000

SERVER = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
SERVER.bind((HOST, PORT))
```

Ohjelma 1. Soketin luominen Python-kielellä.

Ohjelma siis ainoastaan ottaa käyttöön socket-nimisen moduulin, määrittää mitä IP-osoitetta ja porttia soketin on tarkoitus kuunnella, minkä jälkeen alustetaan halutun tyyppinen sokettiolio. Viimeisellä rivillä sidotaan luotu sokettiolio osoitteeseen ja porttiin. Edellinen koodiesimerkki havainnollistaa Pythonin edun siinä, että aloittelevallekin ohjelmoijalle koodi on selkeää ja helposti ymmärrettävää.

Pythonin peruskirjasto tarjoaa tavallisten TCP/IP-yhteyttä hyödyntävien sokettien lisäksi mahdollisuuden käyttää jo mainittuja Unix-soketteja, NETLINK-soketteja sekä esimerkiksi bluetooth-soketteja. (Python Software Foundation, 2023) Kuten jo mainittua Unix-soketteja voidaan hyödyntää samalla koneella olevien prosessien välisessä kommunikaatiossa tehokkaammin kuin IP-yhteyttä käyttävien, sillä resursseja ei käytetä esimerkiksi reitittämiseen. NETLINK-soketit taas on kehitetty Kernelin ja käyttäjätason prosessien väliseen kommunikaatioon sekä Kernelin sisäiseen viestien välitykseen (Salim, et al., 2003). Palvelinpuolelle Pythonille on socketserver-kirjasto, joka tarjoaa luokat TCP- ja UDP-soketeille sekä UnixStream- ja UnixDatagramServer-luokat Unix-tyyppisille soketeille. Yhteyksien salaamiseen Pythonilla voi hyödyntää ssl-kirjastoa, jonka avulla voidaan hyödyntää TLS- ja SSL-protokollia sokettien välisen tiedonsiirron suojaamiseen. (Python Software Foundation, 2023) TLS- ja SSL-protokollista kerrotaan lisää luvussa 3.

2.4 Java

C- ja C++-kielten rajoitteena on koodin mukauttaminen käyttöjärjestelmään ja prosessoriarkkitehtuuriin. C- ja C++-kielinen koodi vaatii kääntäjän, joka on suunniteltu prosessoriarkkitehtuurille, jolla ohjelmakoodi halutaan suorittaa. Java kehitettiin vastaamaan tähän ongelmaan, kun halutiin kehittää ohjelmointikieli, joka ei olisi alustariippuvainen C:n ja C++:n tavoin. (Schildt, 2018)

Sun Microsystems julkaisi ensimmäisen version Javasta vuonna 1995, ja internetin käytön yleistyessä myös Javan suosio kasvoi. Javan syntaksi on lähellä C ja C++ kielten syntaksia, ja C++:n tavoin Java noudattaa olio-ohjelmoinnin periaatteita. Java poikkeaa C-kielestä kuitenkin siinä, että kirjoitettu koodi käännetään tavukoodiksi ennen ajamista Java-virtuaalikoneella. Tavukoodin ja virtuaalikoneen avulla saavutettiin haluttu alustariippumattomuus, jonka lisäksi turvallisuus lisääntyi, sillä virtuaalikoneen hallinnoidessa ohjelman suoritusta voitiin ehkäistä suoritettavan ohjelman pääsy sille kuulumattomiin paikkoihin järjestelmässä. (Schildt, 2018)

Javan alustariippumattomuus mahdollistaa helpomman hajautetun järjestelmän hyödyntämisen. Hajautetussa järjestelmässä isompi ongelma jaetaan pienempiin osiin

ja nämä osat jaetaan eri tietokoneiden ratkaistavaksi verkkoyhteyden kautta. Jotta eri tietokoneilla olevat prosessit voivat keskustella keskenään, tarvitaan soketteja, jotka ovat Javassa keskeisessä roolissa. (Malleswara & Pattamsetti, 2017, p. 35)

2.5 Ohjelmointikielet hajautetussa järjestelmässä

Aiemmin todettiin, että sokettien toteutuskielellä ei ole väliä, ja ne voivat kommunikoida keskenään riippumatta siitä, millä ohjelmointikielellä ne on toteutettu. Tästä huolimatta useammalla eri ohjelmointikielillä toteutettujen hajautettujen järjestelmien tehokkuus riippuu myös siinä käytettyjen kielien yhteistoiminnasta. Hajautetulla järjestelmällä tarkoitetaan järjestelmää, jossa useampi järjestelmä, esimerkiksi tietokone, toimii yhteistyössä ongelman tai tehtävän ratkaisemiseksi (IBM, 2021). Tällaisessa järjestelmässä eri sijainneissa olevien tietokoneiden ja niissä käynnissä olevien prosessien on pystyttävä kommunikoimaan keskenään, ja tämä voidaan toteuttaa esimerkiksi sokettien avulla.

IEEE Symposium on Computers and Communicationissa (ISSE) julkaistussa konferenssijulkaisussa käsitellään datavirtoihin pohjautuvien tietojenkäsittelyjärjestelmien toimintaa, kun ne on toteutettu hajautetusti niin, että eri osat järjestelmästä on toteutettu eri ohjelmointikielillä. Julkaisussa vertaillaan C, Python ja Java kielillä toteutettujen komponenttien yhteistoimintaa ja yhtenä datan välitysmuotona komponenttien välillä käytettiin TCP-soketteja. Toisena kieleltään paikallisena tiedonvälitysmuotoina tutkimuksessa on käytettiin nimettyjä putkia (FIFO). Kolmas tapa murtaa niin sanottu kielimuuri järjestelmien välillä oli hyödyntää rajapintoja, kuten Javan JNI sekä Pythonille luotu ctypes, joiden avulla C-kielisiä metodeja voidaan hyödyntää Javalla sekä Pythonilla ja päinvastoin. Tämän työn kannalta olennaista tutkimuksessa on TCP-sokettien avulla toteutetut datan siirrot. (Ruediger, et al., 2016)

Tutkimuksessa käytettiin datageneraattoreja, jotka loivat vakiokokoisia datailmentymiä. Nämä datailmentymät lähetettiin ryhmissä, ja datan luomiseen rajoitteita loivat itse datan luominen, sen siirtäminen eteenpäin ja vastaanottaminen. Mittayksikkönä eri kielillä luotujen komponenttien välillä toimi datailmentymät per sekunti (data instances per second). Tutkimuksessa käytetty järjestelmä koostui datageneraattorista, lähettimestä ja vastaanottimesta. Kielien yhdistelmät testattiin niin, että toinen kielistä vastasi datan luomisesta ja lähettämisestä, kun taas toinen sen vastaanottamisesta. (Ruediger, et al., 2016)

Yhdistelmässä, jossa lähetys päässä käytettiin Pythonia ja vastaanottimessa C:tä, jäätiin ainoastaan 2Mdips:n nopeuteen. Tämän arveltiin johtuvan Pythonin suorituskyvystä

datailmentymien kokoamisessa lähetettäviksi ryhmiksi, jonka vuoksi Pythonia ei muissa tapauksissa käytetty lähettävässä komponentissa. 64 tavun datainstanssien maksiminopeudessa C:n ja Javan yhdistelmä saavutti parhaan tuloksen, kun tiedonvälitysmuotona oli soketti. Tämä yhdistelmä saavutti 66,8 Mdips:n suorituskyvyn. Yhdistelmät, joissa C:llä toteutettu komponentti oli lähettävässä päässä, pärjäsivät parhaiten. Kahden C:llä toteutetun komponentin suorituskyky oli 64,8 Mdips ja C:n sekä Pythonin yhdistelmän puolestaan 64,7 Mdips. Javan ollessa lähettävässä päässä suorituskyvyt olivat 56,3 Mdips C:n kanssa ja 56,9 Pythonin kanssa. C:llä sokettien avulla lähettävien järjestelmien suorituskyvyt olivat Fifo:a hyödyntäviä tehokkaampia, kun taas Javalla Fifo:a käyttävät järjestelmät päihittivät sokettien avulla toteutetut. (Ruediger, et al., 2016)

Kuten tutkimuksen tuloksista huomaamme ohjelmointikielellä on merkitystä tiedonsiirron ja datan käsittelyn kannalta. Tuloksista havaitaan Pythonin jo mainittu rajoitteisuus suorituskyvyn kannalta Javaan ja C:hen verrattuna, sekä se kuinka tutkimuksen järjestelmien tehokkuus oli soketteja hyödyntäessä parempi, kun lähettävässä komponentissa käytettiin C:tä Javan sijasta. Vaikka esitelty tutkimuksen tulokset eivät ainoastaan perustu siihen millä kielellä itse soketit on luotu, havainnollistaa se silti, että kun sokettien kanssa käytettyjä kieliä vaihdetaan, muuttuu järjestelmän suorituskyky kokonaisuudessaan.

3. JAVA SOKETTIOHJELMOINTI

3.1 Yleistä

Kuten jo aiemmin todettua saadaksemme aikaan yhteyden kahden eri IP-osoitteissa sijaitsevan prosessin välillä hyödyntäen soketteja, on määriteltävä 5 olennaista asiaa yhteyden muodostamisen kannalta. IP-osoite ja porttinumero, josta haluamme yhteyden muodostaa, sekä kohdeosoite ja -portti laitteelle, jonka haluamme tavoittaa. Näiden lisäksi on määriteltävä, mitä kuljetuskerroksen protokollaa tiedonsiirtoon käytetään, eli minkä tyyppistä sokettia hyödyntäen tiedonsiirto mahdollistetaan. Tavallisimmin kuljetuskerroksen protokollaksi valikoituu joko TCP- tai UDP-protokolla.

Javalla sokettiohjelmointiin tarkoitettuja luokkia tarjoaa Java.net -niminen kirjasto (Oracle, 2020). Kirjasto tarjoaa rajapintojensa kautta monia hyödyllisiä luokkia verkko-ohjelmointiin aina todennuksista poikkeuksien käsittelyihin. Tässä työssä hyödynnetään pääosin erityyppisiä sokettiluokkia sekä IP-osoitteiden ja datan käsittelyä helpottavia luokkia. Sokettien luomiseen kirjasto tarjoaa muun muassa Socket- ja ServerSocket-luokat, joista Socket toteuttaa asiakaspuolen TCP-tyyppisen soketin ja ServerSocket palvelinpuolen soketin. Kuten todettua soketin tyyppi voi olla myös esimerkiksi UDP-soketti, ja tähän Java.net kirjasto tarjoaa DatagramSocket-luokan. Myöhemmin syvennymme sekä TCP- että UDP-soketit tarjoaviin luokkiin, minkä lisäksi käsittelemme multicast-yhteyden mahdollistavan MulticastSocket-luokan.

Pelkkien sokettien luomiseen java.net -kirjasto riittää, mutta jotta voidaan käsitellä sokettien välittämää datavirtaa tarvitaan java.io -kirjastoa. Kirjasto tarjoaa erilaisia luokkia datavirtojen luomiseen ja lukemiseen. Tämän työn koodiesimerkeissä java.io -kirjastoa hyödynnetään viestien lähettämiseen asiakkaan ja palvelimen välillä.

3.2 TCP

TCP-tiedonsiirtoprotokolla mahdollistaa luotettavan tiedonsiirron kahden osapuolen välillä. Luotettavuudella tarkoitetaan tässä tapauksessa sitä, että protokolla huolehtii siitä, että kaikki tiedonsiirron aikana lähetetyt paketit saapuvat perille, pysyvät muuttumattomina ja oikeassa järjestyksessä. TCP-yhteyttä käytetään esimerkiksi sähköpostiviestien lähettämisessä, jolloin on tärkeää, että viestin sisältö saapuu muuttumattomana vastaanottajalle. (IBM, 2023) TCP-yhteyden avaamisessa toteutetaan niin sanottu kolmivaihekättely päätepisteiden välillä, mutta yksinkertaisissa kooditoteutuksessa tätä ei tarvitse huomioida. TCP-yhteyttä käyttävän soketin luominen

Javalla onnistuu Socket-luokan avulla. Oliolle annetaan IP-osoite, jossa vastaanottava soketti sijaitsee, sekä porttinumero, jota se palvelee. Java.io -kirjaston tarjoaman PrintWriter-luokan avulla voimme lähettää soketin kautta viestejä palvelimelle yhteyden muodostamisen jälkeen. Seuraavaksi on esitetty yksinkertainen asiakasohjelma, jossa muodostetaan TCP-yhteys toiseen sokettiin.

```
import java.net.*;
import java.io.*;

public class ClientSocket {
    public static void main(String args[]) throws IOException {
        System.out.println("Yhdistetään...");
        // Luodaan TCP-soketti yhteydenottoa varten
        Socket client = new Socket("localhost",3000);
        // Pidetään yhteyttä yllä, kunnes käyttäjä haluaa sen sulkea
        while(!client.isClosed()){
            BufferedReader readInput = new BufferedReader
            (new In-putStreamReader(System.in));
            System.out.println("Haluatko sulkea
            yhteyden?(kyllä/ei)?");
            PrintWriter output = new PrintWriter
            (client.getOutputStream(), true);
            String input = readInput.readLine();
            // Jos käyttäjä vastaa kyllä, suljetaan yhteys ja
            //välitetään tästä tieto palvelimelle.
            if (input.equals("kyllä")) {
                output.println("Suljetaan yhteys");
                client.close();
                System.out.println("Yhteys suljettu");
            } else {
                System.out.println("Säilytetään yhteys");
            }
        }
    }
}
```

Ohjelma 2. Yksinkertainen asiakasohjelma.

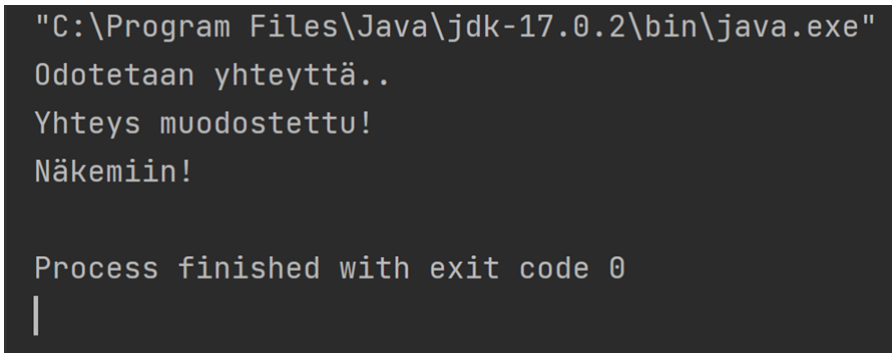
Kuten huomataan kooditoteutus ei ole kovinkaan monimutkainen. Ohjelma ainoastaan luo soketin, joka ottaa yhteyden porttiin 3000 ja pitää yhteyden yllä niin kauan kunnes käyttäjä haluaa sulkea yhteyden. Huomattavaa on itse sokettiolion määrityksessä käytetty ”localhost” avainsana. ”Local Host” -nimitys viittaa IPv4-osoitteeseen 127.0.0.1, eli niin sanottuun ”loopback-osoitteeseen”. Tämä osoite reititetään takaisin samalle laitteelle, ja sitä voidaan käyttää esimerkiksi ohjelmien kehittämisessä ja testaamisessa paikallisesti omalla laitteella. Tätä ominaisuutta käytetään myös esimerkkiohjelmassa. (GeeksforGeeks, 2023) Esimerkkiohjelmat ovat hyvin yksinkertaisia ja prosessit ovat samalla tietokoneella, mutta havainnollistavat kuitenkin sokettien toimintaa ja miten ne luodaan.

Tässä yksinkertaisessa ohjelmassa palvelinpuoli sijaitsee asiakasohjelman kanssa samalla tietokoneella, joten erillistä IP-osoitetta ei tarivta vaan myös palvelimen osoitteena toimii loopback-osoite. Myös jo mainittu avainsana ”localhost” löytyy palvelinpuolen ohjelmasta, joka esitellään seuraavaksi.

```
import java.net.*;
import java.io.*;

public class Server {
    public static void main(String args[]) throws IOException {
        System.out.println("Odotetaan yhteyttä..");
        //Luodaan TCP-soketti vastaanottamaan yhteyksiä.
        ServerSocket server = new ServerSocket(3000);
        Socket client = server.accept();
        System.out.println("Yhteys muodostettu!");
        BufferedReader readInput = new BufferedReader
        (new InputStreamReader(client.getInputStream()));
        String input = readInput.readLine();
        // Suljetaan yhteys, asiakkaan lähettäessä viestin
        // "suljetaan yhteys."
        if(input.equals("suljetaan yhteys")) {
            server.close();
        }
        System.out.println("Näkemiin!");

    }
}
```

Ohjelma 3. Yksinkertainen palvelinohjelma.

Palvelinpuolellakaan koodi ei ole erityisen monimutkainen. Ohjelma luo TCP-yhteyttä tukevan palvelinsoketin, ja accept-metodi käskee sokettia kuuntelemaan saapuvia yhteyksiä ja hyväksymään ne. Tämän jälkeen luetaan yhteyden kautta saapuvia viestejä ja pidetään yhteyttä yllä, kunnes vastaanotetaan ”suljetaan yhteys” -viesti.Seuraavassa kuvassa on esitetty palvelinohjelman tulostamat viestit yhden session aikana



Kuva 1. Palvelinohjelman tulostamat viestit

Aputulostusten avulla huomataan ohjelman kulku palvelimen näkökulmasta. ”Yhteys muodostettu” -teksti ilmaantuu vasta, kun asiakaspuolen ohjelma on käynnistetty.

”Näkemiin!” -teksti puolestaan tulostuu, kun asiakaspuolen ohjelmassa on vastattu käyttäjän toimesta ”kyllä”.

3.3 UDP

UDP-protokolla tarjoaa TCP:n tavoin kuljetuskerroksen palvelun, mutta ei takaa yhteyden luotettavuutta. UDP on TCP:tä kevyempi protokolla, eikä se sisällä esimerkiksi kolmivaihekättelyä, mikä tekee yhteyden muodostamisesta osapuolten välillä nopeampaa. Koska UDP on yhteydetön protokolla, lähettäjä ei tiedä mitkä paketit saapuvat perille ja tulevatko ne oikeassa järjestyksessä. Yhteydettömyys kuitenkin vähentää viivettä tiedonsiirrossa ja säästää kaistanleveyttä, jonka vuoksi UDP:llä on käyttökohteensa reaaliaikaista tiedonsiirtoa vaativissa käyttökohteissa, kuten verkkopeleissä ja suoratoistossa. Tämän tyyppisissä sovelluksissa olennaisempaa on datan liikkuminen nopeasti sen sijaan, että data vastanotettaisiin täydellisenä. (GeeksforGeeks, 2023)

UDP-tyyppiseen sokettiohjelmointiin Javan.net -kirjasto tarjoaa DatagramSocket-luokan, jonka avulla voidaan luoda UDP-tiedonsiirtoa hyödyntävä soketti. TCP-yhteyttä hyödyntävästä Socket-luokasta poiketen DatagramSocket-oliota voidaan käyttää sekä asiakas- että palvelinpuolella, eikä erillistä ServerSocket-luokkaa tarvitse hyödyntää.

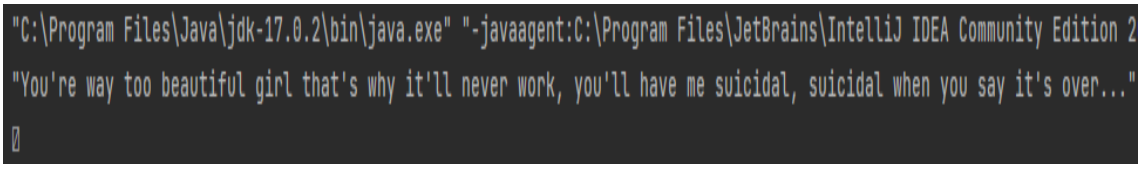
Seuraavaksi tutustumme UDP-tiedonsiirtoa hyödyntävään ohjelmaan, joka hyödyntää ”quote of the day”-palvelua (QOTD). QOTD lähettää pyynnöstä viestin, tai lainauksen ja on tarkoitettu testaustarkoitukseen. Palvelimen kanssa voi kommunikoida joko TCP-, tai UDP-yhteyden avulla ja QOTD palvelee oletuksena portissa 17. (Postel, 1983) Esimerkkiohjelmassa on otettu vaikutteita codejava.net -sivustolla julkaistusta esimerkkiohjelmasta (Minh, 2019).

```
public class UdpClient {
    public static void main(String args[]) throws IOException {
        // Isäntänimi ja porttinumero yhteydenottoa varten
        String host = "djxmmx.net";
        int portNumber = 17;

        InetAddress address = InetAddress.getByName(host);
        DatagramSocket udpSocket = new DatagramSocket();

        byte[] buffer = new byte[512];

        //Alustetaan lähtettävä UDP-paketti
        DatagramPacket request = new DatagramPacket(buffer,
        buffer.length, address, portNumber);
        udpSocket.send(request);

        // Alustetaan vastaanotettava UDP-paketti
        DatagramPacket response = new DatagramPacket(buffer,
        buffer.length);
```

```
            udpSocket.receive(response);

            String quote = new String(buffer,0,response.getLength());
            System.out.println(quote);
        }
}
```

Ohjelma 4. Yksinkertainen UDP-tiedonsiirtoa käyttävä asiakasohjelma.

Esimerkki ohjelmassa alustetaan ensin isäntänimi ja porttinumero, joiden avulla yhteyden muodostus QOTD-palvelimelle onnistuu. Isäntänimen avulla saadaan haettua palvelimen IP-osoite, joka onnistuu InetAddress-luokan tarjoaman getByName-metodin avulla. InetAddress-luokan avulla onnistuu niin IPv4- kuin myös IPv6-osoitteiden määrittäminen. Itse UDP-soketti määritetään tuttuun tapaan, jonka jälkeen luodaan buffer-niminen säiliö, joka on kooltaan 512 tavua. Näin varmistutaan, että datagrammia ei fragmentoida, eli pilkota lähetyksen aikana pienempiin osiin, mikä yksinkertaistaa pakettien käsittelyä. Tämän jälkeen luodaan DatagramPacket-luokan avulla pyyntö, joka lähetetään palvelimelle, sekä DatagramPacket-olio response, jonka avulla palvelimen lähettämä data saadaan käsiteltyä. Lopuksi tulostetaan QOTD-palvelimen lähettämä lainaus, joka on esitetty seuraavassa kuvassa



Kuva 2. Quote of the day -palvelimen lähettämä viesti.

Esimerkkiohjelma saa onnistuneesti vastaanotettua palvelimen lähettämän lainauksen. Toisin kuin TCP-esimerkissä nyt otettiin yhteyttä oikeaan palvelimeen internetin välityksellä. Ohjelmakoodi ei kuitenkaan ole juurikaan monimutkaisempi, ja ohjelman kirjoittaminen sekä ymmärtäminen on ohjelmoijalle varsin yksinkertaista. UDP-tiedonsiirtoa hyödyntävässä palvelinohjelmassa voitaisiin käyttää samaista DatagramSocket-luokkaa. Tämän vuoksi ei ole tarpeellista esittää erillistä palvelinpuolen ohjelmaa, mutta halutessaan lukija voi tutustua codejava.net sivustolla luotuun ohjelmaan, joka toimii QOTD-palvelimen kanssa saman tapaisesti (Minh, 2019). Sekä TCP- että UDP-tiedonsiirrot, jotka esimerkkiohjelmissa toteutettiin olivat niin sanottuja täsmälähetyksiä (unicast). Täsmälähetyksissä dataa lähetetään ainoastaan yhdelle vastaanottajalle. Seuraavassa luvussa esitellään ryhmälähetys (multicast), jossa vastaanottajia voi olla useita.

3.4 Multicast

Multicast eli ryhmälähetys on tapa lähettää IP-kehyksiä useammalle vastaanottajalle yhtäaikaisesti. Ryhmälähetykselle on määritelty oma IP-osoitteensa, johon lähettämällä paketit jaetaan kaikille ennalta määrätyn ryhmän jäsenille. (Deering, 1989) Tyypillisesti ryhmälähetyksessä hyödynnetään UDP-tiedonsiirtoa, jolloin myös käyttökohteet ovat saman tapaisia kuin UDP:n tapauksessa. Ryhmälähetyksen etuna onkin se, että samaa dataa voidaan lähettää useammalle siitä kiinnostuneelle vastaanottajalle saman aikaisesti. Tätä ominaisuutta hyödynnetään esimerkiksi IPTV-palveluissa, joissa usealle kanavaa katsovalle asiakkaille voidaan lähettää samaa dataa. Ryhmälähetyksen hyödyntäminen IPTV-palvelussa on kaistanleveyden kannalta tehokkaampaa kuin UDP-ohjelmassa käytetyn täsmälähetyksen, tai yleislähetyksen (broadcast) käyttö (Li & Lee, 2020). Yleislähetyksessä sama lähetys jaetaan niin, ettei vastaanottajien ryhmää ole ennalta määrätty.

Javassa java.net-kirjasto tarjoaa luokan MulticastSocket, jolla voidaan luoda soketti IP multicast-tyyppisten pakettien lähettämiseen ja vastaanottamiseen. Javan multicast-soketti on jo käytetty DatagramSocket-luokan soketti, jolle on lisätty mahdollisuus liittyä ryhmälähetysisäntien (multicast host) ryhmiin. (Oracle, 2020)

Seuraavaksi havainnollistetaan ryhmälähteystä luomalla kolme vastaanottavaa sokettia sekä yksi DatagramSocket-luokkaa hyödyntävä lähettävä soketti. Yhtä hyvin voitaisiin luoda jokaiselle ryhmän jäsenelle myös lähettävät soketit, mutta esimerkin paisumisen välttämiseksi käytetään vain yhtä lähettävää osapuolta, jota muut kuuntelevat. Lähetyksessä ei tarvita MulticastSocket-luokkaa, sillä datan lähettäminen ryhmälle ei vaadi siihen kuulumista (Oracle, 2020). Alla on esitetty MulticastServer-ohjelma, jossa on määritelty ryhmälähetyksen dataa lähettävä osapuoli.

```
public class MulticastServer {
    public static void main(String[] args)
    throws IOException, InterruptedException {
        int port = 8080;
        DatagramSocket server = new DatagramSocket();
        InetAddress group = InetAddress.getByName("230.0.0.1");
        int number = 0;
        System.out.println("Aloitetaan lähetys...");
            while (number <= 10) {
                String message = valueOf(number);
                byte[] msg = message.getBytes();
                DatagramPacket packet =
                new DatagramPacket(msg, msg.length, group, port);
                server.send(packet);
                number += 1;
                TimeUnit.SECONDS.sleep(3);
            }
            server.close();
        }
```

```
    }
```

Ohjelma 5. Ryhmälähetyksen lähettävä osapuoli.

Ohjelmassa alustetaan UDP-tiedonsiirtoa käyttävä soketti, joka lähettää 3 sekunnin välein numeron, jota kasvatetaan yhdellä joka lähetyksen jälkeen. Paketit lähetetään IP-osoitteeseen 230.0.0.1 ja porttiin 8080, joten seuraavana tehtävänä onkin määrittää kuuntelevat osapuolet vastaanottamaan numeroita näiden tietojen pohjalta.

Kuuntelevien ohjelmien koodit ovat luokkien nimiä lukuun ottamatta identtisiä. Tämän vuoksi riittää ainoastaan ensimmäisen kuuntelevan ohjelman koodin ymmärtäminen ja se on esitetty seuraavaksi.

```
public class MulticastClient1 {
    public static void main(String[] args) throws IOException {
        MulticastSocket clientSocket = new MulticastSocket(8080);
        InetAddress group = InetAddress.getByName("230.0.0.1");

        clientSocket.joinGroup(group);

        byte[] buffer = new byte[1024];
        System.out.println("Vastaanotetaan dataa...");
        DatagramPacket packet = new DatagramPacket(buffer,
        buffer.length);
        clientSocket.receive(packet);
        String message = new String(packet.getData(),
        packet.getOffset(), packet.getLength());
        System.out.println(message);

        clientSocket.leaveGroup(group);
        clientSocket.close();
        }
    }
```

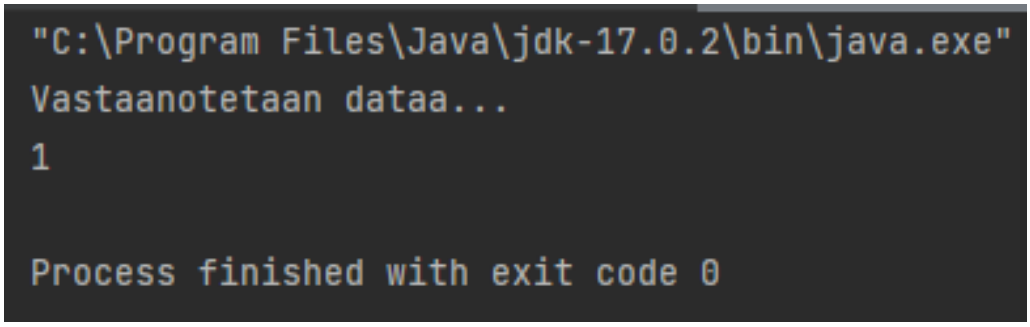
Ohjelma 6. Ryhmälähetystä kuunteleva ohjelma.

Ohjelmassa ryhmälähetystä kuunteleva ohjelma liittyy aiemmin määritetyssä IP-osoitteessa sijaitsevaan ryhmään ja tulostaa sieltä saapuvat viestit. Sinänsä ohjelmakoodi ei juuri poikkea tavallisen UDP-soketin kuuntelemisesta. Uutta on MulticastSocket-luokan käyttö sokettiolion luonnissa sekä joinGroup-metodin käyttö ryhmään liittymisessä. Lisäksi sokettia sulkiessa poistutaan ensin ryhmästä, jonka jälkeen soketti suljetaan. Ohjelma vastaanottaa yhden viestin ryhmälähetyksestä ja tulostaa sen. Kaksi muuta asiakasohjelmaa toimivat vastaavalla tavalla, mutta esimerkin vuoksi luodaan yksi ohjelmista liittymään ryhmään osoitteessa 230.0.0.2.

Oletus siis on, että tämä kolmas asiakasohjelma ei vastaanota lähettävän osapuolen viestejä. Ohjelmat käynnistetään noin 3 sekuntia toistensa jälkeen, jolloin vastaanotetut numerot pitäisivät olla eri suuruisia. Seuraavissa kuvissa on esitetty jokaisen

asiakasohjelman tulostamat luvut, kun ne käynnistetään toistensa jälkeen palvelinohjelman käynnistämisen jälkeen.

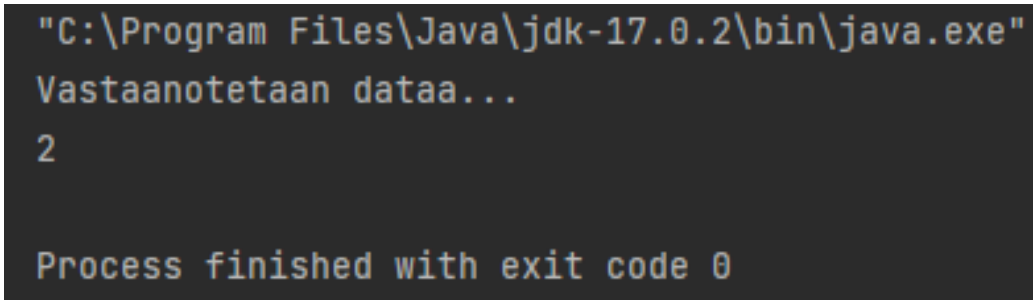
Lähdetään tarkastelemaan asiakasohjelmien kulkua. Seuraavassa kuvassa on esitetty ensimmäisenä käynnistetyn asiakasohjelman suoritus.



Kuva 3. Ensimmäisen asiakasohjelman tulostukset.

Kuten huomaamme ohjelma ei ehdi numeron 0 vastaanottamiseen, mutta yhteyden muodostamisen jälkeen palvelin lähettää numeron 1, ja vastaanotettuaan paketin asiakasohjelma tulostaa numeron ja lopettaa suorituksen onnistuneesti.

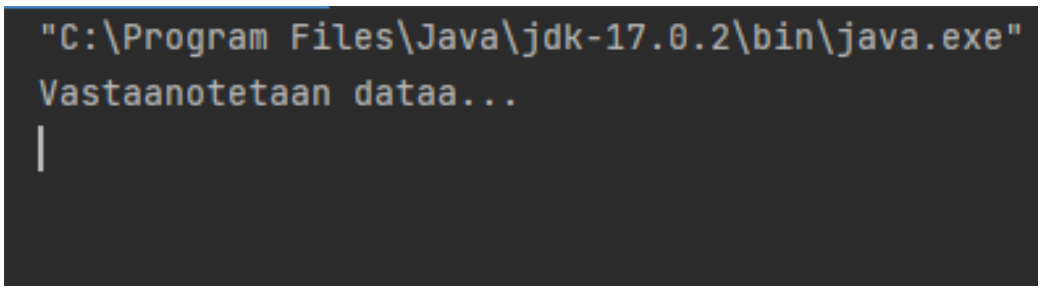
Seuraavassa kuvassa tarkastelemme toisena käynnistetyn asiakasohjelman kulkua.



Kuva 4. Toisen asiakasohjelman tulostukset.

Toinen ohjelma puolestaan yhdistää palvelimelle hetkellä, jonka jälkeen vuorossa on numeron 2 lähetys. Tämäkin ohjelma vastaanottaa ryhmälähetyksen onnistuneesti.

Lopuksi tarkastelemme asiakasohjelmaa 3, joka yhdistää eri multicast-osoitteeseen kuin edelliset asiakasohjelmat. Seuraavassa kuvassa on esitetty asiakasohjelman 3 kulku.



Kuva 5. Kolmannen asiakasohjelman tulostukset.

Kuten huomaamme, ohjelma 3 ei onnistu vastaanottamaan ryhmälähetyksen dataa, vaikka se käyttää yhtä lailla MulticastSocket-oliota, joka on luotu kuuntelemaan porttia 8080. Tämä johtuu siitä, että ohjelma liittyi väärään ryhmälähetysosoitteeseen, eikä sille näin ollen lähetetä dataa palvelinohjelmalta. Kuten TCP-esimerkissä myös nämä prosessit ajetaan samalla tietokoneella. Kuitenkin ohjelmien periaatteet ja sokettien käyttö ei olennaisesti muutu, vaikka ohjelmat ajettaisiin eri laitteilla. Esimerkkiohjelmista huomataan myös, että ryhmälähetystä hyödyntävien ohjelmien kirjoittaminen on yksinkertaista ja tavallisten viestien lähettäminen onnistuu ainoastaan kahta kirjastoa hyödyntämällä.

3.5 JSSE ja tietoturva

Tietoturvaratkaisut on otettava huomioon lähes jokaisessa modernissa tietoliikennejärjestelmässä ja teknologiassa, eivätkä soketit ole poikkeus. Lähettäessä dataa internetin kautta on pidettävä huoli siitä, että lähetetty data on suojattu mahdollisilta hyökkääjiltä, ja ettei ohjelmakoodissa ole mahdollisia haavoittuvuuksia.

Sokettien tietoturvassa lähimpänä toimivat niin sanottu Secure Socket Layer- (SSL) ja tästä paranneltu Transport Layer Security-protokolla (TLS). SSL ja TLS toimivat sovelluskerroksella ja hyödyntävät itse datan siirrossa TCP-protokollaa. UDP-tiedonsiirron kanssa voidaan puolestaan hyödyntää DTLS-protokollaa, joka hyödyntää tavallisen TLS-protokollan ominaisuuksia, mutta ne on sovitettu yhteensopivaksi UDP-tiedonsiirron kanssa. (Pérez, 2014) Javalla edellä mainittujen protokollien hyödyntämisen sokettien kanssa mahdollistaa Java Secure Socket Extension -rajapinta. JSSE tukee SSL-protokollan versiota 3.0 ja TLS-protokollan versioita 1.0–1.3. (Oracle, 2023) Käytännön tasolla ohjelmoija saa JSSE:n sisältämät luokat käyttöönsä javax.net- ja javax.net.ssl -kirjastojen avulla. Ssl kirjaston tarjoamat SSLSocket- ja SSLServerSocket-luokat mahdollistavat SSL- ja TSL-protokollien käytön sokettien yhteydessä. Käytännössä JSSE:n tarjoamat palvelut laajentavat Java.secure- ja java.net-kirjastoja.

Mutkatonta tietoturvan implementointi soketteihin ei kuitenkaan ole. TLS-protokollan hyödyntäminen voi olla ohjelmoijalle monimutkaista toteuttaa ja ymmärtää, mikä puolestaan voi johtaa virheisiin salauksen ja turvallisen yhteyden toteuttamisessa.

New South Walesin yliopiston julkaisemassa artikkelissa arvioitiin JSSE-rajapinnan käytettävyyttä ja tunnistettiin käytettävyysongelmia, jotka voivat johtaa ohjelmoijan virheisiin sovelluskehityksessä. Tutkimus toteutettiin antamalla tutkimukseen osallistuneille henkilöille ohjelmointitehtäviä liittyen JSSE-rajapintojen käyttöön.

Osallistujat saivat tehdä tehtävänsä etänä itselleen sopivalla aikataululla. Heitä pyydettiin tallentamaan työskentelystään kuvakaappausvideota sekä äänitallennetta, missä he kertoivat miten päätyivät tekemiinsä ratkaisuihin. Tutkimuksen tehtävät suoritti 12 osallistujaa, joista kokemattomimmalla oli alle vuoden ohjelmointikokemus ja kokeneimmalla puolestaan yli 10 vuoden. Osallistujat kokivat haastavaksi muun muassa rajapinnan käytön oppimisen, sillä he kokivat, että se sisältää liikaa materiaalia opittavaksi lyhyessä ajassa. Lisäksi haastavaksi koettiin se, että rajapinnan käytön ymmärtäminen vaati aikaisempaa osaamista tietoturvasta ja etenkin puutteelliset tiedot SSL- ja TLS-protokollista aiheuttivat osallistujille vaikeuksia. (Wijayarathna & Arachchilage, 2018)

Artikkelissa esille nostetut käytettävyysongelmat ovat tyypillisiä tietoturvaan liittyviä haasteita, sillä tietoturvan tekninen toteuttaminen vaatii käytettyjen teknologioiden perusteellista ymmärtämistä ja osaamista sekä monimutkaisten kokonaisuuksien hallitsemista. Kokeneellekin ohjelmoijalle voi olla haastavaa soveltaa esimerkiksi edellä mainittuja protokollia käytännön sovelluksissa, jos aikaisemaa kokemusta aiheesta ei ole.

Sokettiohjelmien kooditoteutuksessa on tietoturvan kannalta otettava huomioon myös muita seikkoja kuin pelkkä tiedonsiirron salaaminen. International Journal of Network Security & Its Applicationsissa vuonna 2013 julkaistussa tapaustutkimuksessa esitellään Java-esimerkkikoodin avulla tietoturvariskejä, joita tässäkin työssä esitettyjen esimerkkiohjelmien kaltaisissa ohjelmissa voi olla (Meghanathan, 2013).

Tutkimuksessa käsitellään tiedostoja lukevaa palvelinohjelmaa, joka palvelee vain yhtä asiakasta kerrallaan. Tutkimuksen esimerkkiohjelma käyttää java.net- ja java.io-kirjastoja kuten tässäkin työssä esitellyt ohjelmat. Palvelin kuuntelee saapuvia yhteyksiä ja vastaanottaa niiltä saapuvia viestejä. Viestin saapuessa sen sisältämän tiedostonimen, tai polun avulla avataan tiedosto, luetaan siitä rivejä ja lähetetään ne yhteyden ottaneelle asiakkaalle. Ohjelma käyttää TCP-soketteja lähettämisessä ja vastaanottamisessa. Lähdekoodin analysoinnissa tutkimuksessa käytettiin Fortify Inc.:n kehittämää työkalua Source Code Analyzer (SCA). Ohjelma analysoi koodia ja siihen asetettujen sääntöjen määrittämien kriteerejen mukaan. Säännöt voivat olla ohjelman valmiiksi tarjoamia, tai käyttäjän itse muokkaamia.

Ensimmäinen tutkimuksessa käsiteltävä haavoittuvuus on se, että esimerkkiohjelmassa käyttäjä voi vapaasti syöttää porttinumeron, johon palvelin kytketään. Tämä voi mahdollistaa hyökkääjän pääsyn sille kuulumattomiin resursseihin järjestelmässä ja mahdollisesti lähettää informaatiota kolmannelle osapuolelle. Haavoittuvuus voidaan

korjata asettamalla rajoitukset sille, mitkä porttinumerot ovat ohjelman käyttäjän valittavissa. Tämä sama parannus voitaisiin tehdä myös tämän työn ohjelmiin, jos niitä haluttaisiin käyttää muutoin kuin havainnolistavissa tarkoituksissa.

Toinen tutkimuksessa havaittu haavoittuvuus on se, että palvelimen vastaanottamaa tiedostonimeä ei tarkisteta vaan mikä vain tiedostonimi kelpaa. Tämä mahdollistaa hyökkääjän kirjoittamisen mihin tahansa kriittiseen tiedostoon näin halutessaan ja jälleen pääsyn sille kuulumattomiin resursseihin. Esimerkki on jälleen yksinkertainen, mutta huolimaton ohjelmointi voi johtaa tämän tyyppisiin haavoittuvuuksiin.

Ohjelmasta löydettiin myös haavoittuvuus virhetulostuksen muodossa. PrintStackTrace()-metodi tulostaa hyvinkin tarkkaa virhetulostusta ohjelman ja järjestelmän toiminnasta, eikä tällaista metodia pidä päästää ohjelman käyttäjän näkyville. Jos tällaista tulostusta käytettäisiin esimerkiksi web-ohjelmassa voisi mahdollinen hyökkääjä saada selville kriittistä tietoa ohjelman, tai sivun toiminnasta ja näin päästä iskemään mahdollisiin heikkouksiin.

Tutkimuksessa esitellään myös yleinen verkkosivuja kohtaan tehtävä palvelunestohyökkäys eli DoS. Tässä tapauksessa DoS-hyökkäyksen mahdollistaa readLine()-metodi, jolle ei aseteta rajoitteita sille, kuinka pitkiä merkkijonoja luettava syöte sisältää. Hyökkääjä voi siis lähettää kuinka pitkän viestin tahansa, mikä taas voi aiheuttaa muistin loppumisen ja ohjelman kaatumisen. Yleisesti onkin hyvä asettaa rajoitteet sille kuinka pitkästi tai tiheästi asiakas voi viestittää palvelimelle, jotta palvelun toiminta voidaan taata.

Viimeinen tutkimuksessa mainittu heikkous liittyy vapauttamatta jätettyihin resursseihin. On mahdollista, että sulkevat close()-metodit jäävät kutsumatta esimerkiksi epäonnistuneen tiedoston lukemisen takia. Hyökkääjä voisi hyödyntää heikkoutta lähettämällä tiedoston, jota ei kyetä lukemaan tai sulkemalla soketin ennen kuin tiedostoa, tai vastaanotettavaa dataa lukevia olioita ehditään sulkea. Tällöin näiden varaamat resurssit jäävaät vappauttamatta, mikä luonnollisesti heikentää järjestelmän käytössä olevien resurssien määrää.

Tutkimuksen sisältämä esimerkkiohjelma on toiminnaltaan yksinkertainen, mutta se silti havainnollistaa monia seikkoja, joita tulee ottaa huomioon ohjelmissa, jotka lähettävät ja vastaanottavat dataa tuntemattomien osapuolien kanssa. Tutkimuksen esimerkit osoittavat myös, että vaikka siirrettävän datan salaaminen on tärkeässä roolissa, ei pidä unohtaa myös itse lähdekoodin tietoturvallisuutta.

3.6 Java-sokettien suorituskyvyn lisääminen

Luvussa 2.5 käsiteltiin sokettien hyödyntämistä hajautetussa järjestelmässä ja ohjelmointikielien sekä tiedovälitystavan vaikutusta datan käsittelyn nopeuteen. Sokettien tehokkuutta hajautetuissa järjestelmissä ja pilviympäristöissä on tutkittu, ja esimerkiksi IEEE:n julkaisema konferenssijulkaisu vuodelta 2007 käsittelee aihetta Javan näkökulmasta.

Javan virtuaalikone (JVM) tukeutuu soketti-yhteyksissä TCP/IP-pinon mukaisiin toimintoihin, mikä voi jossain tapauksissa hukata prosessorin resursseja lähetettävän datan käsittelyssä. Javan socket-kirjastosta onkin tehty tehokkuuden osalta kehitettyjä versioita. Yksi tällainen on esitetty IEEE International Parallel and Distributed Processing Symposiumissa. (Taboada, et al., 2007)

Julkaisussa tehostettiin Javan sokettien sekä virtuaalikoneen datan käsittelyä nopeuttamalla primitiivisen datan muuntamista tavuiksi. Tässä onnistuttiin luomalla uusi tapa datan muuttamiseen suoraa tavuiksi ilman sarjallistamista ja kopioiden luomista, jolloin datan käsittelyn suorituskyky saatiin nousemaan. Ohjelmoija voi korvata tavallisen SocketFactoryn, jota käytetään sokettien luomiseen ohjelman suorituksen aikana, JFSFactoryllä, joka puolestaa hyödyntää nopeampia Java Fast Socketteja (JFS).

Tutkimuksessa vertailtiin JFS:n ja tavallisen Java-soketin viivettä sekä kaistanleveyttä viestin pituuden funktiona tavu- ja kokonaislukutaulukkojen lähetyksessä kahdessa eri järjestelmässä. JFS:ää verrattiin myös Ibis-soketteihin, jotka niin ikään on suunniteltu tehostamaan tavallisten Java-sokettien toimintaa.Toisessa järjestelmässä käytettiin arkkitehtuuria, joka pohjautui natiiveille Javan TCP/IP-soketeille. Tämän järjestelmän tapauksessa vertailtavien sokettien välille ei syntynyt suuria eroja. Rinnakkaislaskentaan paremmin sopivan SCI arkkitehtuuria käyttävän järjestelmän tapauksessa JFS oli tavallisia soketteja ja Ibis-soketteja suorituskykyisempi, sillä TCP/IP-protokollapinon aiheuttamat rajoitteet tiedonsiirrossa eivät tasanneet tuloksia eri sokettien välillä.

Tämän tutkimuksen perusteella TCP/IP-pohjaisen sokettiohjelmoinnin kannalta JFS:n tarjoamat edut eivät ole merkittäviä, jos näkökulmana ovat tavalliset ei niin suorituskykyä vaativat ohjelmat. Kuitenkin tällaiset tiedonsiirtoa ja datan käsittelyä nopeuttavat teknologiat voivat olla hyödyllisiä käyttökohteissa, joissa käsitellään ja analysoidaan suuria datamääriä. Yksi tällainen sovellus on esitetty IEEE:n konferensissa, jossa korkean suorituskyvyn Java-soketteja tutkittiin lääketieteellisen datan tiedonsiirrossa pilvialustalla (Amin, et al., 2012).

Julkaisussa ehdotettu sokettityyppi on niin sanottu High Performance Java Socket (HPJS). Tutkimuksessa mainitaan yhtäläisyydestä jo edellä mainittuun JFS:ään. HPJS:n

kuitenki kerrotaan poikkeavan siinä, että JFS hyödyntää Java Native Interfacea (JNI), kun taas HPJS on puhtaasti Javalla toteutettu helposti saatavilla oleville laitteille. JNI on rajapinta, jonka avulla Javan metodeja saadaaan sulautettua laitteelle, joka käyttää prosessoritasolla esimerkiksi C-kieltä. Tutkimuksessa HPJS:ää hyödyntävä järjestelmä on rakennettu niin, että laitteille asennetaan asetustiedosto, josta selviää tarvittavat tiedot pilvijärjestelmän muista laitteista. Tiedoston avulla prosessit jakavat toisillen omat uniikit ID:nsä, joilla soketit tunnistavat tiedonsiirrossa lähde- ja kohde-soketit. Datan käsittelyssä HPJS toimii ainoastaan tavujen avulla käyttäen protokollaa, jossa data kopioidaan ainoastaan kerran, mutta JFS:stä poiketen datatyyppiä ei missään vaiheessa muuteta. (Amin, et al., 2012)

HPJS ei ole valmis teknologia, ja tutkimuksessa ei testattu HPJS:n toimintaa laajemmassa mittakaavassa tai korkeilla järjestelmän rasitustasoilla. Se on kuitenkin mielenkiintoinen tutkimus- ja sovelluskohde sokettien hyödyntämiseen muussa kuin tavanomaisessa TCP/IP-kontekstissa.

4. YHTEENVETO

Tämän tutkielman aikana huomattiin, että TCP/IP-pohjainen sokettiohjelmointi onnistuu Javan tarjoamilla kirjastoilla vaivattomasti, vaikka ohjelmien toteuttaja ei olisi vielä erityisen kokenut ohjelmoija. Muita ohjelmointikieliä käsitellessä kuitenkin huomattiin, että Java ei suinkaan ole ainoa ja absoluuttisesti paras kieli sokettiohjelmointiin, vaan myös esimerkiksi C++ ja Python tarjoavat hyvät työkalut sokettien hyödyntämiseen. Kaiken kaikkiaan kielivalintaa tehdessä on hyvä ottaa huomioon, mitä ominaisuuksia ohjelmalta haluaa, mikä on ohjelmoijan taitotaso, sekä minkälainen ohjelmointitausta tekijällä on. Lisäksi havaittiin, että vaikka sokettien toteutuskielellä ei ole väliä niiden keskinäisen kommunikaation onnistumisen kannalta, on niiden toteutuskielillä merkitystä, kun mittarina on järjestelmien välisen tiedonsiirron tehokkuus.

Tässä työssä pääaiheena ollut Java todettiin oivaksi kieleksi aloittaa sokettiohjelmointi ja huomattiin, että kielelle aiheesta löytyy kattavasti kirjallisuutta, sekä verkkosivuja perehtymiseen. Työssä esitetyt yksinkertaiset esimerkkiohjelmat osoittivat, että Javan dokumentaation, sekä muun aiheeseen liittyvän materiaalin avulla onnistuttiin luomaan toimivia esimerkkejä, niin TCP-, UDP- kuin ryhmälähetyksen eli multicast-yhteyden osalta. On toki otettava huomioon, että esitetyt ohjelmat ovat toiminnoiltaan suppeita ja ne luotiin ainoastaan havainnollistamaan Javan eri yhteyksiin tarkoitettuja soketteja. Vaihtoehtoinen lähestymistapa aiheeseen voisi olla esimerkiksi yhden laajemman vahvasti soketteja hyödyntävän ohjelman luonti ja sen analysointi.

Tutkielmassa sivuttiin myös verkko-sokettien tietoturvaa ja Javan tähän tarjoamia työkaluja. Sokettien välittämän datan salaaminen toteuttaminen onnistuu Javalla, mutta kuten esitellyssä tutkimusartikkelissakin huomattiin, se ei ole jo kokemusta omaavallekaan ohjelmoijalle välttämättä helppoa. Salaamisen lisäksi havaittiin, että itse ohjelmakoodin tietoturvaan on kiinnitettävä huomiota, ja huolimaton ohjelman toteutus voi johtaa monenlaisiin haavoittuvuuksiin.

Javalla sokettien tehokkuutta on mahdollista lisätä hyödyntämällä Javan peruskirjaston ulkopuolisia rajapintoja ja teknologioita. Tällaisten teknologioiden merkitys tämän työn kontekstissa huomattiin kuitenkin vähäiseksi, mutta laajoja datamassoja käsittelevissä sekä tehokkuutta vaativissa järjestelmissä niille on käyttökohteensa.

Koska sokettien tietoturvan käsittely jäi tässä tutkielmassa pintapuoliseksi, voisi hyvä jatkoselvityksen kohde ollakin SSL- ja TLS-protokollat ja miten niitä hyödynnetään sokettien yhteydessä. Lisäksi soketteja hyödyntävien ohjelmiem ja järjestelmien yleiseen

turvallisuuteen voisi syventyä tarkemminkin. Toinen mielenkiintoinen jatkoselvitysaihe on tässäkin työssä mainittu WebSocket-protokolla ja sen hyödyntäminen web-sovelluksissa.

5. LÄHDELUETTELO

Amin, M. B. ym., 2012. *High performance Java sockets (HPJS) for scientific health clouds.* Beijing, IEEE.

Carmouche, J. H., 2006. *IPsec Virtual Private Network Fundamentals.* s.l.:Cisco Press.

Deering, S., 1989. *Host Extensions for IP Multicasting.* [Online] Available at: https://www.rfc-editor.org/rfc/rfc1112

Dmitrović, S., 2021. *Modern C for Absolute Beginners: A Friendly Introduction to the C Programming Language.* s.l.:Apress.

Doraswamy, N. & Harkins, D., 2003. *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks, Second Edition.* s.l.:Pearson.

Fette, I. & Melnikov, A., 2011. *The Web Socket Protocol.* [Online] Available at: https://www.rfc-editor.org/rfc/rfc6455

Gay, W. W., 2021. Chapter 1. Introducing Sockets. Teoksessa: *Linux Socket Programming by Example.* s.l.:Que.

Gay, W. W., 2021. Chapter 1. Introducing Sockets. Teoksessa: *Linux Socket Programming by Example.* s.l.:Que.

GeeksforGeeks, 2023. *User Datagram Protocol (UDP).* [Online] Available at: https://www.geeksforgeeks.org/user-datagram-protocol-udp/ [Haettu 2023].

GeeksforGeeks, 2023. *What is Local Host?.* [Online] Available at: https://www.geeksforgeeks.org/what-is-local-host/ [Haettu 2023].

Gokhale, A. & Mandir, B. S., 2021. Types of socket programming and socket designs: A Review. *Volume 183 – No. 32*, October, p. 34.

Harold, E. R., 2013. Chapter 1. Basic Network Concepts. Teoksessa: *Java Network Programming, 4th Edition.* 4. toim. s.l.:O'Reilly Media, inc..

IBM, 2021. *What is distributed computing.* [Online] Available at: https://www.ibm.com/docs/en/txseries/8.2?topic=overview-what-is-distributed-computing
[Haettu toukokuu 2023].

IBM, 2023. *Transmission Control Protocol.* [Online] Available at: https://www.ibm.com/docs/en/aix/7.2?topic=protocols-transmission-control-protocol
[Haettu 2023].

Java T Poimt, ei pvm *Javatpoint.* [Online] Available at: https://www.javatpoint.com/link-state-routing-algorithm

Java T Point, ei pvm *Border Gateway Protocol.* [Online] Available at: https://www.javatpoint.com/border-gateway-protocol

Java T Point, ei pvm *Javatpoint.* [Online] Available at: https://www.javatpoint.com/distance-vector-routing-algorithm

Javapoint, ei pvm *SSH Meaning| SSH Protocol Definition.* [Online] Available at: https://www.javatpoint.com/ssh-meaning [Haettu Maaliskuu 2023].

JavaTpoint, 2021. *C Header Files.* [Online] Available at: https://www.javatpoint.com/c-header-files [Haettu 2023].

Jin, X. & Chan, S.-H. G., ei pvm 3 Unstructured P2P Networks for File Sharing. Teoksessa: *Handbook of peer-to-peer networking.* s.l.:s.n.

Kumar, S., 2023. *Peer to Peer Networks.* [Online].

Li, M. & Lee, C.-H., 2020. Design and Analysis of CQI Feedback Reduction Mechanism for Adaptive Multicast IPTV in Wireless Cellular Networks. *IEEE Transactions on Vehicular Technology,* 69(2), pp. 2008-2020.

Liu, D., Barber, B. & DiGrande, L., 2009. CHAPTER 6 - Implementing RIP, Version 2. Teoksessa: *Cisco CCNA/CCENT Exam 640-802, 640-822, 640-816 Preparation Kit.* s.l.:s.n.

Maata, R. L. R., Cordova, R., Sudramurthy, B. & Halibas, A., 2017. *Design and Implementation of Client-Server Based Application Using Socket Programming in a Distributed Computing Environment.* Coimbatore: s.n.

Malleswara, R. & Pattamsetti, R., 2017. *Distributed Computing in Java 9.* s.l.:Packt Publishing.

Meghanathan, N., 2013. Source code analysis to remove security vulnerabilities in Java socket programs: a case study. *International Journal of Network Security & Its Applications (IJNSA),* 5(1).

Microsoft, 2021. *Getting started with Winsock.* [Online] Available at: https://learn.microsoft.com/en-us/windows/win32/winsock/getting-started-with-winsock [Haettu 2023].

Minh, N. H., 2019. *Java UDP Client Server Program Example.* [Online] Available at: https://www.codejava.net/java-se/networking/java-udp-client-server-program-example [Haettu Maaliskuu 2023].

Misra, S., 2017. 3.4 Path vector routing. Teoksessa: *Network Routing : Fundamentals, Applications, and Emerging Technologies.* s.l.:John Wiley & Sons, Incorporated.

Oracle, 2020. *Class MulticastSocket.* [Online] Available at: https://docs.oracle.com/javase/7/docs/api/java/net/MulticastSocket.html [Haettu 2023].

Oracle, 2020. *docs.oracle.com.* [Online] Available at: https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html [Haettu 2023].

Oracle, 2023. *Java Secure Socket Extension (JSSE) Reference Guide.* [Online] Available at: https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html#Introduction

Pérez, A., 2014. *Network Security.* Somerset: John Wiley & Sons, Incorporated.

Postel, J., 1983. *Quote of the Day Protocol.* [Online] Available at: https://datatracker.ietf.org/doc/html/rfc865.html

Python Software Foundation, 2023. *socket — Low-level networking interface.* [Online] Available at: https://docs.python.org/3/library/socket.html [Haettu 2023].

Ruediger, G., Martin, K. & Inmaculada, M.-B., 2016. *Local programming language barriers in stream-based systems.* Messina, Institute of Electrical and Electronics Engineers.

Salim, J., Khosravi, H., Kleen, A. & Kuznetsov, A., 2003. *Linux Netlink as an IP Services Protocol.* [Online] Available at: https://www.rfc-editor.org/rfc/rfc3549 [Haettu Toukokuu 2023].

Schildt, H., 2018. Chapter 1 The History and Evolution of Java. Teoksessa: *Java: The Complete Reference. 11th Edition.* s.l.:McGraw-Hill.

Sharma, V. K., Kumar, V., Sharma, S. & Pathak, S., 2021. Introduction to Python Programming. Teoksessa: *Python programming: A Practical Approach.* s.l.:Chapman and Hall/CRC.

SSL2BUY, ei pvm *SSH vs SSL/TLS – What are Differences and Similarities?.* [Online] Available at: https://www.ssl2buy.com/wiki/ssh-vs-ssl-tls

Taboada, G. L., Tourino, J. & Doallo, R., 2007. *High Performance Java Sockets for Parallel Computing on Clusters.* Long Beach, CA, USA, IEEE.

Van Winkle, L., 2019. Putting it together. Teoksessa: *Hands-On Network Programming with C.* s.l.:Packt Publishing.

Vihervaara, J., ei pvm *Computer Networking II.* s.l.:s.n.

Wararkar, P. &. K. N. &. R. V. &. M. Y. &. B. Y., 2016. *Resolving Problems Based on Peer to Peer Network Security Issue's.* s.l., Procedia Computer Science.

Wijayarathna, C. & Arachchilage, N. A. G., 2018. Why Johnny can’t develop a secure. *Computers & Security,* Issue 80, pp. 54 - 73.

Winett, J. M., 1971. *Definition of a socket.* [Online] Available at: https://www.rfc-editor.org/rfc/rfc147