Stanislav Savolainen

TCP/IP-sokettiohjelmoinnin sovelluslogiikka

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

19.5.2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Stanislav Savolainen TCP/IP-sokettiohjelmoinnin sovelluslogiikka 33 sivua + 1 liite 19.5.2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Simo Silander Yliopettaja Erja Nikunen

Insinöörityö kertoo, miten TCP-sokettien sovelluslogiikka rakentuu ja miten sitä voi käyttää ohjelmistoprojektissa. Voidaan tehdä omia verkkosovelluksia esimerkiksi web-selaimia, http-palvelimia, monipelejä, chat-ohjelmia, tiedostojakeluohjelmia ja muita sovelluksia.

Työ näyttää myös sen, että verkko-ohjelmoinnin opettelun aloitusvaihe ei ole kovin vaikeaa eikä tarvitse tietää paljon tietoliikenteestä aloittaakseen yksinkertaisten omien verkko-ohjelmien teon. Kerron askel kerrallaan, miten voi tehdä yksinkertaisen ohjelman TCP-protokollalla, jossa on käyttäjän ja palvelimen välillä tiedonsiirto ja esitän sen sovelluslogiikkana.

Työ alkaa yleistiedolla ja seuraavaksi siirrytään käytäntöön. Tämän jälkeen kerrotaan tekemästäni projektista esittämällä sovellus. Sen jälkeen kerrotaan, miten voidaan ohjelmoida oma yksinkertainen TCP-protokollasovellus, jossa käytetään TCP-soketteja. Siitä näytetään lähdekoodi ja selitetään sen toimivuus huolellisesti.

Tein chattailusovelluksen Javalla, mikä käyttää TCP-protokollaa viestien tiedonsiirtona sisäverkossa. Kyseessä on yhden palvelimen ja usean käyttäjän vuorovaikutus ohjelma. Käyttäjät kommunikoivat keskenään palvelimen kautta.

Avainsanat	Java TCP -Socket , Winsock, WSAStartup, Berkeley socket

Author(s) Title Number of Pages Date	Stanislav Savolainen TCP/IP Socket Programming Application Logic 33 pages + 1 appendix 19 May 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Simo Silander, Senior Lecturer Erja Nikunen, Principal Lecturer

This thesis is about TCP-socket application logic and includes material on how to use it in programming projects. It can be used for network application for example; web browser, http server, multiplayer games, chatting program, file sharing program and other network applications.

The study also shows that beginning to learn network programming is not that hard and does not require additional telecommunication knowledge. The study explains step by step how to make a simple TCP-protocol program where the server and client program communicate with each other and introduces the application logic.

The study begins with basic information and continues with a user guide on how to use the TCP-protocol. After that the program and how it works is introduced. Then the study explains how to make a TCP-program using TCP-sockets, as well as the program source code and how it works in detail.

I did chat-application in Java, which use the TCP-protocol for transferring messages in private networks. This is server and multi-users interaction program. Users communicate with each other's via the server.

Keywords	Java TCP-Socket , Winsock, WSAStartup, Berkeley socket

Sisällys

Lyhenteet

IP	Internet Protocol, Internet-yhteydelle tarkoitettu yhteysprotokolla.
IPV4	32-bittinen ip-osoite, käytetään yhteysmuodostukseen laiteosoitteena.
IPV6	64-bittinen ip-osoite, voidaan käyttää saman tarkoitukseen kuin IPV4:ää.
PORTTI	16-bittinen numero, joka on tarkoitettu laitteen yhteyden kuvaamiseen.
TCP	Transmission Control Protocol, tiedonsiirtoprotokolla, joka käyttää IP-osoitetta ja porttia yhteyden muodostukseen laitteisiin.
UDP	User Datagram Protocol, protokolla, joka siirtää datagram-pakettidataa verkon yli. Samoja ja erilaisia piirteitä TCP-protokollaan verrattuna.
ISP	Internet Service Provider, internetyhteyden operaattori, joka tarjoaa internetiin ip-osoitteen ja yhteyden nettiin.
Topologia	Verkon kytkösarkkitehtuuri eli miten verkossa olevat laitteet on kytketty keskenään. Laitteiden väliset kytkökset voidaan esittää kuvamuodossa. (Topology englanniksi).
HTTP	Hypertext Transfer Protocol, protokolla, joka on tarkoitettu web-sivujen tiedonsiirtoon.
HTTPS	Hypertext Transfer Protocol Secure. Suojattu HTTP-protokolla, joka käyttää SSL/TLS-protokollaa tiedonsiirtoon.
SSL	Secure Socket Layer TCP-protokollasta muodostettu salausprotokolla, joka on tarkoitettu turvallista TCP-soketti-yhteyttä varten.
TLS	Transport Layer Security. Tulee SSL-protokollasta.
MULTIPLEXING	Kanavanvaihtokytkin, voidaan käyttää ei-lukkiutuvissa yhteyksissä usean yhteyden käsittelyyn saman prosessin kautta.

JRE Java Runtime Enviroment, Java-virtuaalikone, jossa Java-sovellukset ajetaan.

JDK Java Development Kit, Java-kehitystyökalu, jota käytetään Java-ohjelmoinnissa.

WINSOCK Windows-käyttöjärjestelmän sokettiohjelmointi.

SOKETTI Oliomuuttuja, joka toimii kytkimenä tiedonsiirtoprotokollien ohjelmoinnissa. Englanniksi ”SOCKET”.

STRIIMI Tietovirta. Tarkoitetaan tiedonsiirtoprosessia.

PAKETTI Tarkoittaa tavumerkkijonoa, joka lähetetään yhtenä tietovirtaprosessina eli striiminä. Suuri tietokokonaisuus muodostuu useasta yksittäisistä paketeista. Käytän sanoja ”data”, ”paketti” ja ”pakettidata”, joilla tarkoitan samaa asiaa. Jokainen yksittäinen paketti sisältää päätelaitteen yhteystiedot soketti-oliomuuttujan avustuksella.

KÄYTTÄJÄ Tässä tekstissä tarkoitetaan aina TCP-sovellusohjelmiston käyttäjää.

1 Johdanto

Insinöörityön tavoitteena on kertoa TCP/IP-protokollan Java- ja C/C++-ohjelmointimenetelmästä ja käyttää niitä omassa ohjelmistoprojektissani. Työn tutustumisen aikana selviää, että oman yksinkertaisen sovelluksen tekeminen ei ole kovin hankala asia eikä tarvitse olla tietoliikenteen suuri asiantuntija, mutta tehokkaan ohjelman tekeminen vaatii enemmän perehtymistä näihin asioihin. Kerron askel kerrallaan perusasioista yksinkertaisen sovelluksen tekemiseen asti ja työn aikana pohdin erilaisia ratkaisumenetelmiä tai ongelmatilanteita.

Työ on tarkoitettu kokeneille ohjelmointiasiantuntijoille, joille ainakin perusohjelmointi on hallussa. Oletan, että ohjelmoinnin peruskäsitteet kuten muuttujat, try-catch-poikkeuksien hallinta, olio-ohjelmoinnin asiat, striimit (input, output), säikeet, MVC-mallin ja graafiset käyttöliittymät tapahtumakuuntelijoineen ovat tuttuja. Näistä asioista en puhu erikseen, koska oletan, että työn lukija tietää ne. Voidaan siis tehdä seuraavan askeleen oppimiseen kokeilemalla luoda TCP-sokettiolion, jolla on parametrimuuttujina annettu yhteystiedot. Tästä kerron myöhemmin, ja nyt tutustutaan TCP/IP-perusasioihin katsomalla tietoliikenteen asioita. Työssä ei siis käsitellä lainkaan IP-modulointia eikä reitityksen algoritmeja, koska ne eivät liity tähän aiheeseen, mutta ovat erittäin tärkeitä tietoliikenteen kannalta. TCP-protokollaan ohjelmoinnin aloitusvaiheessa ne eivät ole tarpeellisia. Tarvitaan siis yleistietoa TCP/IP-protokollasta, jota edellytetään ohjelmoinnissa.

Mainitsen myös huonon ja hyvän sovelluslogiikan tietoturvariskit, SSL/TLS-protokollan toiminnan perustiedot sekä sen, miten se liittyy aiheeseen. Työ alkaa perusasioista kertomalla, mikä on TCP ja miten sitä käytetään käyttäjän näkökulmasta. Tämän jälkeen kerron, missä olemassa olevissa toteutuksissa käytetään TCP-protokollaa, ja näytän, millaisen sovelluksen sain itse ohjelmoitua. Sen jälkeen esitän toteutukselle lähdekoodin ja kerron yksityiskohtaisesti, miten voi tehdä oman tiedonsiirtosovelluksen TCP-protokollaa käyttäen.

2 Yleistä tietoa TCP-protokollasta

2.1 Mikä on TCP-protokolla?

TCP eli Transmission Control Protocol on tiedonsiirtoprotokolla, jonka kautta voi välittää tavumerkkijonon käyttämällä käyttäjäsovelluksessa voimassa olevan palvelimen ip-osoitetta ja porttinumeroa. Kyseessä on sisäverkon ja internetverkon tärkein tiedonsiirtoprotokolla, jonka kautta pakettidata saapuu perille, eli käytännössä kaiken tiedon voi tavuina välittää TCP-protokollalla laitteesta laitteille (laiteella tarkoitan omassa työssäni esimerkiksi pc:tä, matkapuhelinta, tablet-pc:tä tai jotakin muuta).

Verkosta ”striiminä” eli tietovirtana tulevalle pakettidatalle täytyy määritellä sovelluslogiikka, miten tulevaa tavumerkkijonoa käsitellään. Näin saadaan tarvittava tietokokonaisuus pelkistä numeroarvoista. Sen takia on olemassa header-data jokaisen yksittäisen paketin sisällä, mikä antaa infon saapuvan tiedon ominaisuudesta ja sen sisällöstä, eli tiedetään, mihin sovellukseen ja kuinka paljon dataa kulkee. Voidaan todeta, että ”paketti” tässä tekstissä tarkoittaa tietyn pituista tavumerkkijonoa, joita on useita. Verkkosovellus lähettää useita paketteja molempiin suuntiin eli lähettäjältä vastaanottajalle ja vastaanottajalta lähettäjälle. [2; 3.]

2.2 Miten TCP-protokollaa käytetään yhteyden muodostamiseen?

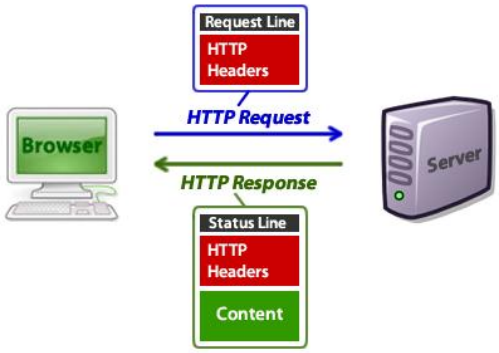
TCP-protokollan käyttäjä eli jonkin käyttäjäsovellusohjelman tarvitsee antaa kaksi parametria palvelimen ip-osoitetta ja porttinumeron yhteyttä varten. Palvelimen pitää olla samanaikaisesti valmis vastaanottamaan uutta yhteyttä käyttäjältä; muuten yhteys ei onnistu. Lisäksi käyttäjän antamien ip-osoitteiden ja porttinumerotietojen täytyy täsmätä palvelimen tietojen kanssa, jonne yhteys muodostetaan.

2.3 Mitkä sovellukset käyttävät TCP-protokollaa?

Monipelissä (Multiplayer game) pelipalvelin osaa käsitellä tulevan datan parametreina pelaajilta esimerkiksi sijaintikoordinaatit, pisteet, skillit, viestit toiselle pelaajalle ja muut komennot. Palvelin käsittelee usean verkkopelaajan antamia tietoja ja suorittaa ohjelmoitua sovelluslogiikkaa. Sen jälkeen palvelin voi lähettää tietylle pelaajalle hänelle tarkoitetut tiedot.

Pelin graafista osuutta ei tarvitse siirtää palvelimelle tai palvelimelta, vaan välitetään ainoastaan pelaajaa koskevat tiedot. Ohjelmoitu sovelluslogiikka sekä palvelin- että asiakaspäässä hoitaa pelin etenemistä molemmille osapuolille. Tiedonsiirron pelaajan ja palvelimen välillä hoitaa TCP-protokolla (hyvä palvelin osaa myös tunnistaa virheellisen datan).

HTTP–palvelin käyttää HTTP-protokollaa web-sovellusten tiedonsiirtoon. Kyseinen protokolla on OSI-mallin 7. kerrokseen pohjautuva sovellusprotokolla, joka käyttää alimmalla OSI-kerroksella TCP-protokollaa apuna tiedonsiirtoon. Yksinkertaisemmin sanottuna tässä tilanteessa kahden ohjelman välillä, jossa asiakaspäässä on tavallinen selain ja palvelimen päässä on HTTP-palvelin, välitetään HTTP-pakettidata, joka on todellisuudessa TCP-protokollan kautta välitetty tavallinen merkkijono, joka sisältää HTTP-pyyntöjä asiakkaalta ”**http-request**” ja vastauksia niihin palvelimelta ”**http-response**”. [11; 12.]

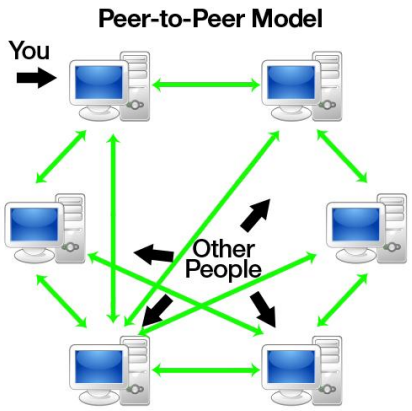


Kuva 1. Kuva näyttää HTTP-palvelimen kommunikointia selaimen kanssa lähettämällä ”HTTP–Request”-pyyntöjä ja vastaanottamalla ”HTTP-Resoponse”-vastauksia palvelimelta

Http-request sisältää globaalit muuttujat **get**, **post** , session ”**session**” ja evästeet ”**cookies**”, jotka tulevat käyttäjän selaimesta palvelimeen pyyntönä. Palvelin osaa vastata niihin ”http-requestilla” ja käytännössä myös pystyy välittämään html-dataan merkkijonon. Käyttäjän www-sivun sisältö siis voi muuttua palvelimen toimesta.

HTTP-protokolla varaa oletuksena TCP-protokollan porttinumeron 80, jonka kautta tiedot kulkevat http-sovelluslogiikka noudattaen TCP-protokollan kautta. Http–request- ja HTTP-response-pakettidataa sisältää http-headerin, globaalit muuttujat (käyttäjältä) ja html-datan (palvelimelta), mikä on käytännössä myös tavumerkkijono, jonka TCP-protokolla käsittelee. [11; 12.]

Tiedostojakelupalvelusovellukset kuten e-mule, Torrent, Kazzaa ja winMx käyttävät yleensä UDP-protokollaa, koska se on nopeampi vaihtoehto tiedonsiirrolle kuin TCP, mutta paljon virhealttiimpi, koska pakettidatan kuittausta perille saapumista ei tarkisteta. Tällaisissa sovelluksissa käytetään **peer-to-peer**-topologia eli ei ole olemassa asiakas-palvelin-yhteysperiaatetta, vaan käyttäjät on yhdistetty keskenään, ja ne vaihtavat pakettidataa. Yhden käyttäjän yhteyden katkaisu sulkee vain yhden yhteyden eli käytännössä katkaisee ”download/upload”-prosessin käyttäjien välillä. [14; 20.]



Kuva 2. Kuvassa nähdään Peer-to-Peer-topologia. Sen tärkeä periaate on se, että ei ole palvelin-käyttäjä-ratkaisua, vaan kaikki käyttäjät voivat kommunikoida keskenään. [14.]

Näille sovelluksille voi tehdä TCP-protokolla-vaihtoehdon. Voidaan kuvitella mini-palvelin jokaisella käyttäjällä, joka on tarkoitettu lähetysominaisuutta varten. Käyttäjä jakelee dataa toiselle käyttäjille oman mini-palvelimien avulla usean käyttäjän kesken, ja toisella käyttäjällä on sama järjestelmä. Näin yhden käyttäjän yhteyden katkaisu ei vaikuta kovin merkitsevästi koko järjestelmään toimivuuteen, kun kyseessä on usean käyttäjän verkko. [14; 20.]

”Torrent download seed” eli latauspisteet voi ymmärtää niin, että kyseinen käyttäjä on valmis lataamaan jonkin tiedoston kokonaisena. Tällöin hän voi jakaa latauslinkin muiden torrent-käyttäjien kanssa. Käyttäjä saa tiedoston palasina, mikä on vain osa tiedostosta. Kun kaikki tämän tiedoston osat on saatu, sitä voi käyttää. Tiedosto siis voi sisältää kaikenlaista mediamateriaalia tai sovelluksen. Muut käyttäjät voivat jakaa tämän tiedoston osia muiden käyttäjien kanssa, ja jonkin materiaalin löytämiseksi voidaan käyttää hakukonetta, jotta löytää kyseisen tiedon latauspisteen kautta. Näin tiedostojenjakeluohjelmat toimivat. [17.]

FTP eli File Transfer Protcol. Tiedostojakeluprotokolla on tarkoitettu tiedostojen tiedonsiirtoon. Käyttäjä voi ladata tiedon palvelimelta. Myös autentikointi on mahdollista. FTP on esitetty sovelluskerroksessa OSI-mallin mukaan, mikä käyttää TCP-protokollaa tiedonsiirtoon. [10; 18.]

SMTP eli Simple Mail Transfer Protocol. Käytetään sähköpostipalvelimen tiedonsiirtoprotokollana. SMTP on esitetty sovelluskerroksessa OSI-mallin mukaan ja käyttää TCP-protokollaa tiedonsiirtoon. [10; 18.]

3 Yleinen teoriatieto

3.1 TCP/IP-viitemalli ja OSI-malli

TCP-protokolla muodostuu useasta loogisesta kerroksesta, mikä esitetään TCP/IP-viitemallina. Tämä viitemalli on oikeastaan OSI-mallin, jolla on seitsemän eri loogista kerrosta, jotka kuvaavat TCP/IP-toiminnan eri tasoilla. Ohjelmointivaiheessa voidaan osittain vaikuttaa näiden kerrosten toimivuuteen.

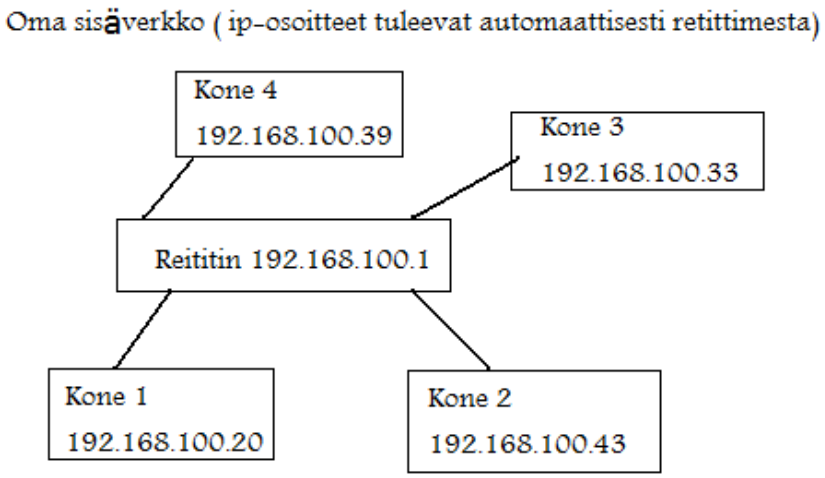
Kuva 3. Kuva esittää OSI-mallin. [10.]

OSI-mallin ja viitemallin mukaan TCP-protokolla on esitetty kuljetuskerroksessa (4. kerros). Tämän kerroksen tärkeä tehtävä on valvoa pakettidatan perille pääsyä lähde- ja kohdelaitteeseen. TCP toteuttaa **ruuhkahallinnan**, mikä käytännössä tarkoittaa ylikuormitetun verkon pakettidatan hallintaa järjestyksessä. Tämä aiheuttaa tietysti aikaviiveen tiedonsiirrossa, kun jokainen paketti on jonottamassa omaa lähetystä eikä voi-

daan siirtyä seuraavan pakettidatan käsittelyyn, kunnes varmistetaan, että tämä paketti on päässyt perille. [9; 10.]

Huomataan että UDP-protokolla on esitetty myös kuljetuskerroksessa. Sen käyttäytyminen **ruuhkatilanteessa** on sellainen, että se pudottaa pois pakettidataa ylikuormitetusta verkossa. Tämä aiheuttaa pakettidatan katoamisen, eli tiedonsiirtolaatu ei ole yhtä hyvä, mutta aikaviive on merkitsevästi pienempi. UDP ei jää odottamaan yhden pakettidatan saapumista perille vaan siirtyy käsittelemään seuraava pakettia. [9; 10.]

OSI-mallin kuljetuskerroksen alemmilla tasoilla on ainakin kolme loogista kerrosta. Ilman niitä TCP-protokolla ei toimisi. Nämä kerrokset ovat fyysinen kerros siirtoyhteyskerros ja verkkokerros. Kun puhutaan fyysisestä kerroksesta, niin tarkoitetaan verkkosovittimen laitetta, johtoja ja muita ”raudan” varusteita, joita käytetään reitittävän ip-osoitteen muodostukseen, mutta itse ip-osoite ei vielä esiinny tällä kerroksella. Seuraavana on siirtoyhteyskerros, joka on toinen kerros OSI-mallissa. Sen tarkoitus on verkkosovittimen MAC-osoitteen esitys verkkokortin yhteystyypin määrittely esimerkiksi langallinen ”ethernet” tai langaton ”wlan 802.11”-yhteystyyppi, virhekäsittely, synkronointitoimenpiteet ja muut verkkosovittimessa tapahtuvat prosessit. Nyt päästään verkkokerrokseen, joka on kolmas kerros OSI-mallissa. Tässä kerroksessa esitetään ip-protokolla ja sitä reitittäviä osoitteita. Puhutaan siis reitittimestä saapuvista ip-osoitteista. Kun sisäverkossa on useita laitteita, jotka on kytketty keskenään tämän reitittimen kautta, niin reititin yleensä jakaa automaattisesti lähiverkon osoitteita. Jokaista laitetta varten on oma sisäverkon ip-osoite, jonka voi käyttää TCP-protokollaan tiedonsiirtoon. Nämä kerrokset siis toimivat laitetasolla ja niiden toimivuuteen voi vaikuttaa ainakin verkkosovittimen ajuriohjelmistolla. [18.]



Kuva 4. Kuvassa näkyy IPV4-ip-osoitteista muodostuva sisäverkko.

OSI-mallissa on vielä muutama kerros kuljetuskerroksen yläpuolella. Ne ovat istuntokerros, esitystapakerros ja sovelluskerros. Näillä kerroksilla ei ole vaikutusta TCP-protokollaan, vaan TCP-protokolla voi vaikuttaa niiden suoritukseen. Hyvänä esimerkkinä on sovelluskerroksessa toimivat protokollat: HTTP, FTP, SMTP ja muita protokollia. Mainitut protokollat ovat kansainvälisesti standardoituja protokollia, jotka käyttävät TCP-protokollaa tiedonsiirtoon, ja niiden sovelluslogiikka on toteutettu ohjelmallisesti. Tämä tarkoittaa, että voidaan kehittää oman sovelluskerroksessa toimivan protokollan, koska voidaan vapaasti ohjelmoida tulevan ja lähtevän datan TCP-protokollan kautta. [9; 10 ; 18.]

3.2 SSL/TLS-protokolla (Suojatut yhteydet)

SSL eli Secure Socket Layer on Netscape Navigaattorin kehittämä salausprotokolla, joka on tarkoitettu turvallisempaan yhteydenmuodostukseen ja kryptattuun tiedonsiirtoon. Netscape on luopunut omasta projektista **IEFT**:lle eli ”Internet Engineering Task Force”-yritykselle, joka jatkoi sen kehitystä TLS-nimellä eli Transport Layer Security, ja sille on olemassa jo monta eri standardia.

SSL/TLS-suojausmenetelmä on pohja HTTPS-protokollalle eli suojattu versio HTTP-protokollasta. HTTPS on siis ”Hypertext Transfer Protocol Secure” salasanavarmennuksilla, SSL-sertifikaattit kuten ”X.509” hyväksyy automaattisesti yhteyden, kun käyttä-

jällä löytyy kyseinen sertifikaatti, joka sisältää tarvittavan tiedon. Tämä HTTPS-protokollan käyttö ja standardi riippuu aika paljon palvelimesta ja käyttäjän selaimesta. Sen turvallisuutta ei voida 100 %:n varmuudella taata, mutta sillä voidaan vaikeuttaa tiedon pääsy vääriin käsin. Joudutaan päivittämään SSL-versiota tietoturvallisista syistä, koska havaitaan, että joku asia ei ole tarpeeksi turvallinen. [15.]

SSL-kehitys jatkuu kuitenkin monissa ohjelmistoprojektissa ainakin Javassa (stanardi, JSSE) tai C/C++:ssa (OpenSSL) , ja monissa muissa ohjelmistokielissä voi ohjelmoida oman SSL-protokollan kautta kulkevaa tiedonsiirtoa. Sen voi käyttää tavallisen TCP-protokollan korvikkeeksi, jolloin TCP-sokettien toiminnallisuus korvataan SSL-sokettien toiminnoiksi ja muutetaan hieman oman ohjelmiston sovelluslogiikkaa yhteensopivaksi niin, että käyttäjän ja palvelimen päissä on molemmilla SSL-protokolla eikä TCP. [25.]

Javassa 1.4 lähtien SSL-ohjelmointi kuuluu Javan ohjelmistostandardiin. Javan ohjelmointiperiaate on sama kuin TCP-protokollalla, mutta sen lisäksi tarvitaan avainpariyhdistelmä uuden yhteyden muodostamiseksi, puhutaan siis salasanatoteutuksesta, jonka ymmärtää palvelin ja käyttäjä. Käyttäjä antaa vahvistuksen yhteydenmuodostuksen aikana, jotta yhteys toimisi. Ohjelmoinnin vaiheessa puhutaan SSLSocket-, SSLSocketFactory-, SSLServerSocket-, SSLServerSocketFactory- ja SSLSession-muuttujista. Lisäksi siihen kuuluvat sertifikaatti-toiminnallisuus, mikä esitetään parametrina SSL-sokettiolioille. Voidaan siis todeta, että sertifikaatti on suojausmenetelmä uutta yhteyttä varten. SSL-protokolla on TCP-protokollasta tehty suojattu protokolla, jonka toimintaperiaate on sama lukuun ottamatta sertifikaatteja ja suojausmenetelmää. SSL-käyttää myös IP-osoitetta, porttinumeroa ja tekee omat soketit yhteyttä varten, joihin kuuluvat suojausmenetelmät parametreina. [25.]

Koodarilla on mahdollisuus valita, kumpi protokolla käyttää yhteyden muodostamiseen TCP:tä, UDP:tä vai SSL/TLS:ää tai joitakin muita. Palvelimen ja käyttäjän koko sovelluslogiikka ei tarvitse muuttaa, jos aikoo vaihtaa sovelluksessa käytetyn TCP:n SSL:ksi, niin voi muuttaa pelkästään protokollan koskevat tiedot eli sokettioliot, kirjastolaajennuksia ja muita sitä koskevia tietoja. Tämän takia suosittelen noudattamaan MVC-mallia tämän kaltaisissa projekteissa, jolloin projektia voi päivittää ja lisätä uusia ominaisuuksia.

Kerron myös uutta tietoa OpenSSL:n haavoittuvuudesta, mikä löytyi OpenSSL-versiossa 1.0.1, missä ”Heart Bleed”-ylivuotobugi vaarantaa käyttäjien suojattua yhteyt-

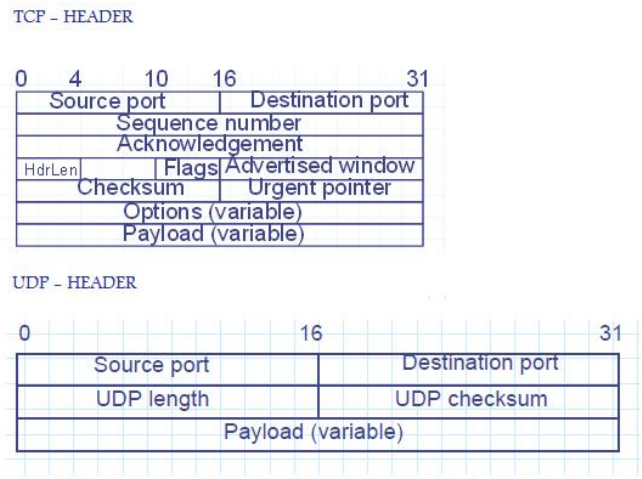
tä. Tämä bugi on aika monessa ohjelmistototeutuksessa, mikä vaarantaa merkittävästi käyttäjien ja palvelimien tietoturvaa. Bugi on korjattu OpenSSL-yhteisössä, mutta useat ohjelmistot käyttävät vielä edellistä versiota. Tämä bugi ei koske TLS/SSL:ää kokonaan eikä myöskään kaikkia OpenSSL-toteutuksia vaan konkreettisia OpenSSL-versioita, joissa on HeartBleed-toteutus käytössä. Siihen materiaaliin kannattaa tutustua, jos aikoo käyttää OpenSSL:ää omassa projektissa. Voidaan todeta, että SSL-protokolla on kuitenkin huomattavasti turvallisempi verrattuna tavalliseen TCP-protokollaan, mutta ei voida tietää kaikkia tietoturvariskejä tiedonsiirrossa. [16.]

3.3 TCP- ja UDP-protokollien vertailu

Molemmat käyttävät sokettiyhteyttä ja parametrina ottavat vastaan IP-osoitteen ja porttinumeron. UDP ei kuitenkaan muodosta yhteyttä palvelimeen, minkä TCP tekee. Samanlainen piirre on myös se, että molemmat voivat lähettää ja vastaanottaa mitä vaan dataa tavumerkkijonona, sekä molemmat tietävät IP-osoitteen ja porttinumeron, mistä viesti on tullut.

Tiedonsiirtomenetelmät poikkeavat niin, että UDP lähettää datagram-paketteja, joiden saapumista määränpäähän ei tarkisteta (datagram-paketti sisältää määränpääosoitteen ja portin header-datassa ja lähetettävän tavumerkkijonon header-datan jälkeen). Tämä tekee UDP-protokollasta nopeamman kuin TCP-tiedonsiirron kannalta, muttei yhtä luotettava kuin TCP:n, koska UDP:n pakettidatan saapumista perille ei tarkisteta. Lisäksi ei tiedetä, onko kyseinen yhteys enää olemassa UDP:n suhteen, kun TCP voi havaita sen aika helposti ilman erillistä sovelluslogiikkaa. [2; 3; 19.]

Seuraavana on kuva UDP- ja TCP-protokollien pakettidatan headeristä, josta nähdään, että TCP-protokollan header-data on huomattavasti UDP:tä suurempi. Kuva on siis Berkeley-yliopiston materiaalista tai wikipediasta.



Kuva 5. Kuvassa on esitetty TCP-header ja UDP-header. [5.]

Berkeley-yliopiston materiaalin mukaan voidaan nähdä TCP:n ja UDP:n etenemisprosessin, kun tapahtuu kommunikointi kahden laitteen välillä. Nämä tiedot on esitetty huipputasolla eli ei tarvitse osata ohjelmointia varten kaikkea sitä. Tarkoitus on havaita, miksi UDP on nopeampi tiedonsiirtoprotokolla kuin TCP ja myös virhealttiimpi. Lyhyesti sanottuna Berkeley-yliopiston analyysista nähdään, että TCP-protokollassa tapahtuu enemmän aikaviivettä tiedonsiirrossa kuin UDP:ssä, koska TCP-protokolla tekee huomattavasti enemmän toimenpiteitä saman datamäärän lähetyksessä. Näihin asioihin liittyy tarkistus, että onko yhteys kahden päätepisteiden välillä olemassa ja tuliko lähetetty data perille. Katsotaan työssä osio ”OSI-malli ja TCP/IP-viitemalli”, mikä kertoo ruuhkahallinnasta kuljetuskerroksessa. Se on myös hyvä perustelu, miksi UDP on nopeampi. [5.]

UDP:tä on kuitenkin hyvä käyttää suurien datamäärien tiedonsiirtoon, missä laatu tai pakettidatan osittain poisjääminen ei vaikuta kovin paljon lopputulokseen. Nämä ovat yleensä video- ja äänitiedonsiirrot, joissa joidenkin pakettien poisjääminen ei vaikuta kovin merkitsevästä tiedon laatuun. Lisäksi tiedostojakeluohjelmat esimerkiksi ”e-mule” käyttävät UDP-protokollaa, koska saadaan nopeammin siirrettyä tiedot. Erillinen sovelluslogiikka ohjelmassa kertoo, mitkä datapaketit puuttuvat, että tiedosto olisi kokonainen verrattuna alkuperäiseen. [14.]

UDP ei käsittele pakettidataa striimeinä eli ei tiedetä, milloin käyttäjä on valmis tiedonsiirtoon. Tarvitaan erikseen jonkinlainen sovelluslogiikka, joka tarkistaa lähettämällä palvelimesta tarkistusviestin siitä, onko käyttäjä paikalla, jolloin käyttäjä antaa kuittausviestin. Näin todetaan, että UDP:ssä ei ole tietoa yhteydenmuodostuksesta eli voidaan lähettää data, jonka perillepääsyä ei tarkisteta, ellei erikseen ole ohjelmoitu sovelluslogiikkaa. TCP:n tapauksesta asia on selvä, kun havaitaan suoraan, että yhteys on muodostettu ja milloin se on katkaistu. Yksinkertaisesti tarkistetaan striiminä tulevan datan sisällön try-catch-lohkossa ”IOException”-poikkeus, joka kertoo, että striimin arvo on ”null”, kun yhteys on katkaistu.

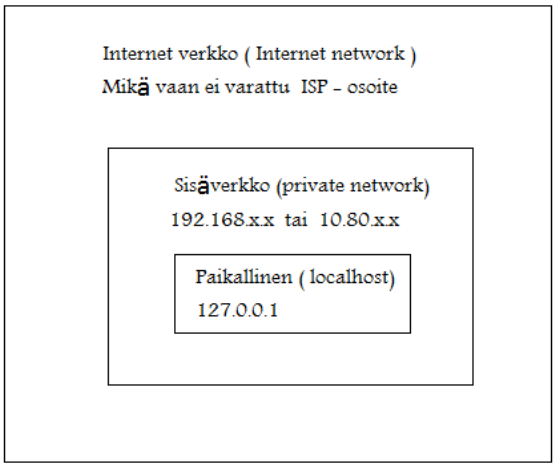
Voidaan siis päätellä, että UDP ei aina ole hyvä ratkaisu, vaikka on nopeampi kuin TCP-protokolla, mutta se pystyy käytännössä lähettämään samanlaisen datan paljon nopeammin. UDP:n hyvänä puolena on se, että odotustilassa vastaanottoa varten oleva kanava pystyy vastaanottamaan useita eri datagram-paketteja monesta eri lähteestä. [2; 3; 5; 19.]

3.4 ISP-palvelun käyttö

ISP eli Internet Service Provider on internetyhteyttä tarjoava palvelu, joka tulee operaattorien toimesta. Suomessa internetyhteyttä tarjoavia operaattoreita ovat mm. Elisa, Sonera ja Dna. Muissa maissa on muita tämän kaltaisia internetyhteyttä tarjoavia yrityksiä, joiden kommunikointi keskenään on standardisoitu. Internetyhteys toimii usean maiden välillä IP-osoitteen avulla ja siitä lähtevästä TCP- tai UDP-protokollan kautta. [21.]

ISP-tarjoaa ip-osoitteen, jonka kautta voi yhdistäytyä koneisiin internetverkon avulla, esimerkiksi Suomesta Kanadaan, ja sen tiedonsiirtonopeus riippuu signaalin kulkureitistä. Nykytekniikassa se on nopea toiminta, mutta ongelmia myös joskus ilmenee.

Tarkoituksena on, että sisäverkkoon tarkoitettuihin sovelluksiin ei päästä internetverkon kautta ellei muuten ole määritelty. Yksi ISP-osoite on sama kaikille sisäverkon koneille, jos koneet on yhdistetty nettiin saman reitittimen ja modeemin avulla. Reititin määrittelee saapuvan pakettidatan algoritmin sisäverkossa. [21.]



Kuva 6. Kuva on tekemäni havaintokuva, jossa voidaan kuvitella verkkorakenteen näkyvyyttä yleisellä tasolla.

Kuvasta havaitaan, että paikallinen verkko on yhdessä koneessa toimiva verkko ilman mitään yhteyksiä eli muista verkoista ei voida ottaa yhteyttä tähän verkkoon käyttämällä sisäverkon ip-osoitetta. Sisäverkko on reitittimen tekemä verkko ja tähän verkkoon kuuluvat vain tähän reitittimeen yhdistyneet koneet, eikä internetverkon kautta voi muodostaa yhteyttä sisäverkon ip-osoitteen avulla. Oletetaan, että reititin aina määrittelee automaattisesti sisäverkon ip-osoitteen. Kun joku laite on yhdistetty tähän sisäverkkoon, niin tiedetään, mistä laitteesta on kyse, ja voidaan viitata tähän laitteeseen sille tarkoitetulla sisäverkon ip-osoitteella. Kun puhutaan ISP-verkosta, niin tarkoitetaan internetverkkoa. Tämän verkon ip-osoitetta voivat käyttää kaikki, jotka ovat internetverkkoon yhdistäytyneet, mutta tietoturvan vuoksi palomuurit, virustorjunta-ohjelmat, ISP:tä tarjoava operaattori voivat estää uuden yhteyden muodostamista. Tähän kannatta tutustua erikseen ja myös ottaa huomioon riskitekijät liittyen oman yhteyden avaamiseen internetverkossa.

Yhteydet kannatta suojata palomuurilla, SSL/TLS-protokolalla tai tehdä oma vastaava ohjelmisto, joka suorittaa suojaustoimenpiteen. Uuden soketti-yhteyden avaaminen vaatii yleensä poikkeuksen, miksi se tehdään. Tämä on tarkoitettu haittaohjelmien estämiseksi, jolloin pyydetään varmistusta, että tämä yhteys todella halutaan tehdä.

3.5 Palvelimeen kohdistuvat hyökkäykset ja suojausmenetelmät

”Brute Force” on palvelimeen käyttäjiin kohdistuva hyökkäys, jonka avulla pyritään selvittämään käyttäjän salasanalla käymällä kaikki salasanavaihtoehdot läpi a-z, A-Z, 0–9. Huomioidaan siis kaikki sallitut merkit, jotka voivat esiintyä salasanassa. Tämä on hidas toiminto, ja tarvitaan paljon niin sanottuja ”XOR-vertailuja”, että saadaan oikea salasanakombinaatio. ”Brute Force” -hyökkäykseen estämiseksi voidaan kehittää tarkistusalgoritmi palvelimella, joka estää useita sisäänkirjautumisia palvelimen käyttäjään lyhyessä ajassa, jos palvelimelle on yritetty sisäänkirjautua esimerkiksi viisi kertaa alle minuutissa. Tällöin sen voi tulkita ”Brute Force” -hyökkäyksellä tai jolloin väkisin pyrkivällä ohjelmalla, joka yrittää murtautua käyttäjätilille. Sisäänkirjautumisominaisuus on hyvä lukita tietylle ip-osoitteelle ainakin noin 15 minuutiksi, jos on ollut useita sisäänkirjautumisia tuloksetta. [23.]

”DDOS”-hyökkäyksen eli palvelunestohyökkäyksen tarkoituksena on estää uusien yhteyksien muodostus palvelimeen tai kaataa palvelin, että palvelinsovellus ei enää pystyisi toimimaan kunnolla. Tehdään yhteyspyyntö useasta eri lähteestä samanaikaisesti. Yhdistyvät koneet voivat olla itse uhrina haittaohjelmilla, joiden tehtävä on tehdä ”DDOS”-hyökkäys tietylle palvelimelle. Siltä on vaikea suojautua varsinkin, kun kyseessä on yksi palvelin, joka ylläpitää järjestelmää. Siihen voi keksiä jonkun suodatusratkaisun, mikä vapauttaa tai ei vastaanota yhteyksiä, joilla ei ole selkeitä kommunikaatiotavoitteita. Yleensä tällaiset hyökkäykset ovat kohdistettuja konkreettisiin palvelimiin, eivätkä kaikki palvelimet ole vaarassa. [24.]

HTTP- tai FTP-pavelimissa tiedostojen luku- ja kirjoitusoikeudet kannattaa tarkistaa, etteivät ulkopuoliset käyttäjät pääsisi muuttamaan niitä. Jos on käytössä OpenSSL-toteutus esimerkiksi HTTPS-protokollan toteutus sen kanssa, niin kannattaa käyttää OpenSSL-versiota, joka ei käytä ”HeartBleed”-ylivuotobugia. [16.]

Pelipalvelimiin voi tulla väärennetty pakettidataa käyttäjiltä, jotka voivat käyttää huijausohjelmistoja rikkoakseen pelisääntöjä. Palvelimen sovelluslogiikka on tunnistettava, ellei tuleva data vastaa totuutta. Palvelin tietää pelin sovelluslogiikan ja osaa oman algoritmin perusteella todeta, että tieto ei ole virheellinen. Monista pelipalvelimista puuttuu sellainen suojaus. Sen takia monissa pelipalvelimissa käyttäjällä on mahdollisuus

käyttää esimerkiksi ”speed hack” -huijausta, jossa pelaaja liikkuu nopeammin pelikentällä kuin pelin sovelluslogiikka sallii. Tällöin on mahdollisuus saada pelipisteitä tai pelitavaroita huijaustavoilla, jotka eivät vastaa pelisääntöjä.

SSL:ään ”Heart Bleed”-havoittuvuuden on löydetty versiossa 1.0.1, eli se on ollut kaksi vuotta tietoturvariskinä. Codenomicon-tietoturvayritys löysi sen huhtikuussa vuonna 2014 ja ilmoitti suoraan OpenSSL-yhteisölle, joka päätti julkaista seuraavan OpenSSL-version, jossa HeartBleed-toiminnallisuus on poistettu väliaikaisesti tai pysyvästi käytöstä. Tämä on ylivuotobugi, joka mahdollistaa salaisen tiedon vuotamisen vääriin käsiin. [16.]

Löytyy varmasti useita muitakin hyökkäysmenetelmiä, jotka vaarantavat palvelimen toimintaa. Voidaanko luottaa TCP-käyttäjältä striiminä tulevaan pakettidataan? Ei voida aina luottaa, joten on keksittävä sovelluslogiikka, jolla voi todeta, että tuleva ja lähtevä tieto ei ole virheellinen. Myös salainen tieto ei vuoda sellaisille, kenellä ei ole oikeutta siihen.

3.6 Käyttöjärjestelmät ja laitteet

Missä käyttöjärjestelmissä ja laitteissa käytetään TCP–protokollaa?

TCP/IP-protokollaa käytetään kaikissa internetverkon laitteissa. Yleisimmät laitteet ovat tietokoneet, joissa niitä käytetään, Windows- ja Linux-käyttöjärjestelmä tai Mac-koneiden ”Mac OS” -käyttöjärjestelmä. Myös mobiililaitteissa kuten älypuhelimissa ja Tablet:PC:ssä on käytössä muiden laitteiden kanssa yhteensopiva TCP/IP-protokolla. Lisäksi TCP/IP-protokollan tukevat useat nykyaikaiset pelikonsolit kuten PS3, jotka käyttävät tämän protokollan monipeliominaisuutta varten.

Perusideana on, että kakki nämä laitteet voidaan yhdistää keskenään aliverkon tai internetverkon kautta ja välittää tavumerkkijono TCP-protokollan avulla. Ohjelmointitapa ja järjestelmän rakenne poikkeavat jokainen omalla tavallaan eli kannattaa tutustua konkreettiseen järjestelmään testisovelluksilla. Kuten mainitsin jo aikaisemmin, tämä kommunikointiominaisuus on yleisesti käytössä ja standardisoitu. Periaatteessa mikä vaan hieman vanhentunut laite sopii.

Verkkorakenteiden esimerkit TCP/IP-kokeilua varten

Matkapuhelimen sisäänrakennettu 3G-modeemi ja langaton ”wifi/wlan”-verkkoreititys on erittäin sopiva testeille, kun voidaan rakentaa kannettava sisäverkko ja laittaa se nopeasti pystyyn demosovelluksia varten.

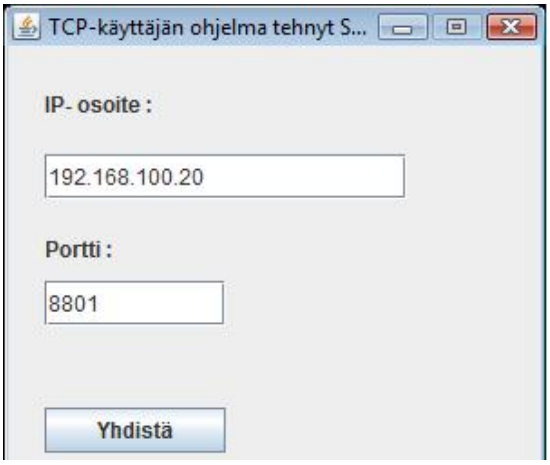
Langallinen ADSL-verkko ja sen perusreititys LAN-johdoilla sopii hyvin koti- tai yrityskäyttöön. Tämä on yleinen ympäristö, jossa olen tehnyt ja testannut demosovelluksia. Se on varma ja luotettava tapa sisäverkon muodostamiseksi.

Virtuaalinen sisäverkko on tietokoneiden sovellusohjelma, joka antaa virtuaalisen ip-osoitteen. Se toimii internetyhteyden kautta tai muiden yhteyksien kautta muodostuvana sisäverkkona, jonka yhdistetyt laitteet ovat fyysisesti kaukana toisistaan mutta ohjelmallisesti sisäverkossa. Voidaan emuloida ohjelmallisesti sisäverkon internetverkon kautta esimerkiksi ”Hamachi”-ohjelma, joka on tarkoitettu virtuaalisisäverkon emulointiin. Tämä ohjelma antaa poikkeuksen TCP-yhteyden muodostamiseen ISP:n kautta. Se siis helpottaa tiedonsiirtoa, kun oletuksena ISP:n kautta tiedonsiirto on lukittu. Tämä ohjelma käytetään usein pc:n monipelien yhteyksien muodostamiseen, jolloin pääsee pelaamaan peliä ihan kuin olisi sisäverkossa mutta internetkäyttäjien kanssa. Lisäksi tämän ohjelman voi käyttää myös muihin sovelluksiin ja omien demo-projektien testailuun. Kyseessä siis on vielä yksi ip-osoite, joka viittaa virtuaalisen sisäverkkoon. Käyttäjä siis antaa palvelimen virtuaaliverkon ip-osoitteen, jos haluaa muodostaa yhteyden palvelimeen. Molemmissa päissä kyseisen ohjelman pitää olla päällä, jotta tämä verkkoyhteys olisi olemassa. Muuten toiminnallisuus on sama kuin fyysisessä sisäverkossa. [13; 21.]

4 Oman projektin kuvaus

Omana vapaamuotoisena projektina tein Javalla chattailusovelluksen syksyllä vuonna 2013, joka oli muokattu esityskelpoiseksi insinöörityödemosovellukseksi insinöörityön kirjoituksen aikana, koska se sopii hyvin tähän aiheeksi. Sovellus muodostuu kahdesta ohjelmasta: palvelin- ja käyttäjäohjelmasta, jotka kommunikoivat keskenään sisäverkossa TCP/IP-protokollan kautta.

TCP-palvelinohjelmisto osaa vastaanottaa usean eri TCP-käyttäjäyhteyden sisäverkossa ja pystyy kommunikoimaan näiden käyttäjien kanssa kaksisuuntaisella tiedonsiirtoyhteydellä eli voi lähettää ja vastaanottaa tiedon usealta eri käyttäjältä samanaikaisesti. Lisäksi palvelin havaitsee, kun joku käyttäjän yhteys on katkennut ja osaa vapauttaa yhteyssäikeen ja luoda sen tilalle uuden odotussäikeen, joka on valmis vastaanottamaan uutta käyttäjää (monisäikeiden, yhteysäikeiden ja odotussäikeen toiminnasta kerron myöhemmin). Yhteystoiminnot ovat ohjelmallisesti automatisoituja, eli palvelinohjelmisto vastaanottaa ja purkaa yhteyden käyttäjän kanssa halutulla tavalla. Lisäksi palvelin generoi jokaiselle yhdistetylle käyttäjälle nimen, jonka avulla hänet tunnistetaan, kun hän lähettää viestin chatissa.



Kuva 7. Kuva esittää TCP-käyttäjän sovellusta.

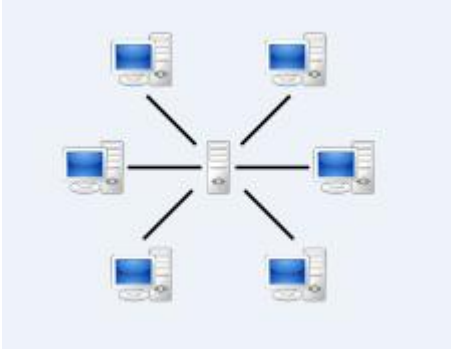
TCP-käyttäjänsovelluksessa on kaksi syöttökenttää, jonne voi antaa palvelimen voimassa olevan ip-osoitteen ja porttinumeron. Nämä parametrit käytetään TCP-palvelimen yhteydenmuodostukseen ja ilmoitetaan käyttäjälle, jos yhteys ei ole mahdollinen. Käyttäjän ja palvelimen välillä muodostettu yhteys tarkoittaa sitä, että tiedonsiirto on mahdollinen ja ollaan chatissa mukana. Voidaan lähettää viesti muille käyttäjille kirjoittamalla lähetettävän tekstikenttään viesti ja painaa lähetysnappia. Viesti saapuu palvelimelle ja ottaa sieltä käyttäjälle tunnistusnimen, jonka jo mainitsin.

Palvelin vastaanottaa käyttäjän viestin ja lähettää sen muille käyttäjille nähtäväksi palvelimen käyttäjälle tarkoitetun tunnistusnimen kanssa. Sovelluksen käyttäjät, jotka on yhdistetty tähän palvelimeen, näkevät kaikkien käyttäjien viestit ja myös oman. Käyttä-

jät voivat päätellä käyttäjänimen perusteella, kuka on lähettänyt viestin. Palvelin tietää aina, kenestä käyttäjästä on kyse, kun chatissa tulee viesti ja myös ohjelmallisesti voi aika paljon vaikuttaa tähän yhteyden toimintaan, joten palvelimen toiminnassa on tapahtunut visuaalista parannusta.

Molempien ohjelmistojen toimintatavoite on toimia fyysisesti useiden tietokoneiden välillä sisäverkossa. Tähän tarvitaan sellainen sovelluslogiikka, josta kerron työn aikana, koska olen itse kehittänyt sen. Testilaitteina toimivat pöytäkoneet ja kannettavat, jotka on varustettu "Java Runtime Enviroment" eli JRE:llä versiolla 1.6-1.7-ohjelmistolla ja Windows-käyttöjärjestelmällä, mieluummin "XP":llä tai uudempi käyttöjärjestelmällä. Linux-pohjaisten sovellusten käyttö on mahdollinen. Tämän oli todettu, kun oli testattu sovelluksen käyttöä Linux Xubuntu 12.04 LTS” -käyttöjärjestelmässä, jossa on "ICE TEA" -niminen Java-virtuaalikone, joka on yhteensopiva JRE:n kanssa. Tästä nähdään, että "raudan" järjestelmävaatimukset eivät ole suuret verrattuna nykypäivään. Peruskone riittää hyvin tähän tarkoitukseen. Myöhemmin voin kehittää Android-käyttöjärjestelmälle sovelluksen, joka on yhteensopiva tämän chattisovelluksen kanssa, mutta ei nyt.

Kuvassa näkyy omassa projektissa käytetty tähtitopologia, jossa yhdellä palvelimella on monta käyttäjä. Palvelin on esitetty keskellä ja sen ympärillä olevat tietokoneet ovat käyttäjät, joiden yhteydet on havainnollistettu viivoilla. Viiva tarkoittaa yksittäistä kaksisuuntaista yhteyttä tietyn käyttäjän ja palvelimen välillä. [20.]



Kuva 8. Kuva esittää tähtitopologian eli yhden palvelimen, jolla on useita käyttäjiä. [1; 20.]

Verkossa olevat tietokoneet tunnistetaan ip-osoitteen avulla. Projektissa käytetään yleisesti käytettyä IPV4-standardiosoitetta yhteydenmuodostukseen ja lukkiutuvaa säiemekanismia, mikä käytännössä tarkoittaa sitä, että yksi käyttäjä varaa yhteyden

palvelimessa ja muut käyttäjät eivät voi yhdistäytyä siihen. Toisin sanoen palvelimessa on odotussäie, joka odottaa, että joku käyttäjistä yhdistyy siihen. Tällöin palvelimen odotussäie siirtyy seuraavan tilaan ja on valmis lähettämään tai vastaanottamaan tiedon. Sen takia tämä yhteys ei voi enää vastaanottaa uutta käyttäjää ja ainoa, mitä sille voi tehdä, on sulkea se jos yhteys on katkaistu. Voidaan päätellä, että odotussäie on tavallinen säie, joka on tarkoitettu yhteyttä varten käyttäjän kanssa. Nimeän sen myös ”yhteyssäikeeksi”. ”Odotussäie”-nimeä käytän silloin, kun tähän säikeeseen ei ole muodostettu yhteyttä ja kun yhteys on muodostettu, niin puhutaan yhteyssäikeestä eli kun käytetään samaa säiettä eri tilassa, käytetään kahta erilaista nimeä. [1; 20.]

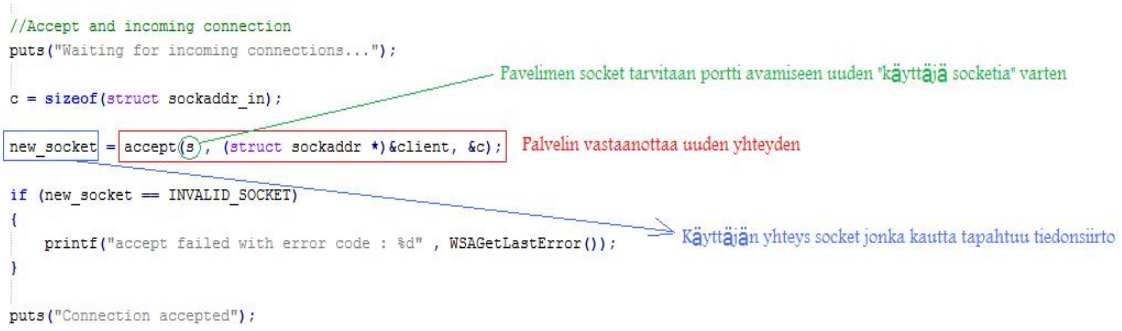
5 Sokettiohjelmointi

5.1 TCP-soketti yleisesti

Käyttäjäsovellus yhdistyy antamalla parametrilla oikea tieto palvelimen ip-osoitteesta ja porttinumerosta. **Soketti** on TCP-, UDP-, SSL-, Bluetooth(RFCOMM)-protokollaa varten oleva oliomuuttuja, joka saa parametrina käyttäjältä palvelimen yhteystiedot. Lisäksi TCP-soketin on tiedettävä, onko kyseessä IPV4- vai IPV6 -ip-osoite ja kummasta yhteysprotokollasta TCP:stä tai UDP:stä on kyse, jos puhutaan ”winsock 2.0”-ohjelmoinnista. Esimerkiksi C/C++:ssa tarvitaan nämä tiedot erikseen esitettyinä ja Javalla ne tiedetään automaattisesti. Muuta erikoisempaa tietoa ei tarvita. Voidaan suorittaa tämä yhteydenmuodostus Javalla try-catch-lohkossa ja C/C++:lla voidaan kirjoittaa if-else-ehtolauseen, jolloin tiedetään, onnistuiko yhteys vai ei. Sekä palvelin että käyttäjä tietävät, kun yhteys on muodostettu ja sovelluslogiikalla voi ohjelmoida ilmoitusviestin sen onnistuttua tai epäonnistuttua. [1; 7; 8.]

Palvelimen pitää olla valmis vastaanottamaan uusi TCP-yhteys. Muuten yhteys ei toimi, vaikka käyttäjä antoi oikean tiedon palvelimesta . Palvelin havaitsee käyttäjän yhteyden muodostamisen ja suorittaa tarvittavat toimenpiteet. Onnistuneesti muodostetun yhteyden kautta tiedonsiirto on mahdollinen.

C/C++ TCP-palvelimen uuden yhteyden vastaanotto TCP-käyttäjältä (Berkeley Socket malli) :



Kuva 9. Kuvassa näkyy C++:n TCP-palvelimen käyttäjän vastaanotto. Kyseessä on siis lukkiutuva yhteys. [6; 7.]

Kuvassa näkyy koodiesimerkki C++:lla TCP-palvelimen käyttäjän vastaanotto, kun palvelin on valmis vastaanottamaan uuden käyttäjän. Voidaan nähdä myös Java- koodiesimerkki ja verrata vastaanottofunktioita keskenään. [6.]

Tässä on Javalla tehty koodiesimerkki :

```
58          try{
59
60              taman_kayttajan_yhteys_socket = sSocket.accept();
61
62              //setClientIsConnected(true);  // oma viritelmä
63
64              System.out.println("Kayttaja : " +kayttajan_nimi+" yhdistetty !");
65
66          }catch(IOException e){ System.out.println("Käyttäjä yhteys ongelma !"); }
67
```

Kuva 10. Kuvassa näkyy Java TCP -palvelimen käyttäjän vastaanotto. Kyseessä on siis lukkiutuva yhteys. [1; 4.]

TCP-palvelimen ja käyttäjän vastaanottofunktiossa Javalla on paljon samoja piirteitä kuin C/C++:lla, mutta se ei tarvitse **sockaddr_in**-tietuetta Javassa, koska se antaa C++-tapauksessa tiedon ip-osoitteen tyyppistä. Esimerkiksi AF_INET on IPV4-

tarkoitettu ip-osoite ja AF_INET6 on IPV6-tarkoitettu ip-osoiteparametri, joka näkyy **sockaddr_in**-tietueessa, TCP- tai UDP-protokollan ominaisuudesta, tietovirran tyypistä. Javan tapauksessa näitä tietoja ei tarvitse erikseen ohjelmoida, eli Java tietää IP-osoitteen tyypin itse, tietovirran asiat ja muut. Ohjelmoitavan striimin lähetys tai striimin vastaanotto Javalla tapahtuu vasta tämän jälkeen. Erittäin tärkeä asia molempien suhteen sekä Java- että C/C++- ja muissa TCP-toteutuksissa on, että palvelimen ja käyttäjän soketti palvelinohjelmassa on valmis uuden yhteyden vastanottoon. Ei voida avata yhteyttä palvelin-käyttäjän välille, jos sokettioliota ei ole määritelty. Sokettimäärittely on siis tehtävä ennen yhteyden muodostusta. Winsock 2.0 täytyy alustaa ensin, ennen kuin voi avata sokettin ”WSAStartup”-funktiolla. [6; 7.]

5.2 Palvelimen porttiavaus

Palvelimen ohjelmointivaihe tapahtuu niin, että tehdään uusi ”ServerSocket”-olio Javalla, joka ei saa parametrina varattua porttinumeroa. Varataan uusi porttinumero tässä sovelluksessa, eli ohjelma tekee palvelimen, jolla on käytössä oma porttinumero tätä yhteyttä varten. Palvelimen ei tarvitse tietää omaa ip-osoitetta ollenkaan.

Kuvassa 11 on Javalla tehty yksinkertainen palvelimen portin avaus. Palvelin yrittää avata porttia try-catch-lohkon sisällä. Jos portin avaus epäonnistuu, niin suoritetaan catch-lohko IOExceptionilla siis ”ServerSocket”-oliota ei ole määritelty, eikä voida jatkaa eteenpäin, vaan ohjelman pitää sulkea tai suorittaa ”ServerSocket”-määrittelytoimenpide uudelleen, kunnes se on määritelty. Palvelimen porttiavaus epäonnistuu, jos yrittää avata porttia, joka on jo varattu toiselle yhteydelle. Voidaan Windows-käyttöjärjestelmässä tarkistaa käytössä olevat portit komennolla ”netstat” tai oman sisäverkon ip-osoitteet komennolla ”ipconfig” komentorivissä. [4.]

```
int  portti = 8801;
ServerSocket  palvelin_sock;
try{
    ServerSocket  palvelin_sock = new ServerSocket(portti);
    //Ok!  palvelin  on avanut porttin. Tämä porttinumero on varattu tähän ohjelmaan.
}catch(IOException e){
    // Tapahtui virhe , ei voi avata porttia
}
```

Kuva 11. Kuva esittää TCP-palvelimen portin avaamista palvelinpäässä Java-koodilla.

Kun portti on määritelty, niin tarvitaan vielä ”Socket”-olio, johon käyttäjä voi muodostaa yhteyden. Tämä soketti vastaanottaa yhteyden käyttäjältä ”accept”-funktion avulla ja on valmis tiedonsiirtoon tämän käyttäjän kanssa. Tästä olen siis jo kertonut tämän luvun alussa. [1; 4.]

5.3 Käyttäjän yhteydenmuodostus palvelimelle

Käyttäjän yhteydenmuodostus palvelimelle tapahtuu luomalla yhteys sokettiin. Yhteys-parametreina ovat palvelimen ip-osoite ja palvelimen porttinumero. Suoritan tämän koodin try-catch-lohkossa, koska yhteydenmuodostus voi epäonnistua, jolloin suoritetaan tarvittava toiminto.

Yhteyden epäonnistuttua sokettiyhteys ei ole määritelty, eikä tiedonsiirto ole mahdollinen. Yhteydenmuodostamista voi siis yrittää uudelleen tai lopettaa ohjelman toiminta Tämä yleensä tapahtuu, jos käyttäjä ei ole antanut oikeaa yhteystietoa palvelimesta, tai palvelin ei ole valmis vastaanottamaan uutta yhteyttä.

Onnistunut yhteydenmuodostus saa aikaan sen, että sokettiyhteys on valmis tiedonsiirtoon. Palvelimella ja käyttäjällä on silloin tieto tästä yhteydestä. Yhteyden katkettua voidaan havaita tilanne molemmissa päissä ja tehdä tarvittava sovelluslogiikka, joka korjaa katkenneen yhteyden tai sulkee yhteyden ja luo sen jälkeen uuden. Tämä on yksi tärkeimmistä syistä, miksi käytän omassa projektissa ”lukkiutuvaa yhteyttä”. Tästä yhteystyypistä kerron toisessa luvussa.

Kuvassa nähdään TCP-käyttäjän koodi Javalla. Soketille annetaan kaksi yhteysparametria, joista oli jo kerrottu. Tärkeänä havaintona on yksinkertainen sovelluslogiikka TCP-protokollan yhteyden muodostuksessa, siis perus Java-ohjelmointia. [1; 4.]

```
String  palvelimen_ip_osoite = ”192.168.100.20”;
int palvelimen_portti = 8801;
Socket  kayttaja_yhteys_sock;
try{
    kayttaja_yhteys_sock  = new Socket(palvelimen_ip_osoite,  palvelimen_portti);
    //Yhteys palvelimelle onnistui
}catch(IOException e){
//Ei yhteyytä palvelimelle
}
```

Kuva 12. Kuva esittää TCP-käyttäjän yhteydenmuodostusta palvelimelle Java-lähdekoodin.

Nähdään myös C/C++-kooditoteutuksen, jossa TCP-käyttäjä muodostaa yhteyden palvelimeen. Esitän ”winsock 2.0”-kooditoteutuksen, jossa ei välittämättä näe kaikkia tärkeitä edellisiä muuttuja-alustuksia, mutta nähdään varsinainen yhteydenmuodostuskoodi ja pystytään tekemään vertailua Java-toteutuksen kanssa.

```
struct sockaddr_in   kayttaja;
.....
kayttaja.sin_addr.s_addr = inet_addr("192.168.100.20");
//ip-osoitteen tyyppi    AF_INET  = IPV4    ja   AF_INET6 = IPV6
kayttaja.sin_family = AF_INET;
//portti
kayttaja.sin_port = htons( 8801 );
.....
if (connect(   s ,  (struct sockaddr *)&server , sizeof(server) ) < 0) {
//ei yhteyttä palvelimeen
} else{
//yhteys muodostettu
}
```

Kuva 13. Kuva esittää TCP-käyttäjän yhteydenmuodostusta palvelimelle C/C++-ohjelmoinnilla.

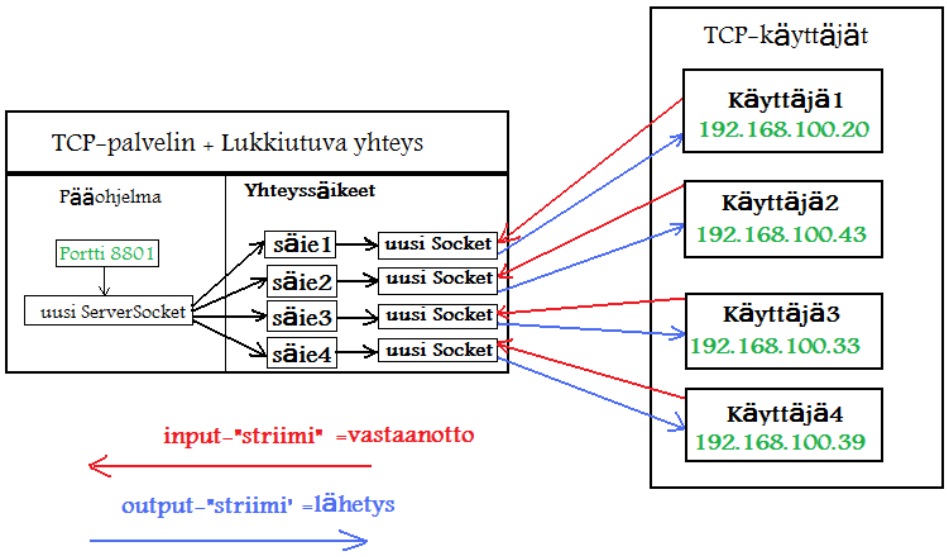
Kuvasta nähdään, että koodi näyttää hieman erilaiselta verrattuna Java-toteutukseen, mutta toimintaperiaate on sama ja myös yhteensopiva yhteys Javan kanssa. Voidaan

muodostaa yhteys C/C++-sovelluslogiikalla ohjelmoidun TCP-käyttäjän Java TCP-palvelimen kanssa niin, että molemmat päät havaitsevat toisensa. Tämä on yksi perusteluista TCP/IP-protokollan standardi tekniikalle eri järjestelmien ja sovellusten välillä. Tiedonsiirto tässä tapauksessa on myös mahdollinen. [4; 5; 7; 22.]

5.4 Lukkiutuva yhteys

Lukkiutuva yhteys tarkoittaa, että palvelimen yhteys odottaa käyttäjän yhdistäytymistä siihen. Tämä koodipätkä Javalla ”userSocket = sSocket.accept();” saa aikaan palvelimen sokettiprosessin odotustilaan, kunnes käyttäjä on yhdistetty, jolloin kommunikointi kyseisen käyttäjän kanssa on mahdollinen.

Ratkaisu siihen, että palvelin ymmärtäisi usean käyttäjän samanaikaisesti, tapahtuu niin, että luodaan sokettiyhteys erikseen uudessa säie-oliossa. Näin ollen jokaista yksittäistä käyttäjää ja palvelimen välistä yhteyttä varten on olemassa oma säie. Tämä vapauttaa palvelimen pääprosessia, eikä se jää odotustilaan odottamaan yksittäistä käyttäjää, vaan palvelinohjelma voi vapaasti jatkaa omaa toimintaansa eikä itse pääprosessi ole odotustilassa, eli palvelimen pääohjelma pystyy käsittelemään useita säieyhteyksiä lukitsematta itseään.



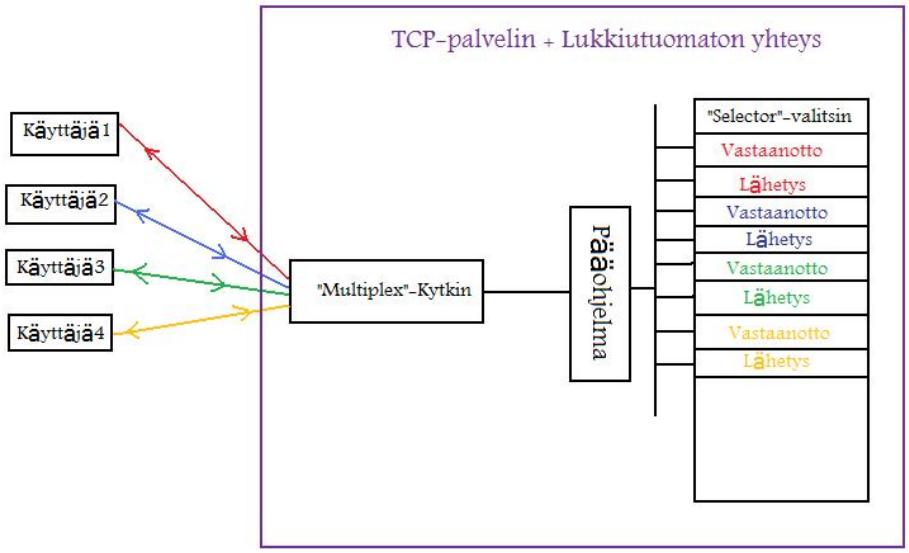
Kuva 14. Kuva esittää oman projektin TCP-palvelimen ”Lukkiutuva yhteyden” monisäieyhteystoteutusta

Kuva 14 näyttää projektini java-chattailu sovelluksen rakenteen. Jokaista käyttäjää varten teen uuden säikeen, jonka tarkoituksena on luoda uusi TCP-sokettyhteys palvelimen ja käyttäjän välillä. Tämä yhteys on siis kaksisuuntainen eli voidaan lähettää ja vastaanottaa dataa palvelimelta ja käyttäjältä.

Ohjelmallisesti voidaan todeta, että säie on uusi olio Javalla ja että voidaan asettaa muuttujan arvot kyseiselle säikeelle. Voidaan myös valvoa säikeen TCP-yhteyden muodostusta ja kertoa, mikä yhteystilanne on menossa. Lisäksi voidaan sanoa, että säie on verrattavissa C++-”lapsiprosessiin” eli tässä tapauksessa säikeen valvojana on palvelimen pääohjelma, joka käynnistää tämän säikeen. Säikeen muuttujat näkyvät pääohjelmassa, eli pääohjelma osaa käsitellä usean käyttäjän TCP-yhteydet yhtenä kokonaisuutena. Tämän takia projektini palvelinsovelluksessa on käytössä MVC-mallinen rakenne, missä yhteyssäikeet käsitellään erillisinä malli-olioina. Palvelinsovelluksen pääohjelma toimii ohjainluokkana ja käyttöliittymää varten tehdään erillinen olio, jossa tapahtumakuuntelija välittää tiedot pääohjelmaan. Tällaisen sovelluslogiikan olen itse keksinyt ja toteuttanut Java-ohjelmoinnilla. [1; 4.]

5.5 Lukkiutumaton yhteys

Lukkiutumaton yhteys vapauttaa palvelimen monisäikeisyydestä tai useasta lapsiprosessista. Palvelimen resurssien käyttö ja kuormitus on pienempi, kun ei ole useita odotussäikeitä odottamassa yhteyden muodostamista, vaan kaikki tapahtuu vain yhden prosessin kautta. Kun käyttää useita kanavia yhden prosessin kautta, niin kyseessä on ”**multiplexing”** eli kytkin usean eri toiminnon välillä. Esimerkiksi datan lähetys tai vastaanotto saman prosessin kautta, mutta ylimääräisellä aputiedolla, joiden tarkoituksena on määritellä paketin perillemenemistä oikeaan kohteeseen, voidaan kutsua ”**selector**”-nimellä eli valitsin.



Kuva 15. Kuvassa on esitetty ”Lukkiutumaton yhteys” -ratkaisu usealle käyttäjälle.

Kuvassa 15 on hahmotelma, miltä tämä rakenne näyttäisi teoreettisesti. Kaikki toiminnot tapahtuvat yhden prosessin kautta. Siis yhtä käyttäjää varten on tehtävä ainakin kaksi tapahtumaa lähtevän ja tulevan datan varten, mutta vain yksi tapahtuma tapahtuu kerrallaan. Nämä tapahtumat tehdään eri prosessina, siis valitsin valitsee datan lähetys- tai vastaanottotoimintoa konkreettiselle käyttäjälle. ”Multiplex”-kytkin laittaa tämän tapahtuman käyntiin, ja sen jälkeen siirtyy seuraavaan tapahtuman. Lyhyesti sanottuna tapahtumalla on kaksi erilaista ominaisuutta lähetys tai vastaanotto ja tieto, kenelle käyttäjälle tapahtuma tehdään. Nämä tiedot ovat tapahtumien ”header”-datassa, joita ”Multiplex”-kytkin tunnistaa.

Tämä voi aiheuttaa pakettipäällekkäisyyden, jos ei ylläpidä erillistä käskylistaa, mikä määrittele tulevien ja lähtevien pakettidatojen suoritusjärjestyksen. Tarvitaan siis logi-, historia-, kuittaus-, aikaleimatietoja jokaisesta yksittäisestä lähetysdatasta tai tulevasta pakettidatasta. Tämä kuitenkin tuhlaa ylimääräistä kaistaleveyttä, kun tarvitaan ylimääräistä tietoa käyttäjien yhteyksien ylläpitämisen. Lisäksi **selector** osaa valita, mikä toimenpide tehdään ja milloin tietyn käyttäjän vuoron tulee tehdä hänelle tarkoitetut toiminnot eli noudatetaan käskyt järjestyksessä päällekkäisyyden estämiseksi. Lyhyesti sanottuna selector valitsee sopivan toiminnon ja välittää sen **multiplex**-käsittelyä varten. [1.]

Javassa lukkiutumaton säie-toiminnallisuus tuli käyttöön versiosta 1.4 lähtien. Se mahdollistaa yhteyden muodostamisen lukitsematta sokettiyhteyttä. Käytetään kytkintä usean käyttäjän välillä, mitä kutsutaan termillä **multiplexing**. Hyvänä puolena on se, ettei tarvitse luoda montaa yhteyssäiettä, kun prosessi tai säie ei mene lukkoon. Myös resurssien käytössä se on edullisempi, jos palvelinparametrit ovat heikoimmat. Huonona puolena on ainakin poikkeuksien hallinta ja vaikeammin ohjelmoitava järjestelmä. Usean käyttäjän kanssa toimiminen tällaisessa järjestelmässä tarvitsee käyttäjien historiatietojen ylläpitämisen. Näin tiedetään, mistä käyttäjästä on kysymys. Tällainen sovelluslogiikka toimii hyvin ”lukkiutuva yhteyden” korvikkeeksi palvelimessa, kun ei enää tarvitse käynnistää useita säikeitä usean käyttäjän ylläpitämiseksi. Multiplexing-palvelimia voi ohjelmoida Javalla tai C/C++:lla. Molemmista on saatavilla ohjelmointidokumentaatiot, mutta en käytä Multiplexing-sovelluslogiikka käytännössä, vaan olen kertonut teoreettisesti, mikä olisi toinen vaihtoehto monisäikeisyys-toteutukselle, jotta useiden käyttäjien yhteys samanaikaisesti palvelimelle olisi mahdollinen. [1.]

5.6 Tiedonsiirto-ohjelmointi

Kun käyttäjän ja palvelimen välillä yhteys on muodostettu, niin voi suorittaa tiedonsiirron molempiin suuntiin eli palvelin voi vastaanottaa ja lähettää dataa käyttäjälle, ja käyttäjä voi vastaanottaa ja lähettää dataa palvelimelle. Palvelin voi oman sovelluslogiikan avulla käsitellä datan ja välittää sen muille käyttäjille, kun tiedetään, mistä yhteyssäikeestä data on peräisin eli tiedetään, kenen kanssa palvelin kommunikoi, vaikka olisikin yli 100 käyttäjää samanaikaisesti samassa palvelimessa.

Varsinainen tiedonsiirto tapahtuu sokettiolion kautta, eli kun tämä yhteys on onnistuneesti alustettu käyttäjä- ja palvelinpäässä, niin ollaan valmiina tiedonsiirtoon. Tämän takia käytän ”striimi”-sanaa eli tarkoitan sillä tietovirtaa, joka mahdollistaa tiedon lähetyksen ja vastaanoton molemmissa päissä sekä käyttäjäsovelluksessa että palvelimessa. Striiminä ei siirretä koko dataa yhtenä toimintona, vaan pieninä palasina. Suurien datamäärien lähetys voi täyttää puskurin, tai virhetilanteessa tämän datan uudelleenlähetys on huomattavasti raskaampi. Suosittelen lähettämään alle 64 kilotavun kokoisen datan yhdessä paketissa. Lopputuloksena voi olla esimerkiksi kahden gigatavun tietokokonaisuus, kun kaikki paketit ovat perillä. Varaudutaan aina siihen, että yhteys voi

mennä poikki tiedonsiirron aikana. Tämä on striimien poikkeustilanne sekä Javalla että C/C++:lla, jolloin tehdään yhteyden katkaisu ja sen jälkeen sen uudelleen muodostus ohjelmallisesti. Lisäksi kannattaa tarkistaa, mitä dataa on saapunut jo perille ja voidaanko jatkaa tiedonsiirtoa eteenpäin tai pitääkö lähettää kaikki data uudelleen tiedon eheyden vuoksi. [4; 22.]

Java-ohjelmoinnissa puhutaan ”InputStream”- ja ”OutputStream”-tiedonsiirrosta. ”InputStream” on tiedon vastaanottostriimi ja ”OutputStream” on lähetysstriimi. Molemmat striimit ovat käytössä sekä palvelimella että käyttäjällä. Nämä striimit tarvitsevat soketit, joilla on yhteys auki. Toteutus on tehtävä try-catch-lohkossa, jotta voidaan havaita virhe tiedonsiirrossa. Esimerkiksi yhteyden katkeaminen palvelimen tai käyttäjän syystä aiheuttaa tällaisen tilanteen. Kuten jo mainitsin, sekä Javalla että C/C++-toteutuksessa on sama tiedonsiirtoperiaate. C/C++-tapauksessa sovelluslogiikka on esitetty eri tavalla, jolloin tarvitaan enemmän yhteysparametreja, jotka korostavat, että käytössä on TCP-protokolla eikä UDP ja että ip-osoitteen tyyppi on esimerkiksi IPV4. Lisäksi ei luoda striimi-oliomuuttujia C/C++-toteutuksessa, vaan tiedonsiirto tapahtuu konkreettisten funktioiden avulla, mitkä ovat ”winsock”-kirjaston komponentteja, mutta kuitenkin viittaus soketti-oliomuuttujan tiedonsiirtofunktioista toimii saman periaatteen mukaisesti verrattuna Javaan. [4; 22.]

Javalla tehdään TCP-palvelimen viestin vastaanotto, kun soketti on alustettu tiedonsiirtoon.

```
73
74                  try{
75
76                     vastaanotto_striimi = new DataInputStream(taman_kayttajan_yhteys_socket.getInputStream());
77
78                     kayttajalta_tuleeva_viesti =  kayttajan_nimi+">"+vastaanotto_striimi.readUTF();
79
80                  }catch(IOException e){
81
```

Kuva 16. Kuvassa on esitetty palvelimen vastaanottostriimi Javalla.

Huomataan, että koodin vastaanottostriimi pitää vaihtaa ”readUTF”-funktion tilalle ”read”-funktio, jos aikomuksena on vastaanottaa C/C++-käyttäjäsovelluksesta tuleva data, koska ”readUTF”-funktio Javalla ei ole yhteensopiva C/C++:n kanssa, niin joudutaan käyttämään Javan ”read”-funktiota, mikä lukee tavumerkkijonon. Sen jälkeen

voidaan erikseen ohjelmoida sovelluslogiikka, mikä muuttaa nämä tavumerkkijonot konkreettiseksi tiedoksi. [4; 22.]

Taulukko 1. Tiedonsiirto-funktiot Javalla ja C/C++:lla taulukkomuodossa.

	Java (JRE)	C/C++ (Winsock)
Tiedon vastaanotto	InputStream -> read	“recv” -funktio
Tiedon lähetys	OutputStream -> write	“send” -funktio

Taulukossa näkyy TCP-protokollan tiedonsiirtofunktiot, jolloin voidaan havaita, miten Javan ja C/C++-toteutuksen voi keskenään toteuttaa. Kuten on jo ymmärretty, tämän insinöörityön luvusta 5.1 ja 5.3, missä kerrotaan Javan ja C/C++:n yhteydenmuodostuksesta keskenään, niin tässä on seuraava vaihe, kun yhteys on olemassa. [4; 22.]

Muistetaan vastaanottaa lähetettävä data TCP-protokollan kautta. Muuten pakettidata ei saavu perille. Tämä aiheuttaa pakettidatan jumiin menemiseen puskurissa, kun perillepääsykuittausta ei saada. Tällöin paketit ovat odotustilassa, mistä ne eivät pääse pois. TCP-protokolla aina varmistaa kuittauksella, että lähetettävä data on saapunut perille ja tarvittaessa lähettää tämän datan uudelleen. Pakettidatan pois jääminen laittaa TCP-yhteyden tukkoon, ja yhteys menee kokonaan jumiin. Tämä aiheuttaa sen, että kaikki käyttäjäsovellukset täytyy sammuttaa, mikäli ne on yhdistetty tähän palvelimeen. Näiden ohjelmistojen uudelleenkäynnistys korjaa tilanteen, mutta aiheuttaa tämän ongelman uudelleen, jos asiaa ei ole korjattu. Onneksi tässä ongelmassa menevät ainoastaan näiden sovellusten yhteydet jumiin eikä koko TCP-protokollan tiedonsiirto.

6 Yhteenveto ja pohdinta

Yhteenvetona voi kertoa, että TCP-sokettien ohjelmointi on erittäin hyödyllistä varsinkin verkko-ohjelmoinnissa, mutta sen opettelu ja loppuun asti ymmärtäminen on vaativa myös kokeneille ohjelmoitsijoille. Runsaan materiaalin ja dokumentaation olemassaolo edellyttää hyvää oppimista. Lisäksi TCP-protokollan tiedonsiirto on yhteensopiva keskenään useissa nykylaitteissa ja myös vanhentuneissa laitteissa, eli kustannukset eivät ole kovin suuret, kun voi hankkia hieman vanhentuneen koneen ja ohjelmoida ilmaisilla kehitystyökaluilla. Ne tarjoavat hyvän TCP-sokettiohjelmoinnin dokumentoinnin ja runsaasti tietoliikenteen materiaalia.

Nyt tiedetään, että TCP-sokettiohjelmoinnin aloitusvaihe ei ole niin raskas kuin voisi kuvitella ja että laajentamalla omia yksinkertaisia sovelluksia ymmärretään myös suurempia kokonaisuuksia kuten ”lukkiutuva yhteys ” ja ”lukkiutumaton yhteys”. Sen jälkeen osaa hahmottaa yhteystopologioita ja sitä, millainen yhteysmalli on TCP-käyttäjien välillä. Seuraavaksi voidaan miettiä konkreettisia sovelluksia kuten monipelit, chattailu-ohjelmat ja web-/http-toiminnallisuus TCP-protokollan kautta. Tästä eteenpäin verkko-ohjelmoinnin yleinen tekninen ymmärrys kasvaa ja uusien protokollien ohjelmoinnin omaksuminen on helpompaa. Esimerkiksi bluetooth rfcomm -protokollan toimintaperiaatessa on paljon samoja piirteitä verrattuna TCP-protokollaan, ainakin ohjelmoinnin kannalta.

Lähteet

1. Killer Java-Network basic chapter 23 (Java TCP-ohjelmointi)
   http://fivedots.coe.psu.ac.th/~ad/jg/ch18/ch18.pdf
   Luettu 2.5.2014.

2. Wikipedia TCP-protokolla englanniksi:
   http://en.wikipedia.org/wiki/Transmission_Control_Protocol
   Luettu 4.4.2014.

3. Wikipedia TCP-protokolla suomeksi:
   http://fi.wikipedia.org/wiki/TCP
   Luettu 4.4.2014.

4. TCP-soketti Javalla (ensimmäinen demosovellus)
   http://systembash.com/content/a-simple-java-tcp-server-and-tcp-client/
   Luettu 27.3.2014.

5. Berkeley-yliopiston TCP/UDP “Header Data”- analyysi:
   http://robotics.eecs.berkeley.edu/~wlr/12203/transport-slides.pdf
   Luettu 30.4.2014.

6. ”Berkeley Socket” , Wikipedian C/C++ -ohjelmointi:
   http://en.wikipedia.org/wiki/Berkeley_sockets
   Luettu 9.4.2014.

7. Winsock 2.0 (Windows TCP-soketti ohjelmointi) C/C++ ohjelmointi:
   http://www.sockets.com/winsock2.htm
   Luettu 10.4.2014.

8. Verkko-soketti yleisesti , Wikipedia:
   http://en.wikipedia.org/wiki/Network_socket
   Luettu 26.4.2014.

9. TCP/IP-viitemalli:
http://fi.wikipedia.org/wiki/TCP/IP-viitemalli
Luettu 5.5.2014.

10. OSI-malli:
http://fi.wikipedia.org/wiki/OSI-malli
Luettu 5.5.2014.

11. HTTP-palvelimessa pyynnöt ”http-request” ja ”http-response :
http://code.tutsplus.com/tutorials/http-headers-for-dummies--net-8039
Luettu 4.5.2014.

12. HTTP-protokolla käyttää TCP-protokollaa tiedonsiirtona:
http://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-2--net-31155
Luettu 4.5.2014.

13. IP-osoite:
http://fi.wikipedia.org/wiki/IP-osoite
Luettu 30.4.2014.

14. Tiedostojakelupalveluohjelmat kuten e-mule, torrent ja muita:
http://www.codeproject.com/Articles/614028/www.emule-project.net
Luettu 18.4.201.

15. SSL /TLS-materiaalia:
http://en.wikipedia.org/wiki/Secure_Sockets_Layer
Luettu 22.4.2014.

16. OpenSSL-haavoittuvuus ”Heartbleed”:
http://heartbleed.com/
Luettu 22.4.2014.

17. Torrent download seed:
https://in.answers.yahoo.com/question/index?qid=20061221204347AAS06ef
Luettu 18.4.2014.

18. OSI-model Wikipedia (english):
http://en.wikipedia.org/wiki/OSI_model
Luettu 5.5.2014.

19. UDP:
http://en.wikipedia.org/wiki/User_Datagram_Protocol
Luettu 12.4.2014.

20. Verkkotopologia:
http://fi.wikipedia.org/wiki/Verkkotopologia
Luettu 10.4.2014.

21. Internet-palvelutarjoaja ”ISP”:
http://fi.wikipedia.org/wiki/Internet-palveluntarjoaja
Luettu 23.4.2014.

22. ”Winsock”-esimerkki:
http://www.binarytides.com/code-tcp-socket-server-winsock/
Luettu 19.4.2014.

23. ”Brute Force”-hyökkäys :
http://en.wikipedia.org/wiki/Brute-force_attack
Luettu 26.4.2014.

24. ”Ddos”-hyökkäys:
http://en.wikipedia.org/wiki/DDOS#Distributed_attack
Luettu 26.4.2014.

25. Oracle Java SSL-ohjelmointi:
http://docs.oracle.com/javase/7/docs/technotes/guides/security/jsse/JSSERefGuide.html
Luettu 20.4.2014.

Liite 1 : Monisäietoteutus

TCP-palvelimen monisäetoteutus Javalla:

Pääohjelma:

```
private static KayttajaSaie[] kayttaja_saikeet;
private static KayttajaSaie kayttajan_apusaie;
int enintaan_kayttajia;
kayttaja_saikeet =  new KayttajaSaie[enintaan_kayttajia];
/* Yhteyssäikeiden luonti, alustus ja käynnistys pääohjelmassa.
   Säikeiden määrä on sama kuin käyttäjien enimmäismäärä palvelimella.
*/
        for(int i = 0; i < enintaan_kayttajia; i++){

            kayttajan_apusaie = new KayttajaSaie();
            kayttajan_apusaie.initClient(palvelimen_soketti, ("käyttäjä"+i) );

             kayttaja_saikeet[i] = kayttajan_apusaie;
             kayttaja_saikeet[i].start();


        }
```

KäyttäjäSäie:

```
//====== Käyttäjän säikeen alustusfunktio ============
public void initClient(ServerSocket uusi_palvelimen_soketti , String uusi_kayttaja_nimi){
       palvelimen_soketti = uusi_palvelimen_soketti;
        kayttaja_nimi = uusi_kayttaja_nimi;
          paivitaYhteys = false;
   }
//====== Säie käynnistetään ========================
@Override
 public void run(){

         try{
                  kayttaja_yhteys_soketti = palvelimen_soketti.accept();
                  // Koodi on poistettu  try-catch lohkon välistä; tästä alkaa tiedonsiirtototeutus

         }catch(IOException e){
  // Yhteys on katkaistu tai yhteysongelma
 }
```