jamk.fi

# Verkkoyhteysongelmien simulointi ohjelmiston testausketjussa

Mikko Pöyhönen

Opinnäytetyö
Tammikuu 2016
Tekniikan ja liikenteen ala
Insinööri (AMK) ICT
Tietotekniikka
Tietoverkkotekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

**Kuvailulehti**

| Tekijä(t) | Julkaisun laji | Päivämäärä |
|---|---|---|
| Pöyhönen, Mikko | Opinnäytetyö, AMK | Huhtikuu 2016 |
| | Sivumäärä<br>87 | Julkaisun kieli<br>Suomi |
| | | Verkkojulkaisulupa myönnetty: x |

Työn nimi

**Verkkoyhteysongelmien simulointi ohjelmiston testausketjussa**

Tutkinto-ohjelma

Tietotekniikan koulutusohjelma

Työn ohjaaja(t)

Mika Rantonen

Jarmo Viinikanoja

Toimeksiantaja(t)

N4S@JAMK

Marko Rintamäki

Tiivistelmä

Opinnäytetyön toimeksiantajana oli N4S@JAMK-projektiryhmä. N4S@jamk-–projektiryhmä on osa Digilen Need For Speed -tutkimusohjelmaa, jonka tarkoituksena on testata reaaliaikaisia liiketoimintamalleja käytännössä ja luoda perusta kotimaisten ohjelmistoyritysten menestykselle.

Työn tavoitteena oli suunnitella ja toteuttaa palvelu, jonka avulla voidaan simuloida huonoja verkko-olosuhteita ja joka voidaan liittää osaksi ohjelmistotestausketjua. Palvelun avulla testauskohteesta voidaan löytää huonojen verkko-olosuhteiden aiheuttamia toiminnallisia puutteita ja näin ollen parantaa lopputuotteen toimivuutta.

Työ oli toteutettava siten, että palvelun käyttöönottoa varten ei tarvita erikoisosaamista tai laajempaa perehtymistä sen toimintaan. Työ toteutettiin Amazon Elastic Cloud 2 -ympäristössä virtuaalisella palvelininfrastruktuurilla.

Työssä palvelu liitettiin osaksi suorituskykytestausta, joka toteutettiin Locust-ohjelmalla. Testauskohteena toimi N4S@JAMK-projektiryhmän toteuttama Contriboard-palvelu. Suorituskykytestauksella pyrittiin löytämään palvelusta yleisimmät huonojen verkko-olosuhteiden aiheuttamat virheet Contriboard-palvelussa.

Työn tuloksena toteutettiin toimiva palvelu verkko-olosuhteiden simuloimiseen. Palvelun käyttöönottoa helpottamaan luotiin skripti, jonka avulla tarvittavien työkalujen ja ohjelmien asentaminen saadaan automatisoitua.

Avainsanat (asiasanat)

N4S@JAMK, Need For Speed, verkko-olosuhteiden simulointi, ohjelmistotestaus, Locust, Contriboard

Muut tiedot

jamk.fi

**Description**

| Author(s) | Type of publication | Date |
|---|---|---|
| Pöyhönen, Mikko | Bachelor's thesis | April 2016 |
| | 87 | Language of publication: Finnish |
| | | Permission for web publication: x |

Title of publication

**Simulation of poor network conditions in software testing chain**

Degree programme

Data Network Technology

Supervisor(s)

Rantonen, Mika

Viinikanoja, Jarmo

Assigned by

N4S@JAMK

Rintamäki, Marko

Description

The thesis was assigned by N4S@JAMK project group. N4S@JAMK project group is a part of Digile's Need For Speed research program. The purpose of Need For Speed research program is to test the real-time business-operations models in practice and provide the fundamentals for domestic programming business.

The goal was to design and implement a service that can be used to simulate bad network conditions, and which can be integrated into software testing chain. With the service integrated to the software testing chain, operational deficiencies caused by poor network conditions can be found, and the developers can improve the product functionality.

The product was implemented in such a way that the building of the service does not require special skills or a wider familiarization with its activities. The product was built in Amazon Elastic Cloud 2 environment with virtual server infrastructures. The service was used for performance testing chain carried out with Locust program. The test target for this thesis was Contriboard, a service developed by N4S@JAMK project. Performance testing service aimed to find the most common errors caused by poor network conditions in the Contriboard service.

As a result, the service was implemented to simulate poor network conditions. Also, a script was made to ease the installation of the service for users.

Keywords (subjects)

N4S@JAMK, Need For Speed, Locust, Contriboard, Network condition simulation, software testing

Miscellanous

# Sisältö

# Kuviot

# LYHENTEET

AWS = Amazon Web Services

EC2 = Elastic Cloud 2

Iaas = Infrastructure as a Service

IP = Internet Protocol

N4S = Need For Speed

NCP = Network Control Protocol

Paas = Platform as a Service

PPTP = Point to Point Tunneling Protocol

RSA = Rivest-Shamir-Adleman

Saas = Software as a Service

API = Application Programming Interface

VPN = Virtual Private Network

# 1 TYÖN LÄHTÖKOHDAT

## 1.1 Toimeksiantaja

Työn toimeksiantajana toimi Need For Speed -yhteistyökonsortioon kuuluva projektiryhmä N4S@JAMK. Neljä vuotta kestävään ohjelmaan kuuluu yhteensä kolmetoista laajaa yritystä, kuusitoista pientä tai keskisuurta yritystä ja yksitoista tutkimusinstituutiota ja yliopistoa. Need for Speed -ohjelman tavoitteena on kokeilla käytännössä reaaliaikaisia liiketoimintamalleja ja näin ollen luoda perusta suomalaisten ohjelmistoyritysten menestykselle. (N4S-ohjelma 2016.)

Projektilla on kolme pääasiallista painopistettä(N4S-ohjelma 2016.):

**1) Arvon tuottaminen reaaliajassa:** Suomalainen ohjelmistointensiivinen teollisuus on kokenut uudistuksia. Näitä uudistuksia tukemaan on kehitetty tekninen infrastruktuuri ja siltä vaaditut ominaisuudet.

**2) Syvällinen asiakastuntemus – parempi tuotto:** Suomessa ohjelmistointensiiviset teollisuuden alat käyttävät hyödyksi uutta teknistä infrastruktuuria ja informaatiokanavia parantamaan omaa asiakastuntemustaan. Tuntemalla asiakkaan tarpeet ja käyttäytymisen voidaan parantaa myyntiä huomattavasti.

**3) ”Elohopeabisnes” – Uutta rahaa etsimässä:** ”Elohopeabisnes” kuvaa yrityksen kykyä etsiä aktiivisesti uusia liiketoimintamahdollisuuksia ja tilanteen tullen laajentaa aggressiivisesti uusille markkinoille.

## 1.2 Työn tavoitteet ja vaatimusmäärittely

Mobiililaitteiden ja pilvipalveluiden yleistyttyä verkko-olosuhteet näyttelevät yhä suurempaa roolia palvelun saatavuudessa ja käytettävyydessä. Verkkopalvelusta tai ohjelmasta voidaan löytää täysin uudenlaisia virheitä verkko-olosuhteiden heikennettyä. Tämän takia huonot verkko-olosuhteet tulisi ottaa osaksi verifiointi-ketjua. Simuloimalla

huonoja verkko-olosuhteita verifiointivaiheessa voidaan määrittää ohjelman tai palvelun toiminnan kannalta minimi verkko-olosuhteet ja parantaa käytettävyyttä heikommilla verkkoyhteyksillä.

### 1.2.1 Tarkoitus ja kattavuus

Alustavan selvityksen mukaan valmiita palveluita verkko-olosuhteiden simuloimiseen ei löydy. Verkko-olosuhteita on kuitenkin mahdollista simuloida Linux-käyttöjärjestelmien Qdisc-työkalulla. Tämä on kuitenkin työlästä ja vaatii perehtymistä asiaan. Toimeksiantajan päämääränä on implementoida palvelu, jolla voidaan helposti simuloida heikkoja verkko-olosuhteita graafisesta verkkokäyttöliittymästä. Palvelu on ilmainen, ja käyttöönotosta laaditaan asianmukainen ohjeistus, jotta mahdollisimman moni yritys voisi ottaa sen osaksi omaa verifiointitestausketjua. Tuotteen varsinainen vaatimusmäärittely on esitetty kappaleessa ”Tuotteen määrittely ja käyttötarkoitus”.

### 1.2.2 Tuote ja ympäristö

Koska tuotteella simuloidaan verkossa tapahtuvia häiriöitä, tuotetta kutsutaan nimellä ”Häiriögeneraattori”. Häiriögeneraattorin alustana toimii Linux-palvelinkäyttöjärjestelmä, ja se implementoidaan pilvi-infrastruktuuriin. Häiriögeneraattorilla voidaan simuloida pakettien häviämistä, latenssia ja korruptoitumista. Palvelun hallinnoinnin ja pystyttämisen tulee olla helppoa ja yksinkertaista. Valmis tuote julkaistaan Github-palvelussa.

# 2 PILVIPALVELUT

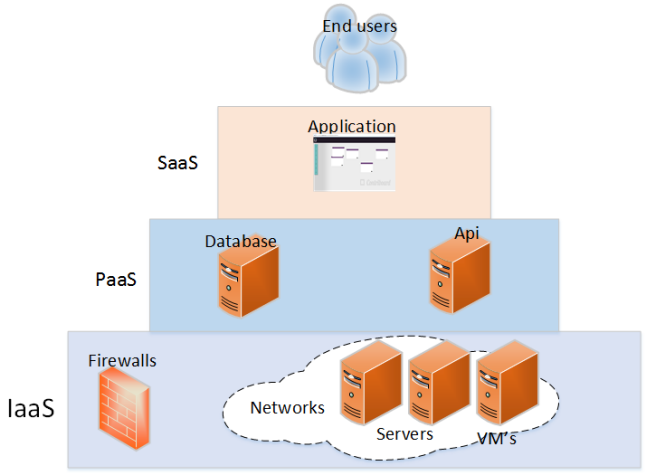
## 2.1 Yleistä

Pilvipalvelulla tarkoitetaan palvelua tai ympäristöä, joka tuotetaan asiakkaalle täysin virtuaalisesti. Pilvipalvelun tarkoituksena on luoda asiakkaalle palvelun tai ympäristön toi-

minnollisuus ilman asiakkaan tarkempaa tietämystä fyysisestä laitteistosta. Pilvipalveluiden ansiosta palveluita voidaan tuottaa kustannustehokkaammin, koska fyysisiin laitteisiin ei tarvitse investoida. Palvelin voidaan vuokrata ja tuottaa palvelut itse tai vaihtoehtoisesti maksaa valmiista palvelusta. Tätä mallia kutsutaan nimellä ”IT as a Service” eli ITaaS. (Wiley & Sons 2011.)

## 2.2 Palvelumallit

IT as a Service -malli jakautuu kolmeen eri tasoon. Näihin kuuluvat IaaS eli Infrastructure As a Service, PaaS eli Platform As a Service ja SaaS eli Software as a Service. (Furht & Escalante 2011.) Kuviossa 1. nähdään eri ITaaS-tasot.




Kuvio 1. ITaaS-tasot

### 2.2.1 Infrastructure as a Service

Infrastructure as a Service eli IaaS-tasolla asiakkaalle tarjotaan tyypillisesti tyhjä palvelin ja tallennustilaa. Palvelimeen voi liittää verkkorajapintoja, ja käyttäjä voi luoda palveluntarjoajan pilveen omia aliverkkoja. Asiakasta laskutetaan käytettyjen resurssien, kuten prosessorin tai muistin käytön mukaan. (Furht & Escalante 2011.) Tämän kategorian palveluntarjoajia ovat muun muassa Amazon Elastic Cloud 2 ja Digital Ocean:

**Amazon Elastic Cloud 2** on pilvipalveluympäristö, jossa voidaan virtualisoida verkkoja ja laitteita kuten palomuureja ja palvelimia. Amazon Elastic Cloud 2 eli EC2 tarjoaa ympäristön, johon voidaan luoda palvelimia valmiiden levykuvien pohjalta ja käyttää niitä kuten oikeita fyysisiä palvelimia. Käyttäjä voi myös luoda omia aliverkkoja ja liittää niihin halutut virtuaaliset laitteet. Virtuaalisia laitteita hallitaan konsoliyhteyden kautta. Amazon tarjoaa myös ”AWS client”-sovelluksen, jonka avulla ympäristöä voidaan hallita suoraan paikalliselta komentokehotteelta. Amazon Elastic Cloud -palvelun laskutus perustuu käytettyihin resursseihin. (Amazon Web Services 2016.)

**Digital Ocean** on Amazon EC2 -ympäristön tavoin ”hiekkalaatikko”, johon voidaan luoda virtuaalikoneita ja omia aliverkkoja. Digital Ocean tarjoaa myös oman ohjelmointirajapinnan, jonka kautta ympäristöä voidaan hallita http-pyynnöillä. Digital Ocean -palvelun laskutus perustuu käytettyihin resursseihin. Vaihtoehtoina ovat kuukausi–ja tuntiveloitus. (Digital Ocean 2016.)

### 2.2.2 Platform as a Service

PaaS eli Platform as a Service -tasolla voidaan vuokrata loogisia resursseja kuten tietokantoja tai ohjelmointirajapintoja. Näitä palveluita käytetään pääsääntöisesti ohjelmistokehitykseen. Sovellukset voidaan kehittää suoraan pilveen selaimen tai konsoliyhteyden avulla ilman, että paikallisella työasemalla on kehitystyökaluja. Platform as a Service palveluita ovat esimerkiksi Google AppEngine ja Microsoft Azure (Kavis 2014.):

**Google AppEngine** on alusta johon voidaan rakentaa verkkosovelluksia. AppEnginellä voidaan rakentaa tietokantoja ja ohjelmointirajapintoja. AppEngine tarjoaa myös automaattisen skaalauksen palvelulle. Automaattinen skaalautuminen tarkoittaa sitä, että mikäli käyttäjämäärät palvelussa nousevat, palvelun käytössä olevia resursseja kasvatetaan automaattisesti. AppEngine-palvelun kustannukset koostuvat käytetyistä resursseista. (Google 2016.)

**Microsoft Azure** on AppEnginen tavoin alusta, johon voi rakentaa tietokantoja, verkkosivuja ja ohjelmointirajapintoja. Tämän lisäksi Microsoft Azure-palvelusta löytyy myös

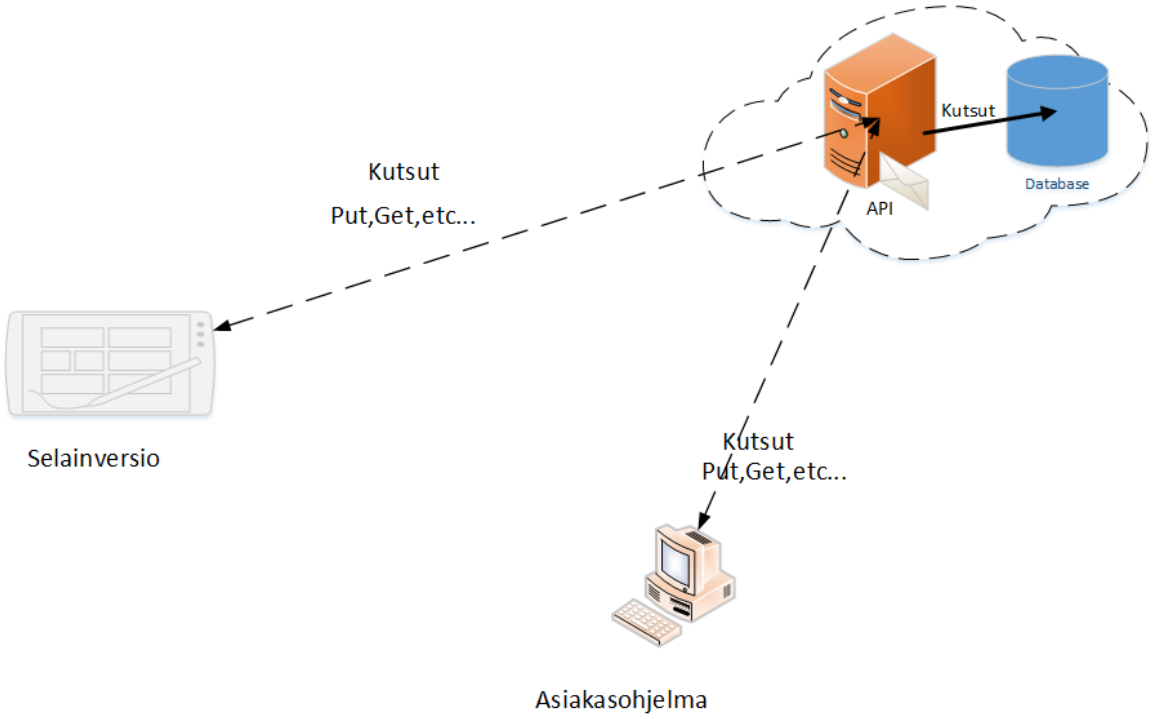
IaaS-tason palvelut. Palvelun kustannukset koostuvat muiden pilvipalvelujen tavoin vain käytetyistä resursseista, mutta poiketen muista palveluntarjoajista valittavissa on myös minuuttiveloitus. Azure tarjoaa ilmaisen tason, jolla voidaan tuottaa kymmenen verkkosovellusta tai ohjelmointirajapintaa täysin ilmaiseksi ensimmäisen vuoden ajan. Palveluntarjoaja lupaa saatavuudeksi 99.95%. (Microsoft 2016.)

### 2.2.3 Software as a Service

Software as a Service eli SaaS on ITaaSin ylin taso, jossa asiakkaille tarjotaan ohjelmistolisenssin sijasta täysin käyttövalmis ja eristetty palvelu, jota asiakas voi hallinnoida ja käyttää verkkoselaimen kautta. Tyypillisesti palveluista laskutetaan käytön ja käytössä olevien ominaisuuksien perusteella. (Kavis 2014.)

## 2.3 Palvelun rajapinnat

Ohjelmointirajapintoja kutsutaan yleisesti nimellä API eli Application Program Interface. API on kokoelma menetelmiä, työkaluja ja protokollia, joita käytetään sovellusten rakentamisessa. API määrittelee, kuinka palvelun eri komponentit ovat vuorovaikutuksessa toistensa kanssa ja mistä poluista tai millä kutsuilla eri palvelun osia kutsutaan. API:n avulla resursseja voidaan käyttää erilaisista ohjelmista tai graafisista käyttöliittymistä. Esimerkkinä sähköpostipalvelin, joka sisältää tietokannan käyttäjistä. Käyttäjä voi kirjautua palveluun tietokoneelle asennetusta asiakasohjelmasta tai vaihtoehtoisesti mobiililla verkkoselaimen kautta. Kuviossa 2. kuvataan, kuinka API käyttää tietokantaa asiakas-sovelluksesta riippumatta. (What is an API? 2012.)




Kuvio 2. API:n toiminta.

# 3 PALVELUN TESTAUS

## 3.1 Testauksen merkitys ohjelmistokehityksessä

Ohjelmistotestauksella on suuri vaikutus lopputuotteen tietoturvaan, toiminnollisuuteen ja saatavuuteen. Testauksen tarkoituksena on löytää järjestelmästä tai sen osasta virheitä ja selvittää täyttääkö se sille asetetut vaatimukset. (Pekkinen 2011.)

## 3.2 Testausstrategiat

Verifiointitestaus jakautuu kolmeen eri testausstrategiaan. Näihin strategioihin lukeutuvat ”Black-Box”, ”White-Box” ja ”Grey-Box” –strategiat.

**Black-box testing**

Black-Box-testauksessa eli mustalaatikkotestauksessa ohjelmaa tai oliota käsitellään ikään kuin mustana laatikkona. Tämä tarkoittaa sitä, että tietoa koodin sisällöstä ja ohjelman toiminnasta ei tarvitse olla. Mustalaatikkotestaus perustuu täysin ohjelmalle asetettuihin vaatimuksiin ja määräyksiin. Tämä tarkoittaa sitä, että tiedetään miten ohjelman tai koodin tulisi kussakin tilanteessa vastata. Testitapauksia voidaan tässä tapauksessa luoda heti, kun ohjelmalle asetetut vaatimukset ovat tiedossa. Kuviossa 3. on havainnollistettu mustalaatikkotestausta. (Myers, Sandler & Badgett 2012.)


Kuvio 3. Mustalaatikkotestaus

Mustalaatikkotestauksessa ohjelman toimivuutta testataan käyttäjän näkökulmasta, joten olisi suotavaa, että itse ohjelmoija tai ohjelmiston suunnittelija ei suorittaisi testausta. Mustalaatikkotestauksen heikkona puolena on testitapauksien suunnittelun vaikeus, myös huonot koodausmenetelmät jäävät mustalaatikkotestauksessa huomaamatta. (Myers, Sandler & Badgett 2012.)

**White-Box testing**

White-Box-testauksessa eli ”lasilaatikkotestauksessa” keskitytään ohjelman sisäisiin komponentteihin, toimintamalleihin ja koodirakenteeseen. Mustalaatikkotestaukseen

verrattuna testaajalta vaaditaan siis paljon enemmän tietoa itse ohjelmasta ja ohjelmointikielestä. (Koirala & Sheikh 2008.) Kuviossa 4. nähdään lasilaatikkotestauksen periaate.


Kuvio 4. Lasilaatikkotestaus

**Grey-Box testing**

Grey-Box-testaus eli harmaalaatikkotestauksessa hyödynnetään molempia lasilaatikko-ja mustalaatikkotestaustyylejä. Harmaalaatikkotestauksessa tutustutaan ohjelmaan vain sen verran, että tiedetään, miten se on rakennettu. Tämän jälkeen testit rakennetaan ja suoritetaan kuin mustalaatikkotestauksessa. Seurauksena voidaan rakentaa paljon kattavampia testitapauksia, joita ajetaan kuten mustalaatikkotestejä. (Koirala & Sheikh 2008.)

## 3.3 Testaustasot

Ohjelmakoodin testaaminen jakautuu karkeasti kolmeen testaustasoon. Tällä pyritään laadukkaampaan lopputulokseen ja siihen, että ohjelma täyttää sille asetetut vaatimukset jokaisessa vaiheessa. Testaustasoihin kuuluvat yksikkötestaus, järjestelmätestaus ja integraatiotestaus. (Myers, Sandler & Badgett 2012.)

### 3.3.1 Yksikkötestaus

Yksikkötestauksella tarkoitetaan ohjelman osan tai yksittäisten aliohjelmien testaamista paloittain. Mitä useammassa vaiheessa yksikkötestejä suoritetaan, sitä eheämpi koodi syntyy. Yksikkötestaus kuuluu hyvin vahvasti lasilaatikkotestaukseen, sillä yksikkötestauksissa keskitytään koodiin ja sen toimintaan.

Yksikkötestaukset suorittaa usein itse ohjelmoija, sillä testauksessa tarvitaan tietämys koodin sisällöstä. (Myers, Sandler & Badgett 2012.)

### 3.3.2 Järjestelmätestaus

Järjestelmätestauksessa testataan palvelua ja ympäristöä sille asetettujen vaatimusten puitteissa. Järjestelmätestauksessa yritetään todistaa, että järjestelmä ei täytä sille asetettuja vaatimuksia ja määrittelyjä. Testaustapaukset jakautuvat seuraaviin kategorioihin(Myers, Sandler & Badgett 2012.):

**Yhteensopivuustestaus,** mikäli olemassa oleva järjestelmä korvataan, tulee yhteensopivuus testata. Yhteensopivuustestillä näytetään toteen, että uusi järjestelmä ei täytä yhteensopivuuskriteerejä.

**Käytettävyyttä** joudutaan usein mittaamaan pidemmältä aikaväliltä, koska on käytännössä mahdotonta laskea, onko palvelu käytettävissä tulevana vuonna 2000 -vai 3000 tuntia.

**Tietoturvallisuustesteillä** yritetään näyttää toteen, että järjestelmä ei saavuta tietoturvavaatimuksia ja määrittelyjä.

**Penetraatiotestauksessa** järjestelmään tuotetaan kokeellisesti suuri määrä liikennettä ja seurataan ohjelman käyttäytymistä.

**Suorituskykytestauksella** pyritään osoittamaan, että ohjelma tai palvelu ei täytä sille asetettuja vaatimuksia suorituskyvyn suhteen. Mittauskohteina voivat toimia esimerkiksi ohjelman vastausaika tai liikennemäärä.

**Muistitestauksessa** palvelulle tai ohjelmalle syötetään huomattavan suuri määrä dataa tai kutsuja.

**Toiminnallisilla** testauksilla pyritään osoittamaan, että ohjelman toiminnot eivät täytä sille asetettuja vaatimuksia.

**Palautumistestauksella** yritetään osoittaa todeksi, että järjestelmä ei kykene palautumaan ohjelmointi tai laitteistovirheistä.

**Huollettavuustesteillä** testataan mahdolliset ylläpitoa tukevat järjestelmät, kuten lokitietojen tallennusjärjestelmä.

**Menettelytesteillä** testataan erilaisilta käyttäjiltä vaadittavat menettelyt, kuten ylläpitäjältä vaadittavat toimenpiteet esimerkiksi päivitystilanteessa.

**Asennusmenetelmien** käytettävyys tulee ottaa huomioon ja mikäli käytössä on automatisoitu asennusohjelma, tulee näille suorittaa omat testauksensa.

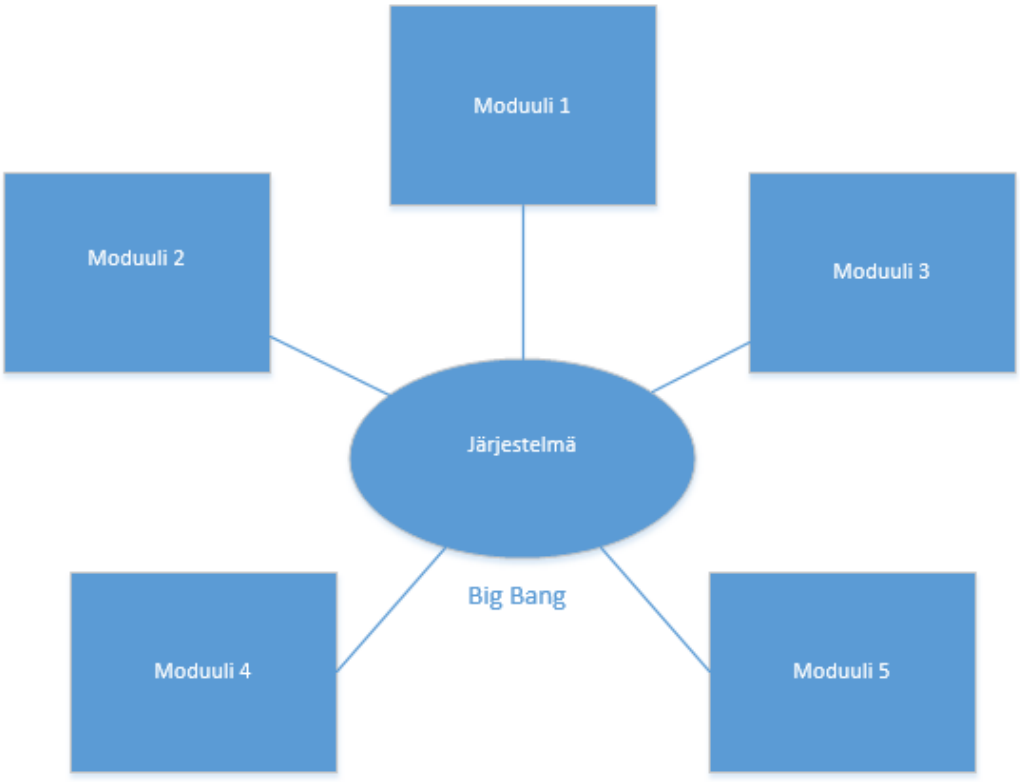
**Dokumentointitesteillä** tarkastetaan, ovatko palvelun tai ohjelman dokumentaatiot tarkoituksenmukaisia ja korrekteja.

**Konfiguraatiotesteillä** määritetään minimi ja maksimi laitekonfiguraatio, joka vaaditaan palvelun tai ohjelman ajamiseen.

### 3.3.3 Integraatiotestaus

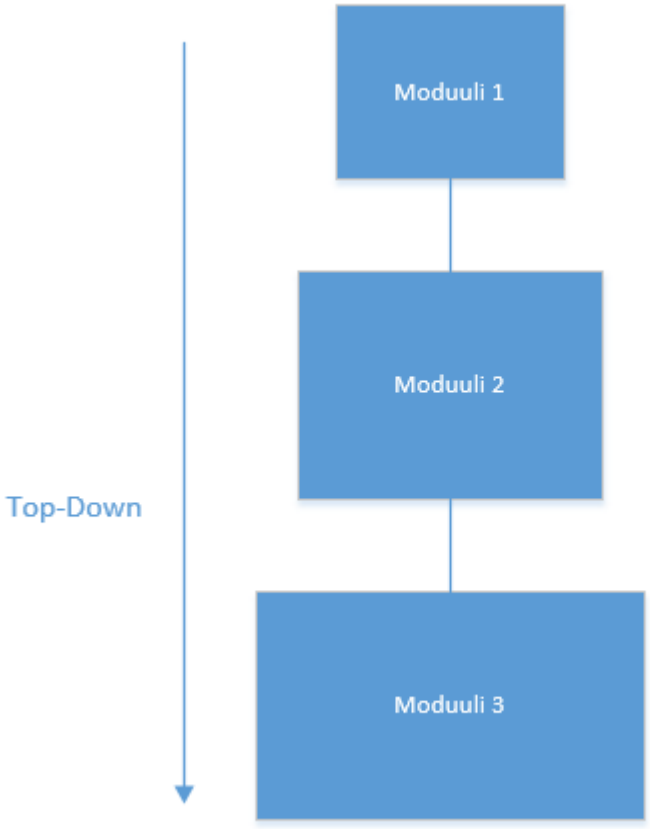
Integraatiotestaus on mustalaatikkostrategiaan kuuluva testaustaso, jossa keskitytään yksittäisten komponenttien välisiin rajapintoihin. Testauksessa apuna käytetään ohjelmistoajureita ja testitynkiä. Ohjelmistoajurit hoitaa testattavan komponentin käyttämisen ja kutsumisen. Testityngät taas toimivat testattavan komponentin kutsumina komponentteina. Integraatiotestaukseen on kolme lähestymistapaa (ISTQB 2016):

**”Big Bang”-integraatiotestauksessa** kaikki järjestelmässä oleva integraatio testataan kokonaisuutena. Tämän tyylin hyötynä on tehokkuus, kaikki tulee testattua kerralla. Haittana Big Bang -integraatiotestauksessa on vianselvitys: isossa kokonaisuudessa vian selvittäminen on vaikeaa. Kuviossa 5. havainnollistettu Big Bang -malli.




Kuvio 5. Big Bang -integraatiotestaus

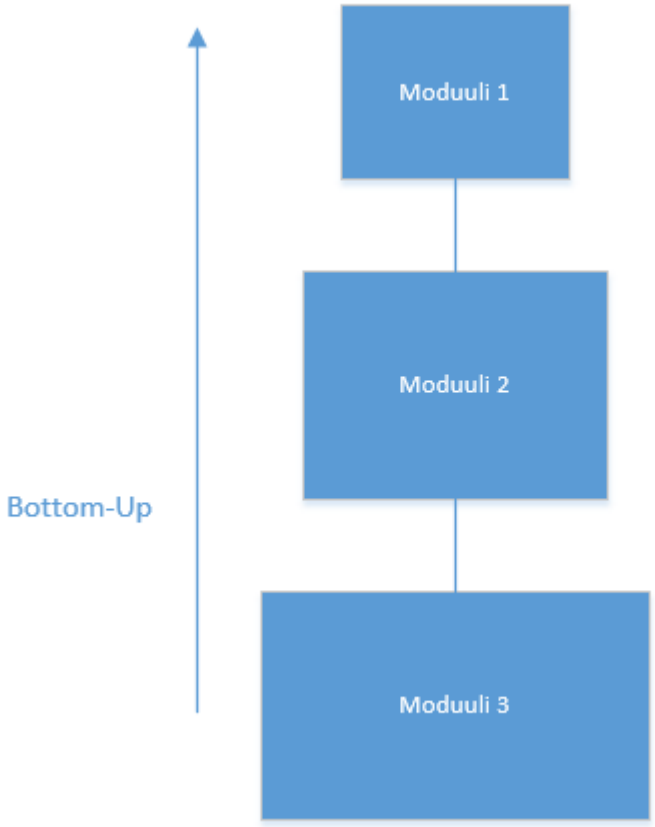
**”Top-down”-integraatiotestauksessa** järjestelmä testataan ohjelma-arkkitehtuurissa ylhäältä alas, eli testaaminen aloitetaan esimerkiksi graafisesta käyttöliittymästä. Tässä testausmallissa testien luominen on helppoa, koska testiajureita ei tarvitse kirjoittaa ja testityngät on helpompia luoda. Heikkoutena Top-Down-testaamisessa on se, että joidenkin perustoimintojen testaaminen tapahtuu vasta loppuvaiheessa. Kuviossa 6. nähdään ”Top-Down”-testausmalli.




Kuvio 6. Top-Down-integraatiotestaus

**”Bottom-up”-integraatiotestauksessa** järjestelmä testataan ohjelma-arkkitehtuurissa alhaalta ylöspäin. Etuna ”Bottom-Up”-testausmallissa on se, että testit voidaan suorittaa ohjelmointivaiheessa. Heikkoutena puolestaan on testiajureiden rakentaminen. Tämä on

huomattavasti vaativampaa kuin testitynkien luominen. Kuviossa 7. on ”Bottom-Up”-testausmalli.




Kuvio 7. Bottom-Up integraatiotestaus

## 3.4 Testausautomaatio

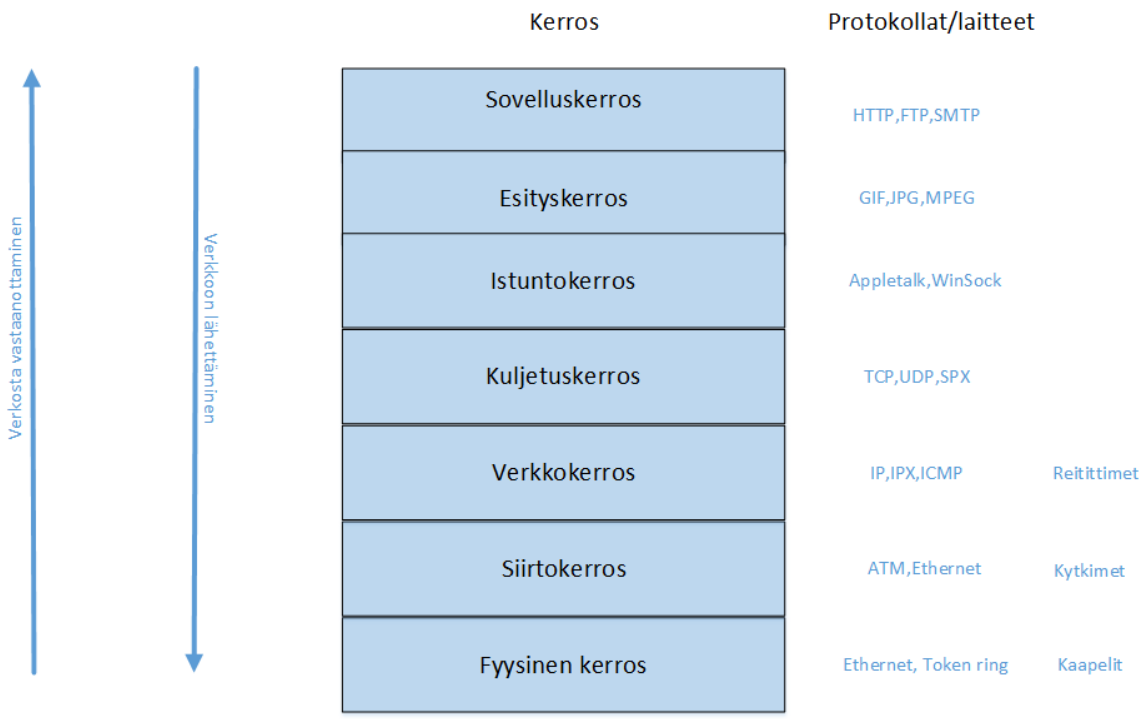
Testauksen automatisoinnilla pyritään tekemään enemmän testejä nopeammin ja sitomatta siihen ylimääräisiä resursseja. Panostamalla testausautomaation rakentamiseen yritys voi vähentää resurssien käyttöä manuaaliseen testaamiseen. Automaation avulla vähennetään myös riskejä, koska testit suoritetaan aina samalla tavalla verrattuna ihmisen tekemiin testeihin, jotka saattavat erota toisistaan inhimillisten tekijöiden takia. Automaation avulla myös testaustuloksista saadaan yhteenveto tulostettua nopeasti. (Fewster & Grahm 1999.)

Testausautomaatioon on olemassa lukuisia sovelluskehyksiä, jotka toimivat ympäristönä testiautomaatiolle. Sovelluskehyksiä on rakennettu eri ohjelmointikielille ja käyttötarkoituksiin. Tunnettuja sovelluskehyksiä ovat muun muassa Xunit-tuoteperhe, ja Robot Framework. (Jäsberg 2011.)

# 4 KÄYTETYT TEKNOLOGIAT

## 4.1 OSI-Malli

Open Systems Interconnection Reference Model eli OSI-mallia käytetään kuvaamaan tiedonsiirtoprotokollien eri kerroksia. OSI-mallissa on seitsemän kerrosta, joista jokainen kerros käyttää alemman kerroksen palveluita tarjotakseen palveluita ylemmälle kerrokselle. Kerroksien käsittelemän datan yksikkönä toimii PDU eli Protocol Data Unit. (Alani 2014.) Kuviossa 8. nähdään OSI-mallin seitsemän kerrosta ja niiden käyttämät protokollat.




Kuvio 8. OSI-mallin kerrokset

OSI-malli jakautuu seuraaviin kerroksiin(Alani 2014.):

**Fyysisellä kerroksella** määritetään tiedoston fyysinen media, joka siirtää bittejä. Fyysisen kerroksen PDU on bitti. Bitit siirretään läpinäkyvästi lähettäjältä vastaanottajalle. Fyysisellä tasolla tapahtuu myös fyysisen yhteyden muodostaminen ja datan limitys.

**Siirtokerroksella** käytettävä PDU on kehys. Siirtokerroksen ominaisuuksiin kuuluvat laitteiden tunnistaminen ja identifiointi, vuonhallinta, virheenkorjaus, datan kehystäminen ja yhteyden muodostaminen ja sulkeminen.

**Verkkokerroksella** käytettävä PDU on paketti. Verkkokerros hoitaa liikenteen reitittämisen, yhteyden luomisen kuljetuskerrokseen, vuonhallinnan ja datan lomittamisen. Verkkokerroksella tapahtuu myös IP-ja MAC-osoitteiden taltiointi.

**Kuljetuskerros** takaa erilaisia toimintoja yhteydettömille ja yhteydellisille palveluille. Kuljetuskerroksella dataa käsitellään segmentteinä, eli sen PDU on segmentti. Kuljetuskerros hoitaa yhteyden avaamisen ja sulkemisen tahojen välillä. Kuljetuskerros vastaa myös palvelun laadullisten parametrien monitoroinnista, datan järjestämisestä ja järjestyksen säilyttämisestä. Yhteydellisessä keskustelussa kuljetuskerroksella on vastuu virheenkorjauksesta, mutta yhteydettömässä virheenkorjausta ei toteuteta.

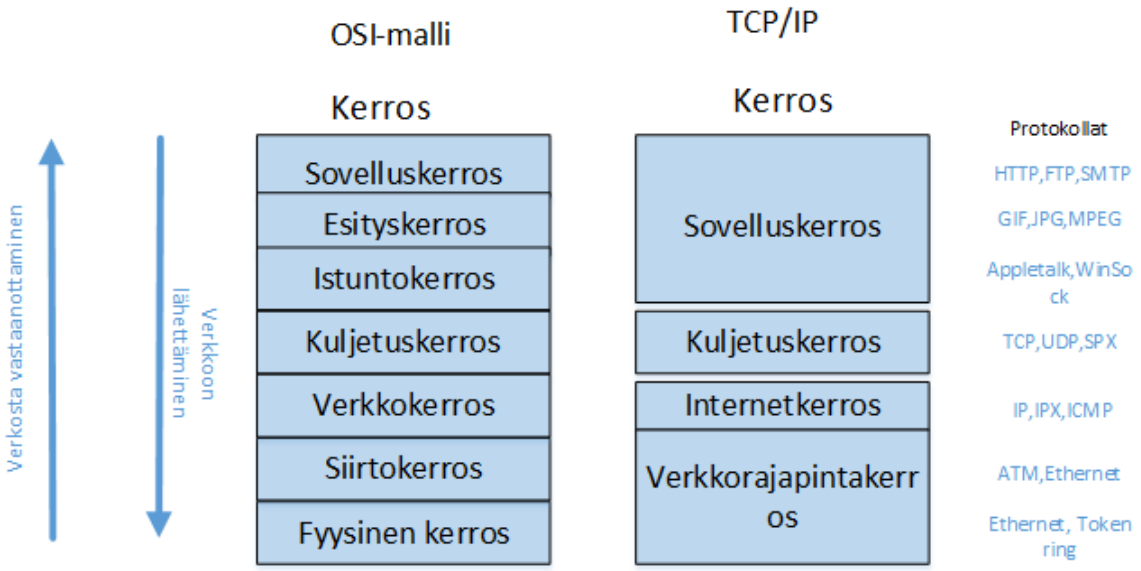
**Istuntokerros** mahdollistaa useiden yhtäaikaisten istuntojen luomisen samanaikaisesti. Istuntokerroksella ei ole omaa PDU:ta.

**Esityskerros** neuvottelee, missä muodossa data siirretään kullekin taholle. Esityskerros hoitaa myös datan pakkaamista ja salaamista.

**Sovelluskerros** määrittelee, mitä sovelluksen käyttäjä näkee. Sovelluskerroksella määritetään myös käytettävät tietoturvaominaisuudet kuten käyttäjän tunnistaminen ja päätetään, mitä palvelun laadun parantamiseen käytettäviä parametreja käytetään.

## 4.2 TCP/IP-malli

Transmission Control Protocol / Internet Protocol eli TCP/IP on useista eri verkkoprotokollista koostuva protokollapino, joka on rakennettu käytettävien protokollien pohjalta. Protokollapinon tehtäviin kuuluu vastata verkko-osoitteista ja liikenteen lähettämisestä sekä vastaanottamisesta. (Alani 2014.) Kuviossa 9. on esitetty TCP/IP-malli verrattuna OSI-malliin.




Kuvio 9. TCP/IP-malli verrattuna OSI-malliin.

TCP/IP-malliin kuuluvat seuraavat loogiset verkkokerrokset (Alani 2014.):

**Sovelluskerros** hoitaa datan esityksen, koodauksen ja keskustelun kuljetuskerroksen ja sovelluksen välillä. Sovelluskerroksen tehtävänä on siirtää data sovelluksesta kuljetuskerrokselle. OSI-mallista poiketen TCP/IP-malli ei sisällä esitys-ja istuntokerroksia. Sovelluskerroksella käytettävistä protokollista tunnetuin on Hyper Text Transfer Protokolla eli http ja secure http eli https. Muita käytettyjä protokollia ovat Simple Mail Transfer Protokolla eli SMTP, Post Office Protokolla 3 eli POP3 ja File Transfer Protokolla eli FTP.

**Kuljetuskerros** vastaa pakettien lähettämisestä ja vastaanottamisesta oikeassa järjestyksessä. Kuljetuskerroksessa käytetään TCP eli Transmission Control Protokollaa ja UDP eli User Datagram Protokollaa, joista TCP takaa luotettavan yhteyden ja UDP tarjoaa yhteydettömän tiedonsiirron kohteiden välillä.

Transmission Control Protokolla käyttää yhteyden muodostamiseen kolmivaiheista ”kolmitiekättelyä” ja sulkemiseen nelivaiheista ”nelitiekättelyä”. Pakettien lähetyksessä käytetään ”liukuva ikkuna”-tyyliä, jossa jokaisella paketilla on järjestysnumero ja vastaanottajalta odotetaan kuittausta sovitun pakettimäärän jälkeen. Transmission Control Protokolla tarjoaa myös menetelmät ruuhkanhallintaan ja tiedon eheyden tarkistamiseen. Näiden ominaisuuksien ansiosta TCP on usein käytetyin protokolla tiedonsiirrossa.

User Datagram Protokolla ei tarjoa virheenkorjausta ja se on yhteydeton protokolla. Tästä johtuen virheelliset paketit hylätään. User Datagram Protokollaa käytetään kohteissa, joissa ei haluta virheenkorjausta tai yhteydenmuodostamisesta aiheutuvaa kuormaa kuten reaaliaikaisen datan lähettämisessä ja domain-name service-pyynnöissä.

**Internetkerros** vastaa OSI-mallissa osittain siirtoyhteys ja verkkokerroksia. Internet kerros huolehtii liikenteen lähettämisestä, reitityksestä ja verkon diagnostiikasta. Laitteiden osoitteistus tapahtuu myös internetkerroksessa. Internetkerroksella käytettäviä protokollia ovat IP eli Internet Protocol, ARP eli Address Resolution Protocol ja IGMP eli Internet Group Management Protocol.

**Verkkorajapintakerros** pitää sisällä OSI-mallin fyysisen ja siirtoyhteyskerroksen. Tällä tasolla käytettyihin protokoloihin kuuluvat Ethernet, ATM ja ISDN.

## 4.3 Git

Git on avoimen lähdekoodin hajautettu versionhallintajärjestelmä, joka on kehitetty tukemaan ohjelmistokehitysprosessia. Git on alkujaan luotu avuksi Linux käyttöjärjestelmän kehittämiseen vuonna 2005 Linus Torvaldsin toimesta. Gitin avulla koodia voidaan

muokata ja tuottaa mistä vain, milloin vain. Esimerkkinä ohjelmistokehitystiimi, joka sisältää kymmenen henkilöä ja kaikilla pitää olla viimeisimmät versiot koodista ja jokaisella on oma osa, jota he muokkaavat. Gitin avulla viimeisin versio on aina tallennettuna yhteisessä repositoriossa. Repositorio on kaikkien projektiin osallistuvien saatavilla ja pieninkin muutos vaatii uuden version julkaisemisen. Projektin jäsenet voivat tehdä repositoriosta omia haaroja, johon omat muutokset tallennetaan. Lopuksi haarat liitetään master-repositorioon. Laajan suosion myötä Git on yleinen työkalu ohjelmistokehittäjien keskuudessa ja sen ympärille on rakennettu julkisia verkkopalveluita. (Gajda 2013.)

**Github** on verkkopohjainen työskentely-ympäristö ohjelmistokehittäjille. Käyttäjät voivat jakaa ja kehittää omaa koodiaan julkisissa tai yksityisissä repositorioissa. Github tarjoaa käyttäjilleen mahdollisuuden luoda ja haaroittaa repositorioita, luoda organisaatioita ja isännöidä verkkosivuja. Githubin wiki-osiossa voidaan säilyttää myös koodin dokumentaatiota. (Github 2015.)

## 4.4 Shell-skriptit

Shell-skriptit ovat ohjelmia, joita suoritetaan Unix-komentorivillä. Shell-skriptoissa käytetään ASCII-ohjelmointikieltä. Skriptillä voidaan suorittaa lähestulkoon kaikki käyttöjärjestelmän toiminnot, kuten ohjelmien ajaminen ja tiedostojen luominen. Yhdellä skriptillä voidaan esimerkiksi asentaa ja käynnistää useita palveluita ja tulostaa tästä informaatio käyttäjälle. (Shotts 2000-2016) Tässä työssä skriptejä käytetään osana orkestrointia konfigurointiin, tiedostojen hakemiseen ja palveluiden käynnistämiseen.

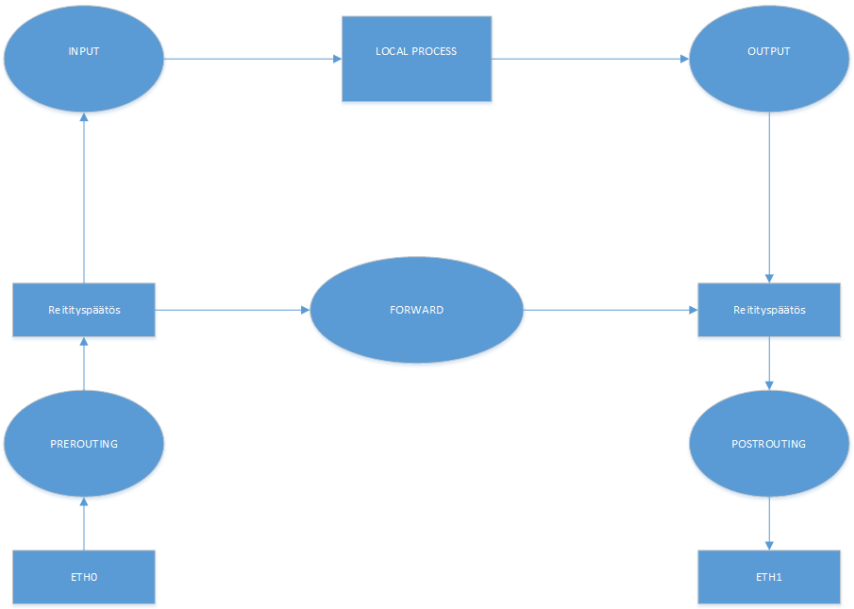
## 4.5 NetEm

NetEm on Linux -komponentti, jonka avulla voidaan simuloida erilaisia verkko-olosuhteita. NetEm sisältyy valmiina useimpiin Linux-distribuutioihin, kuten Fedora, Ubuntu, CentOs ja OpenSuse. NetEm–komponenttia käytetään komentokehotteessa ”tc”-työkalun avulla, joka sisältyy ”iproute2”-työkalupakettiin. Työkalulla voidaan simuloida verk-

korajapintakohtaisesti latenssia, pakettien tiputusta, uudelleenjärjestelyä, kahdentamista ja korruptointia. Työkalulla voidaan myös vaihtaa jonotusalgoritmia tai muuttaa algoritmin käyttämän puskurin kokoa. Latenssiin voidaan lisätä poikkeamia ”distribution”-toiminnolla. Distribution-toiminnolla voidaan simuloida oikeiden olosuhteiden aiheuttamaa poikkeumaa signaaliin. Vaihtoehtoina ovat ”normal” eli ei vaihtelua, ”pareto” ja ”paretonormal”. (Linux Foundation 2009.)

## 4.6 IPTables

Iptables on Linux-työkalu, jonka avulla säädetään Netfilter-pakettisuodatinta. Netfilter-suodattimella voidaan toteuttaa palomuurausta, osoitteenmuutosta ja reititystä. Paketteja käsitellään niille asetettujen sääntöjen perusteella. Paketeille asetettavia sääntöjä ovat ”DROP” eli hylkääminen ja ”ACCEPT” eli paketin hyväksyminen. Säännöissä määritetään myös ehdot, joilla paketit erotetaan. Näihin ehtoihin lukeutuvat lähde ja kohdeosoitteet, portit ja protokollat. Säännöt sidotaan eri ketjuihin. Näitä ketjuja ovat Prerouting, Input, Output, Postrouting ja Forward. Rajapintaan saapuvaa liikennettä käsitellään Input-ketjun sääntöjen mukaan. Vastaavasti rajapinnasta lähtevää liikennettä käsitellään Output-ketjun säännöin. Forward-ketjun sääntöjä käytetään, mikäli paketti vastaanotetaan, mutta sen kohdeosoite ei ole saman järjestelmän osoite vaan paketti lähetetään uudelleen toiselle vastaanottajalle. Prerouting-ketjun sääntöjä sovelletaan jokaiseen rajapintaan saapuvaan pakettiin ja vastaavasti Postrouting-ketjun sääntöjä rajapinnalta lähteviin paketteihin riippumatta siitä, mikä kohde tai lähdeosoite paketilla on. Kuviossa 10. on esitetty ketjujen prosessikaavio.(Linux 2016)



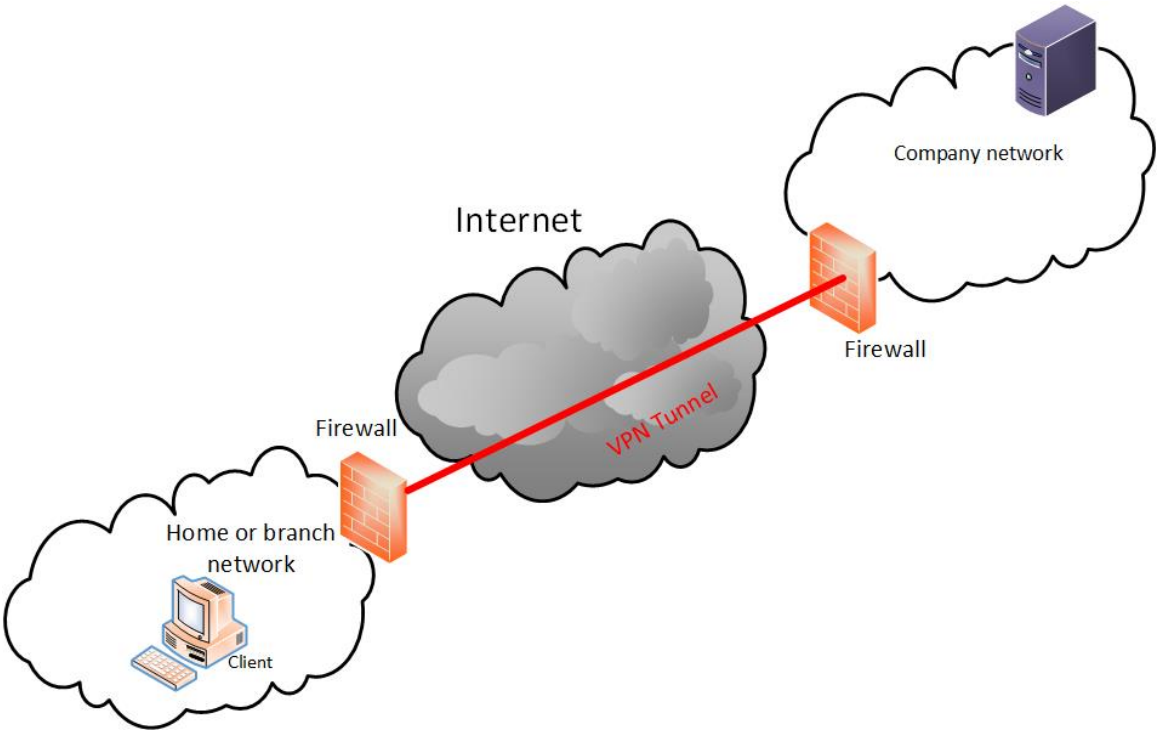

Kuvio 10. Netfilter-ketjujen prosessikavio

Iptables-työkalulla voidaan toteuttaa myös osoitteenmuutoksia. Osoitteenmuutokset tapahtuu muuttamalla joko lähde, tai kohdeosoite halutuksi osoitteeksi. Osoitteenmuutokset toteutetaan Prerouting -ja Postrouting-ketjuihin sidotuilla säännöillä. (Linux, 2016)

## 4.7 VPN

VPN eli Virtual Private Network on virtuaalinen yksityisverkko, jolla voidaan luoda tietoturvallinen tietoliikenneyhteys kahden pisteen välille käyttäen apuna julkista verkkoa. Yhteysosapuolet voivat tämän virtuaalisen tunnelin sisällä siirtää salattua dataa. Nykypäivänä on oleellista pystyä suorittamaan tämä toimenpide tietoturvallisesti. Hyvän tietoturvan komponentteja ovat esimerkiksi datan salaaminen ja käyttäjien autentikointi. VPN–tunneloinnissa paketit kapseloidaan, mikä tarkoittaa käytännössä sitä, että paketit sijoitetaan edelleen uusien pakettien sisään. Tämän tunnelin luomiseen on olemassa erilaisia protokollia, kuten IPSec, L2TP ja PPtP. Tässä työssä käytetään PPtP eli Point-To-

Point Tunnelointi Protokollaa. (Deal 2006.) Kuviossa 11. kuvataan VPN-tunnelin toimintaa.




Kuvio 11. VPN tunnelin toiminta

Point-To-Point protokolla on Microsoftin kehittämä tunnelointiprotokolla, joka on erittäin suosittu sen laajan tuen ansiosta. Protokollalla kapseloidaan PPP eli Point-to-Point Protocol-paketit IP-kehyksen sisälle. Point-To-Point tunnelointiprotokolla tunnetaan Microsoft Windows käyttöjärjestelmien lisäksi laajasti myös mobiili ja Unix käyttöjärjestelmissä. PPtP-protokollalla voidaan kapseloida TCP/IP ja IPX liikenteen lisäksi myös NetBEUI liikennettä. PPtP tarjoaa seuraavat ominaisuudet: (Deal 2006)

**Datan pakkaaminen** tapahtuu Microsoft Point-To-Point Compression-protokollalla, jolla pakataan Point-To-Point-Protokollan hyötykuorma.

**Salaaminen tapahtuu** Microsoft Point-To-Point Encryption-protokollan avulla. **Salauksessa käytetään** RSA RC4-salausalgoritmiä.

**Autentikointiin käytetään** PAP, CHAP tai EAP metodeja, joissa käyttäjän identiteetti tunnistetaan käyttäjänimen ja salasanan avulla.

**Datan toimitus** data paketoidaan PPP-pakettiin, jonka jälkeen se kapseloidaan PPTP-pakettiin ja edelleen IPvX tai NetBEUI-pakettiin.

**Asiakaslaitteiden dynaaminen osoitteistus**. PPtP tukee asiakkaiden dynaamista osoitteistusta, mikä tarkoittaa käytännössä sitä, että asiakkaan yhdistäessä asiakaslaitteille jaetaan IP-osoitteet samasta ennalta määritetystä aliverkosta. Tähän käytetään NCP eli Network Control Protokollaa.

# 5 TUOTTEEN MÄÄRITTELY JA KÄYTTÖTARKOITUS

## 5.1 Käyttötarkoitus

Tuotteen tulee sisältää valitut työkalut ja ohjelmat huonojen verkko-olosuhteiden simuloimiseksi. Verkko-olosuhteiden simulointi toteutetaan välityspalvelimella, jonka kautta liikenne lähetetään asiakaslaitteiden ja testauskohteen välillä. Välityspalvelinta on määrä käyttää osana verkkopalveluiden verifiointitestausta ja sen tulee toimia testauskohteesta ja käyttäjämäärästä riippumatta. Tuote tulee suunnitella ja toteuttaa siten, että se on käyttöönotettavissa mahdollisimman monessa ympäristössä ja käytettävissä mahdollisimman monella asiakaslaitteella. Myös käyttöönoton tulee olla helppoa ja sen apuna käytetään automatisointia skripteillä.

Työssä palvelua sovelletaan suorituskykytestauksessa. Suorituskykytestauksella saadaan selvitettyä, mitkä palvelun toiminnot ovat alttiimpia häiriöille huonojen verkko-olosuhteiden seurauksena.

## 5.2 Toiminnot

Palvelussa tulee olla helppokäyttöinen käyttöliittymä, jonka kautta voidaan yksinkertaisesti asettaa välityspalvelimen verkkorajapintaan halutut rajoitukset. Asetettaviin rajoituksiin kuuluu seuraavat ominaisuudet:

-Pakettien viivästys

-Pakettien pudotus

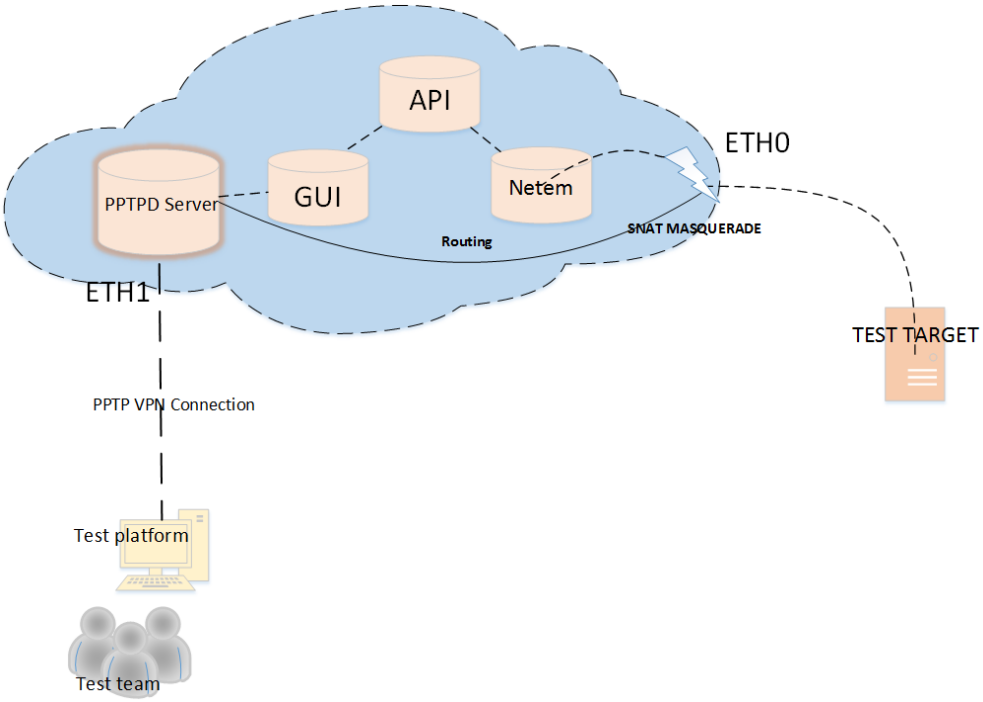
-Pakettien korruptointi

Rajoitukset voidaan asettaa myös automatisoidusti testauksessa käytettävien testausautomaatiotyökalujen avulla.

# 6 TUOTTEEN SUUNNITTELU

## 6.1 Topologia

Tuotteen topologia on suunniteltu vaatimusmäärittelyn ja käyttötapauksen pohjalta. Tuote toteutetaan yhdellä palvelimella, johon asennetaan tarvittavat työkalut ja ohjelmat. Kuviossa 12. on kuvattu tuotteen topologia ja käytetyt ohjelmat.




Kuvio 12. Tuotteen topologia

## 6.2 Ympäristö

Tuote toteutetaan ensisijaisesti Amazon Elastic Cloud 2 –ympäristössä, mutta automatisointi toteutetaan siten, että se on mahdollista käynnistää muissakin fyysisissä ja virtuaaliympäristöissä Red Hat Linux –käyttöjärjestelmissä. Työssä käyetään Amazon EC2 –palvelun valmista Amazon Linux AMI –levykuvaa, joka sisältää Centos 6 –käyttöjärjestelmän, jota on muokattu ympäristöön sopivaksi.

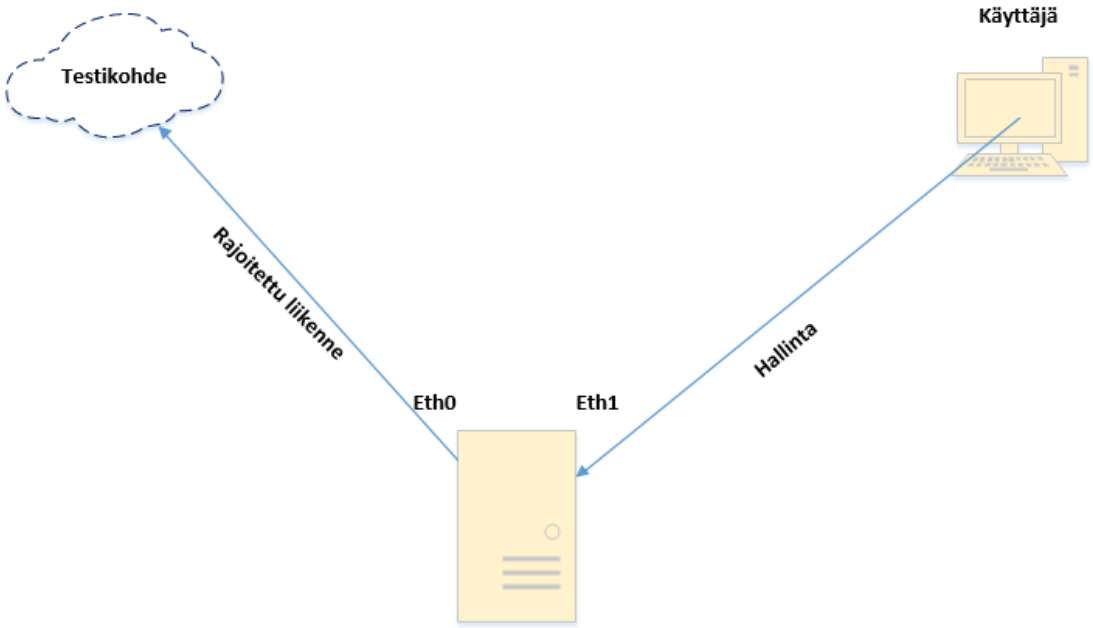
Amazon tarjoaa erilaisia järjestelmätyyppejä suorituskyvyn ja tallennustilan mukaan. Yleisimmät järjestelmätyypit ja niiden ominaisuudet ovat esitetty kuviossa 13.

| Type | vCPUs | Memory (GiB) | Instance Storage (GB) | EBS-Optimized Available | Network Performance |
|---|---|---|---|---|---|
| t2.nano | 1 | 0.5 | EBS only | - | Low to Moderate |
| t2.micro Free tier eligible | 1 | 1 | EBS only | - | Low to Moderate |
| t2.small | 1 | 2 | EBS only | - | Low to Moderate |
| t2.medium | 2 | 4 | EBS only | - | Low to Moderate |
| t2.large | 2 | 8 | EBS only | - | Low to Moderate |
| m4.large | 2 | 8 | EBS only | Yes | Moderate |
| m4.xlarge | 4 | 16 | EBS only | Yes | High |
| m4.2xlarge | 8 | 32 | EBS only | Yes | High |
| m4.4xlarge | 16 | 64 | EBS only | Yes | High |
| m4.10xlarge | 40 | 160 | EBS only | Yes | 10 Gigabit |

Kuvio 13. Amazon järjestelmätyypit

Järjestelmätyypin valinnassa tulee huomioida testausmenetelmät ja alueet. Mikäli välityspalvelinta käytetään vain toiminnallisessa testaamisessa, voidaan käyttää ”t2 large” tai ”m4 large” –järjestelmätyyppejä. Jos välityspalvelimen läpi toteutetaan suorituskykytestausta suurilla käyttäjämäärillä, tulee järjestelmätyypiksi valita liikennemäärästä riippuen järjestelmätyyppi, jossa verkon suorituskyky on vähintään korkea ja virtuaalisia prosessoriytimiä on vähintään neljä kappaletta. Vaikka palvelun tarkoituksena on rajoittaa verkon suorituskykyä, pyritään se tekemään muilla keinoilla, kuin luomalla palvelimesta pullonkaula testausketjuun.

Palvelun hallinta tapahtuu verkon yli, joten palvelu täytyy tuottaa siten, että hallintayhteyttä ei häiritä asetetuilla rajoitteilla. Tästä syystä palvelimelta vaaditaan kahta fyysistä rajapintaa. Kuviossa 14. on havainnollistettu kahden rajapinnan toimintaa.




Kuvio 14. Kahden rajapinnan toiminnallisuus

Amazon EC2 –ympäristössä tulee ensin luoda uusi rajapinta ja liittää siihen julkinen IP-osoite, jonka jälkeen rajapinta kiinnitetään järjestelmään. Huomioitavaa on, että rajapinta johon rajoituksia asetetaan, tulee olla ensimmäisenä järjestelmään lisätty rajapinta, koska käyttöjärjestelmäydin käyttää tätä oletusreittinä pakettien lähettämisessä ulkoverkkoon. Kyseisessä toteutuksessa molemmilla rajapinnoilla on julkinen IP-osoite, vaikkakin julkinen IP-osoite vaaditaan vain siltä rajapinnalta, johon käyttäjät yhdistävät julkisen verkon yli.

Tuotteen lähdekoodi ja automatisoinnissa käytetyt skriptit julkaistaan Github –palvelussa käyttäjän mikkopoyhonen repositorioissa. Repositorioiden osoitteet ovat:

***https://github.com/mikkopoyhonen/noise-builder***

ja

***https://github.com/mikkopoyhonen/noisetester***

## 6.3 Asiakasyhteyden suunnittelu

Aiempien implementaatioiden perusteella voidaan todeta, että välityspalvelimen ja asiakaslaitteen välille tarvitaan kiinteä yhteys eli niin sanottu tunneli. Tämä sen takia, koska useat palvelut paljastavat oman IP-osoitteen suoraan asiakaslaitteelle, jolloin reititys kääntyy pois välityspalvelimelta, mikäli tunnelia ei ole.

Esimerkiksi HAproxy –sovelluksella olisi mahdollista eritellä liikenne siten, että välityspalvelin toimisi ”Frontend” –palvelimena. Tämä kuitenkin vaatisi palvelukohtaista konfigurointia, joka ei toteuta tämän työn vaatimuksia testauskohteista riippumattomuuden osalta.

Asiakasyhteydessä käytetään VPN –yhteyttä, jolloin asiakaslaitteesta tulee osa palvelimen verkkoa ja kaikki liikenne saadaan ohjattua palvelimen kautta. VPN –yhteysvaihtoehdon valinnassa otetaan huomioon, että kyseessä on testausvaiheen työkalu ja se, että välityspalvelin ei ole pysyvä vaan se voidaan käynnistää uudelleen skriptin avulla vaikkapa jokaista eri testauskertaa varten. Näin ollen tietoturvallisuus ei ole olennaisin ominaisuus tässä implementaatiossa. Nykyiset verkkolaitteet tukevat hyvin laajasti kaikkia VPN –yhteyksissä käytettyjä protokollia, kuten IPSec ja PPtP.

Käyttöön valitaan PPtP VPN –yhteys, koska se ei vaadi erillisiä sertifikaatteja käyttäjille ja on näin ollen helppo käyttää ja konfiguroida automaattisesti. PPtP takaa myös parhaan suorituskyvyn tietoturvan kustannuksella.

Linux –käyttöjärjestelmissä eniten käytetty sovellus PPtP –palvelun luomiseen on PPtPd. PPtPd luo PPtP VPN –palvelun, johon käyttäjät kirjautuvat ennalta asetetuilla käyttäjätiedoilla. Palvelu jakaa myös asiakaslaitteille IP-osoitteet määritetystä IP-avaruudesta. PPtPd vaatii toimiakseen PPP –taustaprosessin. Asiakaslaitteet saavat IP-osoitteet yksityisestä osoiteavaruudesta, joten palvelimella tulee käyttää Eth0 -verkkorajapinnassa eli loogisesti ajateltuna ulospäin lähtevässä rajapinnassa käytössä Network Address Translation, eli NAT –ominaisuus. NAT –ominaisuus konfiguroidaan siten, että kaikki yksityiset osoitteet muutetaan MASQUERADE –määritelmällä verkkorajapinnan osoitteeksi.

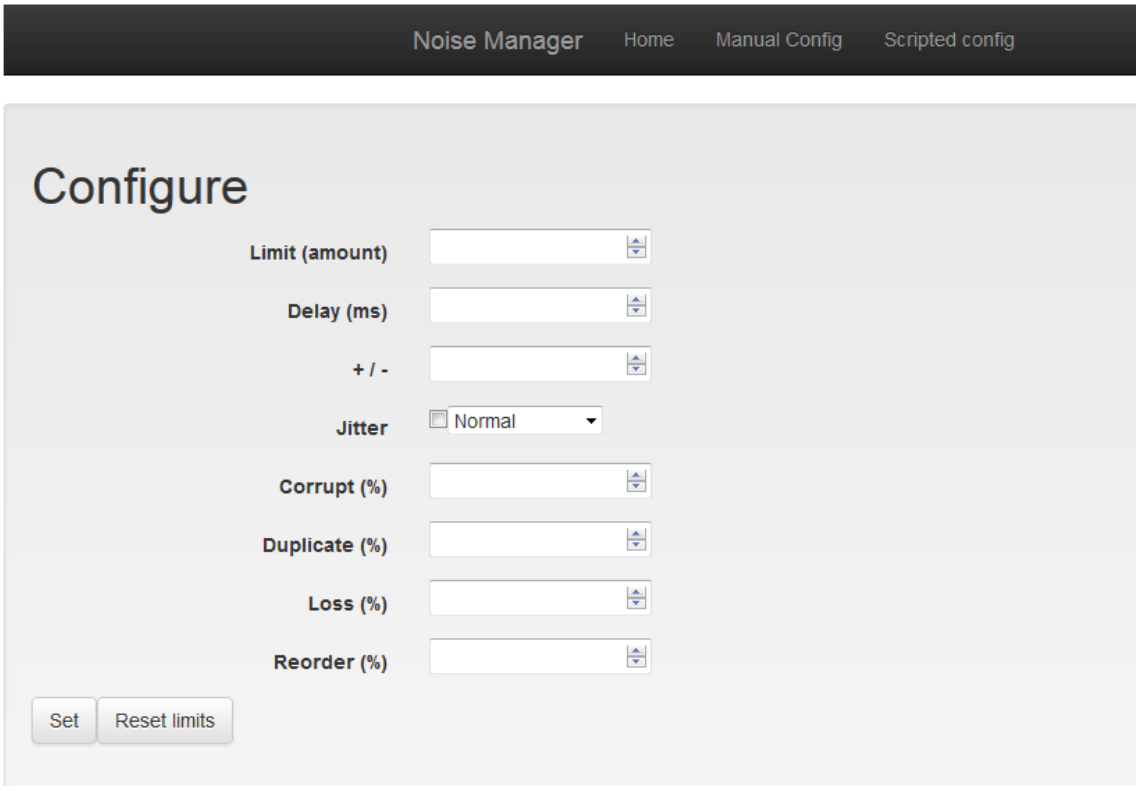
## 6.4 Käyttöliittymä

Käyttäjärajapintana ja käyttöliittymänä käytetään aiempaan toteutukseen rakennettua Noise –käyttöliittymää, johon tehdään vain pieniä muutoksia. Noise –käyttöliittymä on toteutettu Node.js –ohjelmalla. Käyttöliittymää hallinnoidaan selaimella ja sen kautta palvelimella suoritetaan Shell –komentoja. Shell –komennoilla palvelimella käytetään Traffic Control –työkalua, jolla Linux-ytimeen asetetaan halutut parametrit verkko-olosuhteiden simuloimiseksi. Esimerkkikomento, joka syötetään graafisen käyttöjärjestelmän toimesta on:

***tc qdisc add dev eth0 root netem delay 100***

Tällä komennolla *eth0* –verkkorajapinnan lähtevään liikenteeseen lisättäisiin 100 millisekuntia viivettä.

Kuviossa 15. nähdään käyttöliittymän hallinnointi –osio.




Kuvio 15. Noise –käyttöliittymä

Muutettavia parametreja ovat:

**Limit** - kuinka moneen pakettiin rajoitus vaikuttaa

**Delay** – Pakettien viive

**+/-** - Vaihteluväli viiveelle

**Jitter** – Vaihtelun tyyli. Vaihtoehdot: Normal, Pareto, Paretonormal

**Corrupt (%)** – Pakettien korruptointi

**Duplicate (%)** – Pakettien monistaminen.

**Loss (%)** – Pakettien pudotus

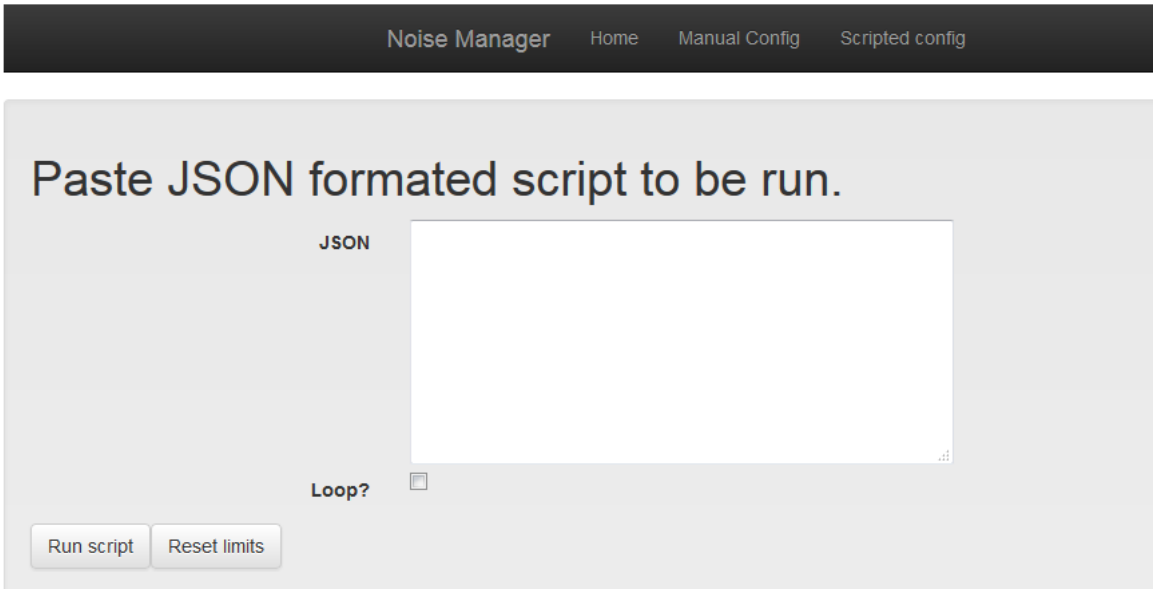
**Reorder (%)** – Pakettien uudelleenjärjestely

Käyttöliittymässä on myös mahdollista syöttää parametrit skriptin avulla. Tässä toiminnossa parametrit voidaan asettaa olemaan voimassa määrätyn ajan. Skriptit syötetään ”Scripted Config” –näkymään JSON -muodossa. Esimerkki skriptistä:

```
{
"0": {
 "loss": 20,
 "delay": 5
},
"5": {
 "loss": 10,
 "delay": 2
}
}
```

Tämä asettaisi 0 sekunnin kohdalla arvot: *Packet loss 20%* ja *Delay 5%* . Viiden sekunnin kohdalla arvot vaihdettaisiin seuraavasti: *Packet loss 10%* ja *Delay 2%* .

Scripted Config –näkymä nähdään kuviossa 16.




Kuvio 16. Scripted Config –näkymä

Scripted Config –toimintoa voidaan käyttää esimerkiksi testauksessa käytettävien testausautomaatiotyökalujen avulla.

Vanhaan käyttöliittymään tehdään seuraavia muutoksia:

**Haproxy** poistetaan repositoriosta, koska sitä ei käytetä tässä toteutuksessa.

**Contriboard –yksilöinti** poistetaan, koska uuden palvelun on määrä toimia testikohteesta riippumatta.

**Target näkymä** poistetaan, koska uudella toteutustavalla testikohdetta ei tarvitse erikseen määrittää.

**Rate –asetus** poistetaan Manual Config –näkymästä, koska jonotuspuskurin säätäminen voi aiheuttaa koko järjestelmän kaatumisen.

**API Qdisc –komennon** kohderajapinta vaihdetaan lo eli loopback -rajapinnasta eth0 -rajapintaan, koska uudessa toteutuksessa parametrit asetetaan fyysiselle rajapinnalle.

Mainittujen muutosten lisäksi repositoriosta siistitään myös ylimääräiset skriptit ja tiedostot, joita ei tässä toteutuksessa käytetä.

## 6.5 Tietoturva

Tietoturvallisuutta on suunniteltu ottaen huomioon toimeksiantajan vaatimukset helposta käyttöönotosta ja palvelun kiinnostavuus mahdollisten hakkereiden ja haittaohjelmien näkökulmasta.

Palvelussa käytetään PPtP VPN –yhteyttä, jossa käytetään paikallista autentikointia ja data salataan käyttäen MS-Chap v2. eli Microsoft Challenge-Handshake Authentication Protokollaa. Tämä yhteysmuoto ei ole turvallisin mahdollinen, mutta ottaen huomioon palvelun mahdolliset riskit, voidaan todeta PPtP VPN turvallisuustaso riittäväksi tähän toteutukseen. Huonomman salaustason kustannuksella PPtP vaatii palvelimelta vähemmän suorituskykyä verrattuna esimerkiksi L2tp/Ipsec –yhteyteen. PPtP –autentikointitietoihin voidaan haluttaessa lisätä myös sallitut IP-osoitteet käyttäjän toimesta.

Käyttöliittymän shell-rajapinta on toteutettu siten, että sen kautta ei pääse syöttämään järjestelmään muita kuin linux Qdisc –komentoja. Näin ollen, mikäli riski tapahtuisi, pahin skenaario olisi datan kaappaus ja palvelunestohyökkäys.

Amazon EC2 –ympäristössä voidaan Security Groups –toiminnon avulla luoda sääntöjä, joilla estetään tai sallitaan liikennettä lähdeosoitteen, kohdeosoitteen, portin ja protokollan perusteella. Testaus ja kehitysvaiheessa sääntöjä ei kuitenkaan luoda.

## 6.6 Automatisointi

Automatisointi toteutetaan Linux Shell –skriptin avulla. Automatisoinnissa pyritään siihen, että käyttäjän tarvitsee tietää järjestelmästä mahdollisimman vähän. Skriptojen tulee olla sellaisia, että ne ovat ympäristöstä riippumattomia. Skriptillä pyritään suorittamaan seuraavat toiminnot:

**PPtPd –sovelluksen asentaminen, konfigurointi ja käynnistys.**

**IPtables –työkalun konfigurointi**

**sysctl konfigurointi**

**Oletusarvojen asettaminen Netem –työkalulle**

**Node –sovelluksen lataaminen, asentaminen ja konfigurointi**

**NPM –sovelluksen lataaminen, asentaminen ja konfigurointi**

**Noise –käyttöliittymän lataaminen, rakentaminen ja käynnistäminen**

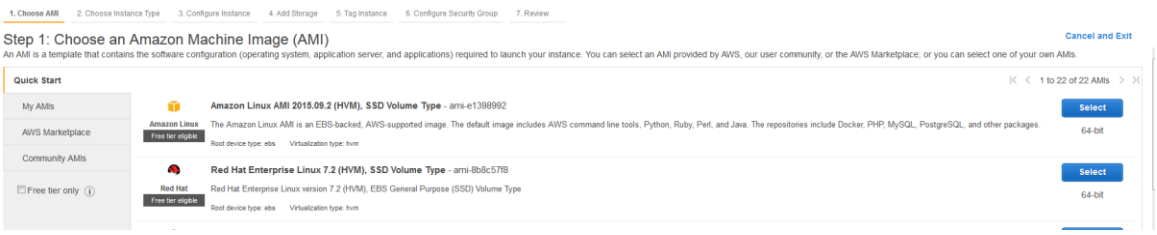
Automatisoinnissa käytetään Shell –skriptiä siitä syystä, että yhden palvelimen toteutuksessa käyttäjälle jää vähiten työtä ja palvelun rakentaminen ei vaadi asiantuntemusta automatisoinnista.

# 7 TUOTTEEN TOTEUTUS

## 7.1 Ympäristön toteutus

Ympäristön rakentaminen aloitetaan kirjautumalla Amazon EC2 –palveluun. Uuden palvelimen luontiin päästään Elastic Cloud 2 Management Consolen Instances –näkymästä. Uuden palvelimen luontiin päästään *Launch Instance* –painikkeen kautta.

Ensimmäisenä valitaan palvelimessa käytettävä käyttöjärjestelmä Amazon –palvelun valmiista AMI eli Amazon Machine Images –listasta. (Kuvio 17.)

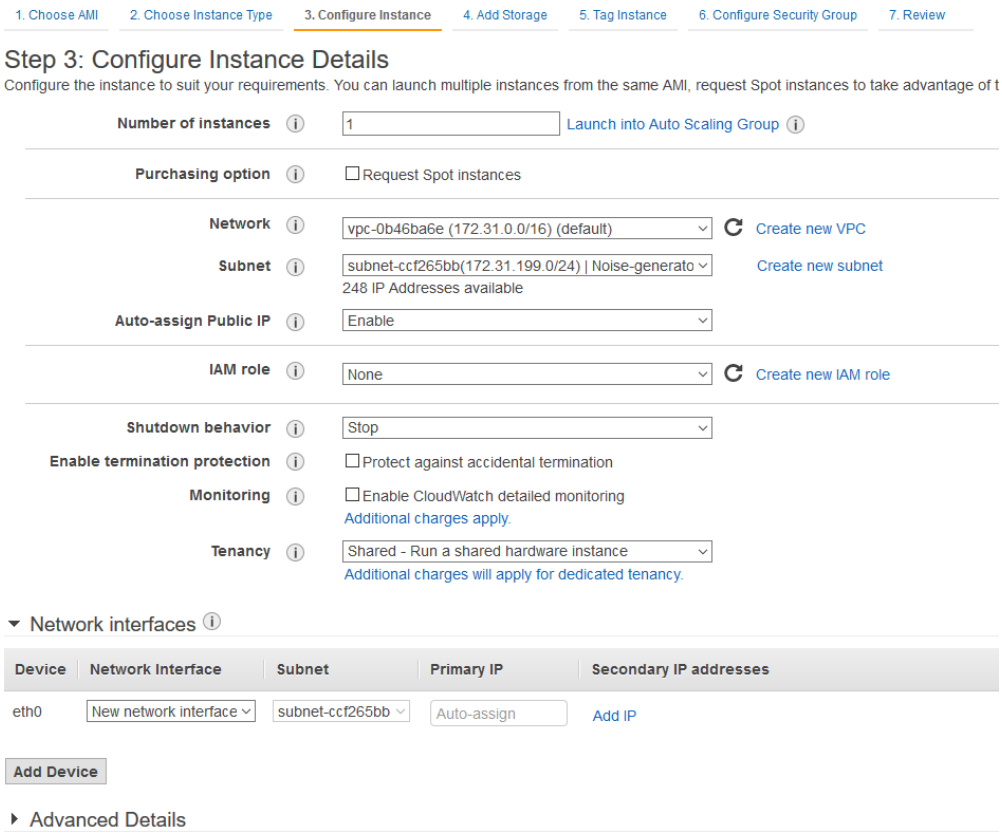



Kuvio 17. Amazon Machine Image –listaus

Tähän toteutukseen valitaan Amazon Linux AMI.

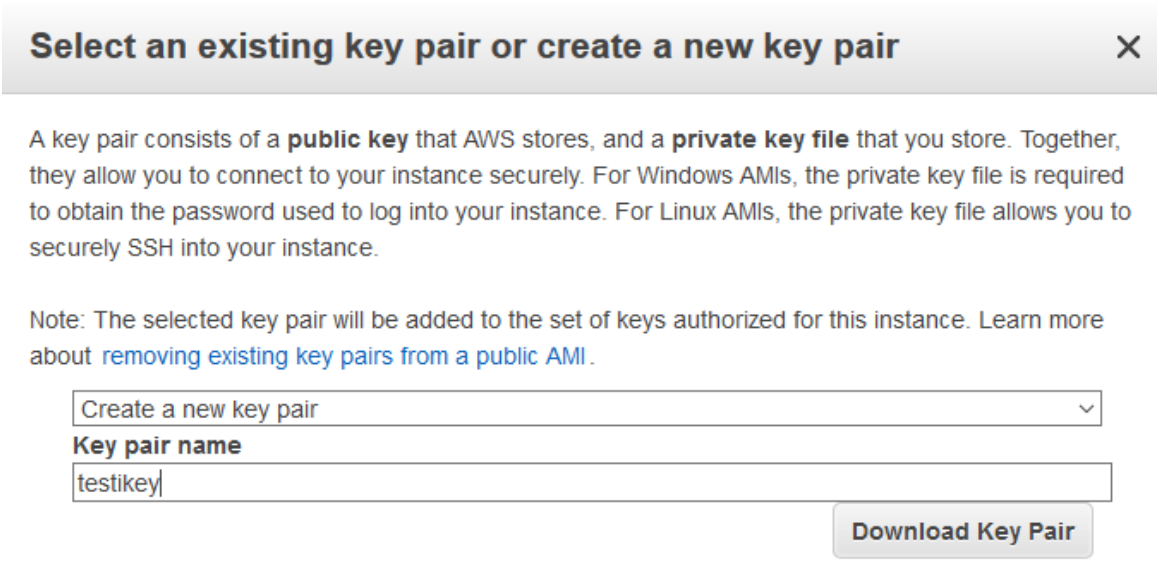
Palvelimen tyypiksi valitaan t2.micro, koska kehitysvaiheessa palvelimelta ei vaadita suorituskykyä.

Konfigurointivaiheessa palvelimelle valitaan aliverkko, johon se liitetään. Konfigurointivaiheessa asetetaan myös kohta Auto-Assign Public IP arvoon *Enable*, jotta palvelin saa automaattisesti julkisen IP-osoitteen. Tässä vaiheessa palvelimelle ei vielä lisätä muita verkkorajapintoja. Kuviossa 18. nähdään palvelimen konfigurointivaihe.




Kuvio 18. Palvelimen konfigurointi

Konfiguroinnin jälkeen palvelimelle asetetaan käytettävä Security Group, mikäli haluttu Security Group on jo olemassa. Tämän jälkeen valitaan Launch ja luodaan uusi tai valitaan olemassa oleva avainpari yhteyden luomista varten.(Kuvio 19.)



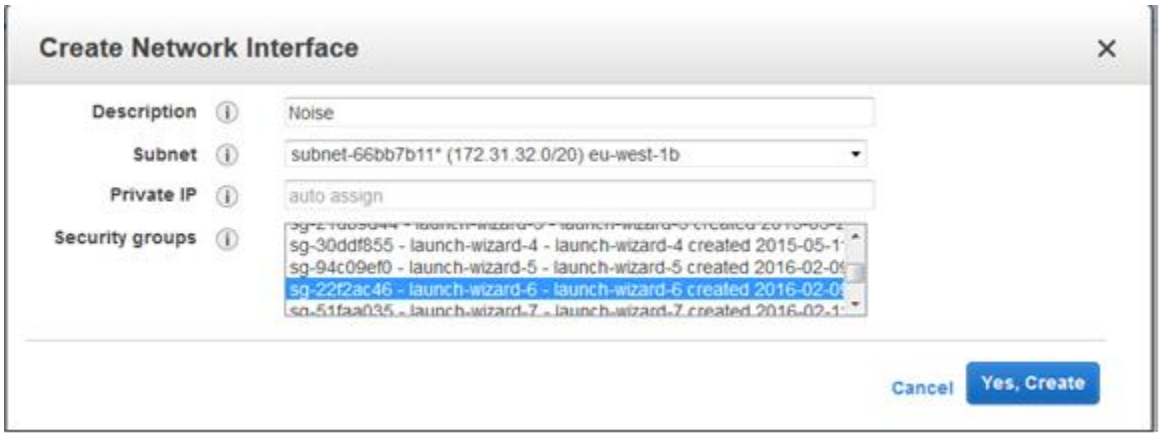

Kuvio 19. Avaimen luominen

Avain tallennetaan ja sitä käytetään hallintayhteyden muodostamisessa palvelimelle.

Ennen toisen verkkorajapinnan kiinnitystä luodaan uusi julkinen IP-osoite. Amazon EC2 –ympäristössä käytetään Elastic IP–osoitetta. Elastic IP–osoitteiden hallinnointi tapahtuu Elastic IP's –näkymästä. Uuden osoitteen luomiseksi valitaan painike *Allocate New Address*.

Seuraavaksi luodaan uusi virtuaalinen verkkorajapinta ja liitetään se luotuun virtuaaliseen palvelimeen. Rajapinta luodaan vasta palvelimen luomisen jälkeen, jotta molemmat verkkorajapinnat saavat julkisen IP-osoitteen. Virtuaalisten verkkorajapintojen hallinta tapahtuu EC2 Management Consolen Network Interfaces –näkymästä. Uuden verkkorajapinnan luominen aloitetaan *Create Network Interface* –painikkeesta.

Verkkorajapinnalle valitaan sama aliverkko ja Security Group kuin palvelimelle aiemmin.(Kuvio 20.)

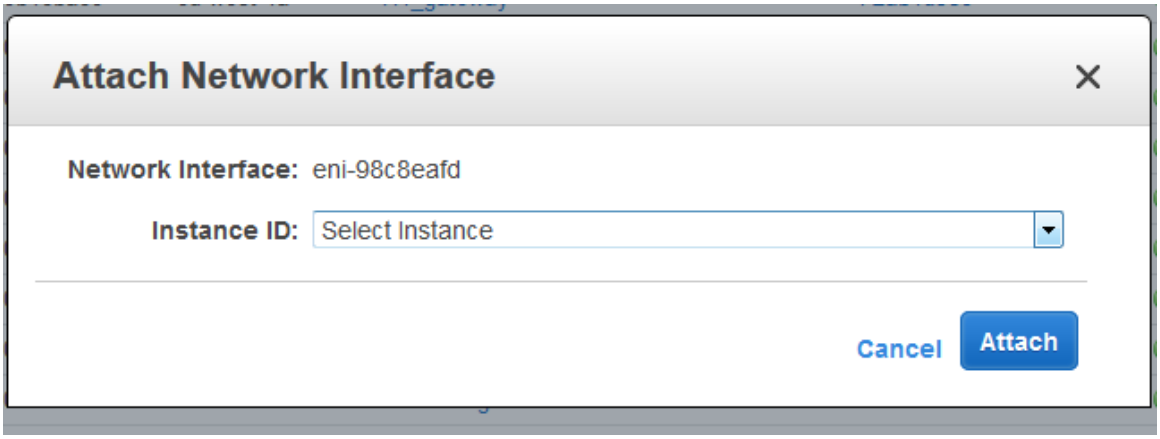



Kuvio 20. Verkkorajapinnan luominen

Kun rajapinta on luotu, asetetaan siihen aiemmin luotu elastinen IP-osoite. IP-osoite lisätään painamalla rajapinnan kohdalta oikealla hiiren painikkeella ja valitsemalla *Associate Address*.(Kuvio 21.)

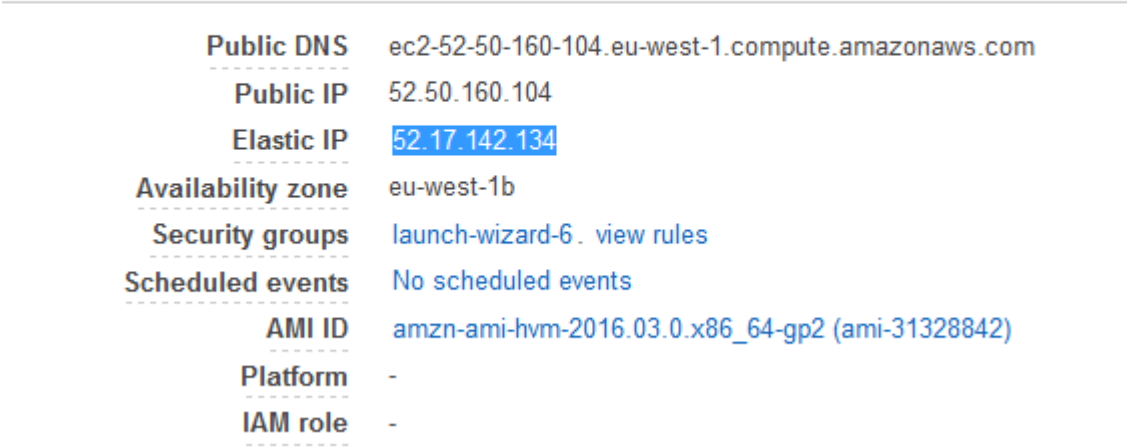

Kuvio 21. IP-osoitteen lisääminen verkkorajapintaan

Viimeisenä verkkorajapinta liitetään virtuaaliseen palvelimeen painamalla hiiren oikealla painikkeella verkkorajapinnan kohdalta ja valitsemalla *Attach*.(Kuvio 22.)




Kuvio 22. Verkkorajapinnan liittäminen palvelimeen

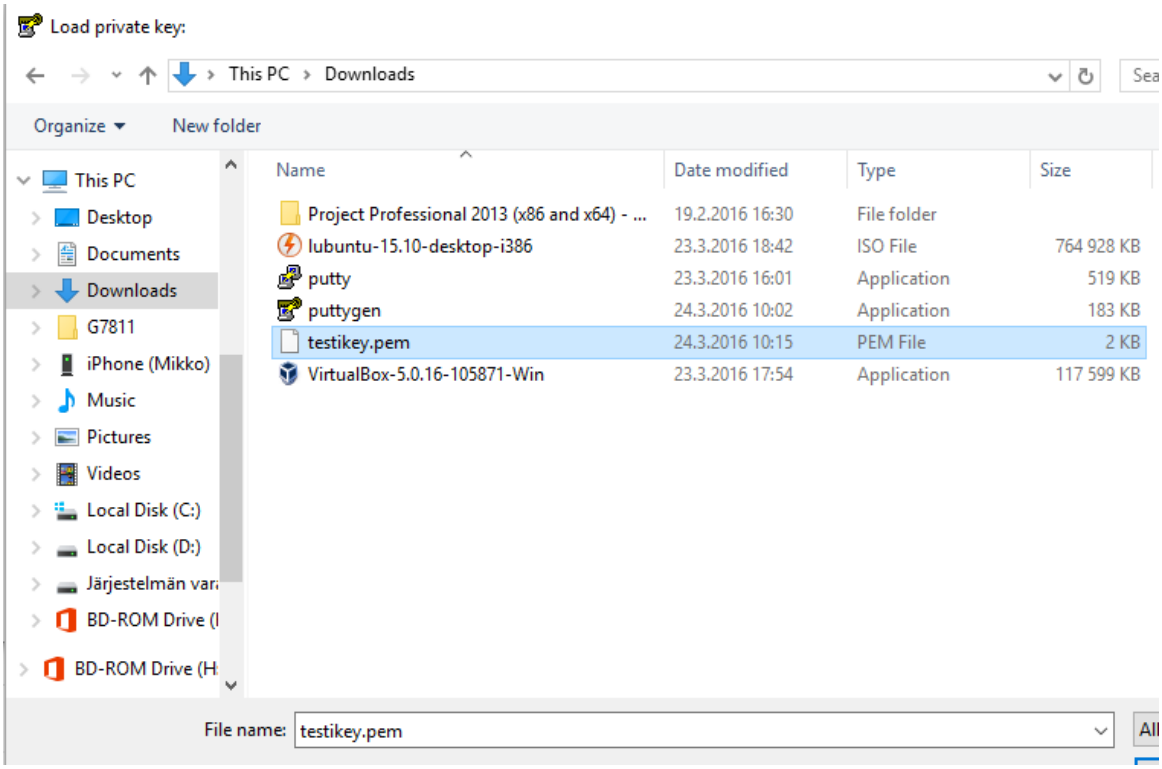
Tarkastetaan vielä *Instances* –näkymästä, että liittäminen onnistui ja palvelimella on kaksi julkista IP-osoitetta.(Kuvio 23.)




Kuvio 23. Julkiset IP-osoitteet

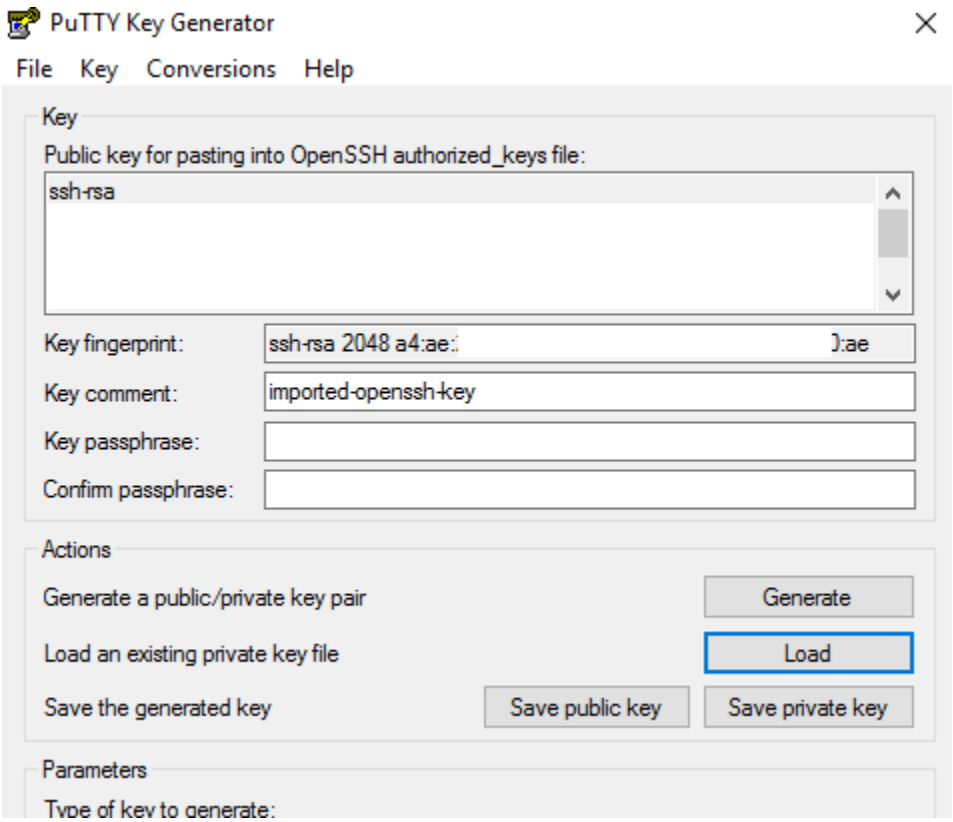
Hallintayhteys palvelimeen luodaan tässä tapauksessa PuTTY –ohjelmalla. Ennen yhdistämistä aiemmin ladattu avain täytyy muuttaa ohjelmalle sopivaan formaattiin. Avaimen muokkaaminen tapahtuu PuTTY Key Generator –ohjelmalla.

Avain ladataan PuTTY Key Generator –ohjelmaan *Load* –painikkeesta.(Kuvio 24.)



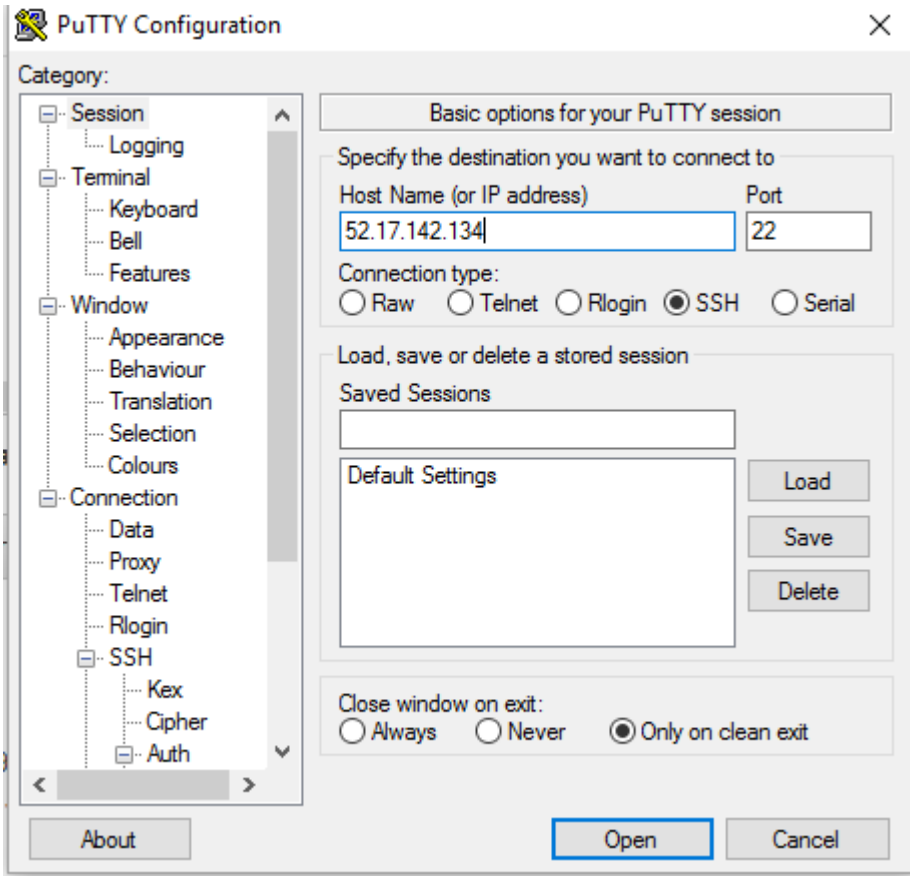

Kuvio 24. Avaimen lataaminen PuTTY Key Generatoriin

Yksityinen avain tallennetaan PuTTY –ohjelman tukemaan formaattiin valitsemalla *Save Private Key*.(Kuvio 25.)



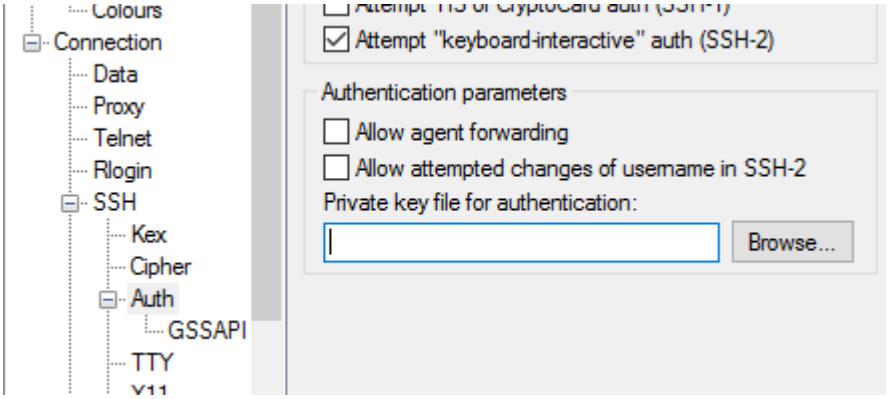

Kuvio 25. Avaimen tallentaminen

Yhteys avataan PuTTY –ohjelman kautta, jossa *Session* –näkymässä *Host Name* –kenttään kirjoitetaan palvelimen eth1 –verkkorajapinnan elastinen IP-osoite.(Kuvio 26.)




Kuvio 26. Yhteyden avaaminen

Autentikoinnissa käytettävä avain valitaan *Connection>SSH>Auth* –näkymässä.(Kuvio 27.)




Kuvio 27. Käytettävän avaimen valinta

Avain löytyy nyt kansiosta, johon se PuTTY Key Generator –ohjelmalla tallennettiin *.ppk* –formaatissa.

Palvelin kysyy käytettävää käyttäjätunnusta, joka Amazon Linux –käyttöjärjestelmässä oletuksena on *ec2-user* .

## 7.2 PPtP toteutus

Point-to-Point –palvelun asentaminen aloitetaan asentamalla palvelimelle PPP –työkalut. PPP –työkalut sisältävät Point-to-Point yhteyden hallinnassa ja autentikoinnissa käytettävät kirjastot, ajurit ja moduulit. PPP –työkalut ladataan ja asennetaan komennolla:

***yum install –y ppp***

Tämän jälkeen ladataan PPtPd asennustiedostot wget-komennolla:

***wget http://poptop.sourceforge.net/yum/stable/rhel6/x86_64/pptpd-1.4.0-1.el6.x86_64.rpm***

palvelu asennetaan ladatuista asennustiedostoista komennolla:

***yum -y localinstall pptpd-1.4.0-1.el6.x86_64.rpm***

komennossa käytetään –y –asetusta, jolla vastataan myöntävästi jokaiseen asennuksessa käyttäjältä kysyttävään vahvistukseen. Tämä tehdään myöhemmin kappaleessa ”Automatisointi” tapahtuvan skriptauksen takia.

PPtPd –konfiguraatiotiedostojen polku on:

***/etc/pptpd.conf***

Konfiguraatiotiedosto sisältävät seuraavat muokattavat kohdat:

**remoteip** - osoitteet, jotka jaetaan asiakaslaitteille. Nämä osoitteet ovat privaattiosoitteita, joita käytetään vain asiakaslaitteen ja palvelimen välisessä liikenteessä.

**localip** – osoite, johon asiakaslaitteet yhdistävät. Tähän lisätään palvelimen Eth1 –rajapinnan julkinen osoite, joka tässä tapauksessa on Amazon EC2 –elastinen IP-osoite.

Muita kenttiä ei tästä tiedostosta muokata. Kuviossa 28. nähdään PPtPd –konfiguraatiotiedoston muokatut kentät.

```
#          IP for each simultaneous client.
#
# (Recommended)
localip 52.49.111.140
remoteip 192.168.0.2-254
# or
#localip 192.168.0.234-238,192.168.0.245
#remoteip 192.168.1.234-238,192.168.1.245
```

Kuvio 28. PPtPd konfiguraatiotiedosto

Seuraavaksi lisätään käyttäjätiedot autentikoinnissa käytettävään tiedostoon, jonka polku on:

***/etc/ppp/chap-secrets***

Chap-secrets –tiedosto on esitetty kuviossa 29.

```
  GNU nano 2.3.1             File: chap-secrets

# Secrets for authentication using CHAP
# client        server  secret                  IP addresses
noise pptpd noise *
```

Kuvio 29. chap-secrets –tiedosto

Tiedostoon kirjataan käyttäjätiedot seuraavassa järjestyksessä

***käyttäjänimi>palvelin>salasana>IP osoitteet***

Erottimena käytetään välilyöntiä. Käyttäjänimeä ja salasanaa käytetään yhteyden muodostamisessa, nämä tiedot voidaan siis mielivaltaisesti päättää. Palvelimena toimii aiemmin asennettu PPtPd ja kaikki IP-osoitteet sallitaan, joten viimeiseen kenttään merkitään tähti.

Asiakaslaitteet saavat IP-osoitteet PPtPd –palvelimelta, mikäli laitteilla on DHCP eli Dynamic Host Configuration Protocol käytössä. Tässä yhteydessä laitteille jaetaan myös DNS

eli Domain Name Service –palvelinten osoitteet. Jaettavat DNS –palvelinten osoitteet määritetään PPP –asetustiedostoon, joka sijaitsee polussa

***/etc/ppp/options.pptpd***

Tässä toteutuksessa käytetään Google –palveluntarjoajan DNS –palvelimia. Näiden palvelinten osoitteet ovat *8.8.8.8 ja 8.8.4.4* . Kuviossa 30. on esitetty options.pptpd –tiedosto.

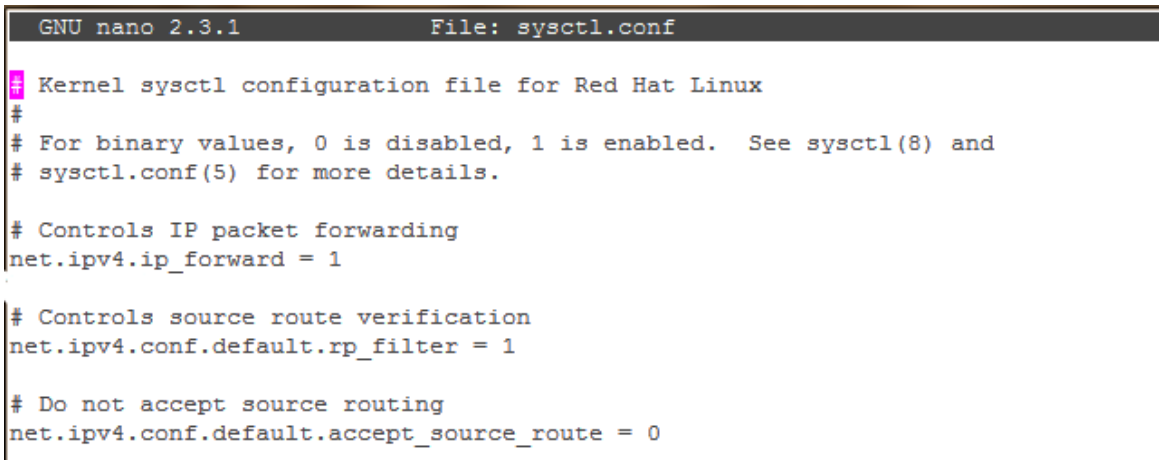
```
  GNU nano 2.3.1              File: options.pptpd

# Require MPPE encryption
# (note that MPPE requires the use of MSCHAP-V2 during authentication)
#mppe-40        # enable either 40-bit or 128-bit, not both
#mppe-128
#mppe-stateless
# }}}


# Network and Routing

# If pppd is acting as a server for Microsoft Windows clients, this
# option allows pppd to supply one or two DNS (Domain Name Server)
# addresses to the clients.  The first instance of this option
# specifies the primary DNS address; the second instance (if given)
# specifies the secondary DNS address.
ms-dns 8.8.8.8
ms-dns 8.8.4.4
```

Kuvio 30. options.pptpd –tiedosto

PPtP -yhteys käyttää Linux –verkkomallin Forward –ketjua. Jotta käyttöjärjestelmäydin hyväksyy muiden, kuin oman IP-osoitteen sisältävien pakettien lähetyksen, täytyy IP-forwarding ominaisuus hyväksyä tiedostossa:

***/etc/sysctl.conf***

Konfiguraatiotiedoston kohta *”net.ipv4.*ip_forward” vaihdetaan oletusarvosta arvoon *”1”* . Kuviossa 31. sysctl.conf –tiedosto.




Kuvio 31. sysctl.conf –tiedosto

Tämän jälkeen asetustiedosto ladataan uudestaan järjestelmän käyttöön komennolla:

***/sbin/sysctl –p***

Koska liikenne halutaan kierrättää välityspalvelimen läpi, tulee lähtevän liikenteen osoitteet muuttaa. Osoitteenmuutos tehdään iptables –työkalun nat –taulun avulla. Nat –tauluun lisätään sääntö, joka muuttaa POSTROUTING –ketjussa kaikki osoitteet rajapinnan osoitteeksi. Tämä sääntö otetaan käyttöön eth0 –rajapinnassa. Sääntö luodaan komennolla:

***iptables –t nat –A POSTROUTING –o eth0 –j MASQUERADE***

Komennon –t -asetus määrää kohdetaulun, joka tässä tapauksessa on “nat”, -A määrittää ketjun, jossa sääntöä sovelletaan. Puolestaan -j –asetus määrittää osoitteen, johon osoitteet muunnetaan, tässä käytetään MASQUERADE –asetusta, jolloin kaikki osoitteet muunnetaan rajapinnassa käytettäväksi IP-osoitteeksi. Lopuksi säännöt tallennetaan ja käynnistetään uudelleen komennoin:

***Service iptables save | Service iptables restart***

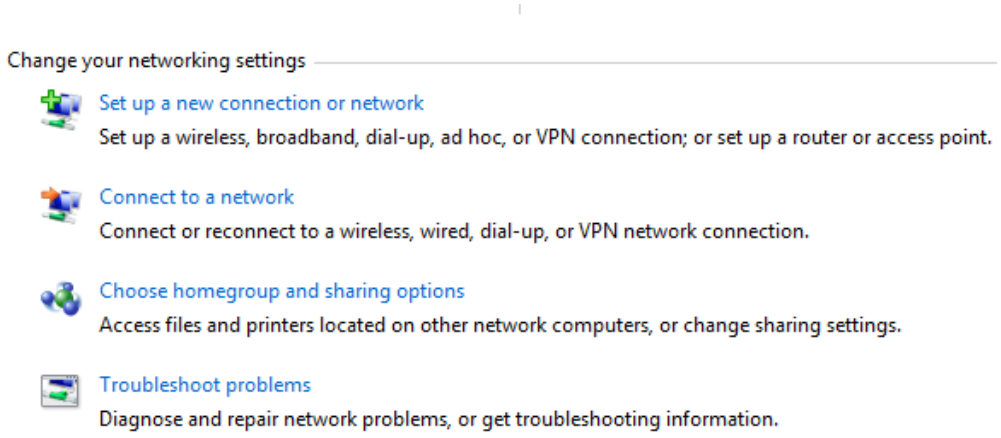
## 7.3 Asiakasyhteyden toteutus

Point-to-Point protokollan ansiosta hyvin monesta verkkolaitteesta löytyy asiakasohjelma, jolla luodaan VPN –yhteys palvelimeen.

Windows –käyttöjärjestelmissä yhteyden luominen tapahtuu Verkko ja Jakamiskeskuksen kautta. Vaiheet yhteyden luomiseen ovat seuraavat:

Avataan Verkko ja Jakamiskeskus Control Panel > All Control Panel Items > Network and Sharing Center.

Valitaan uuden yhteyden luominen ”Connect to a network” tästä alkaa ohjattu toiminto yhteyden avaamiseen(Kuvio 32.).




Kuvio 32. Yhteyden luominen

Palvelimen IP-osoitteena toimii aiemmin PPtPd –konfiguraatioon asetettu julkinen IP-osoite. Käyttäjätunnuksena ja salasanana käytetään Chap-secrets –tiedostoon asetettuja tunnuksia.

Mikäli Windows –käyttöjärjestelmässä on käytössä sekä IPv6 ja IPv4 –protokollat, reitityksen onnistumiseksi joissain tapauksissa joudutaan IPv6 -protokolla ottamaan pois käytöstä. Reittitaulu saadaan tulostettua *route print* –komennolla. Point-to-Point reitin

tulisi näkyä reittitaulussa pienimmällä *metric* –arvolla. Mikäli reitti ei saa pienintä *metric* –arvoa ja ei näin ollen ole valittu oletusreitiksi, tulee arvoa muuttaa komennolla:

***route change ”IP-osoite” mask ”aliverkon peite” “kohdeosoite” metric “uusi metric –arvo”***

Apple IOS –laitteissa asiakasohjelma kuuluu myöskin käyttöjärjestelmän oletustyökaluihin. Näissä järjestelmissä yhteyden muodostaminen tapahtuu Asetukset –valikosta polusta:

***Asetukset>Yleiset>VPN***

Yhteyden määrittämisessä yhteyden tyyppi tulee vaihtaa arvoon PPtP. (Kuvio 33.)


Kuvio 33. IOS -yhteyden muodostaminen

Linux -käyttöjärjestelmissä ei oletuksena ole asiakasohjelmaa PPtP –yhteyden muodostamiseksi. Asiakasohjelma asennetaan seuraavilla komennoilla distribuutiosta riippuen. Asentaminen vaatii internet –yhteyden.

Debian - ***apt-get install pptp-linux***

Fedora- ***yum install pptp***

Asiakasohjelma konfiguroidaan seuraavasti:

1. Lisätään tiedostoon */etc/ppp/options.pptp* rivi:

   ***lock noauth nobsdcomp nodeflate***

2. Lisätään käyttäjätiedot tiedostoon */etc/ppp/chap-secrets:*

   ***domain käyttäjänimi PPTP salasana ****

3. Luodaan tekstitiedosto */etc/ppp/peers/VPN* ,johon lisätään seuraavat rivit:

   ***pty "pptp IP-osoite --nolaunchpppd"***

   ***name käyttäjänimi***

   ***remotename PPTP***

   ***require-mppe-128***

   ***file /etc/ppp/options.pptp***

   ***ipparam VPN***

4. Yhteyden muodostaminen aloitetaan ja katkaistaan komennoilla:

   ***pppd call VPN***

Tämän jälkeen liikenne ohjataan PPP0 –tunneliin komennolla:

***route add default dev ppp0***

Joissain tapauksissa vanha oletusreitti joudutaan poistamaan komennolla:

***Route delete default dev ”interfacename”***

,jossa *“interfacename”* korvataan sen hetkisellä oletusrajapinnalla.

Tässä toteutuksessa ppp0 –rajapinnan MTU eli Maximum Transmission Unit täytyi asettaa oletusarvosta pienemmäksi komennolla:

***ifconfig ppp0 mtu 1400***

Tästä isommat kehykset aiheuttivat Point-To-Point tunnelin tukkeutumisen ja sen myötä yhteyden aikakatkaisun.

## 7.4 Muutokset käyttöliittymän lähdekoodiin

Käyttöliittymän lähdekoodia muokataan suunnitelman mukaisesti, jonka jälkeen se työnnetään uuteen Github –repositorioon.

Ensimmäisenä korjataan rajapinta, jolle rajoitukset asennetaan. Tämä muutos tapahtuu vaihtamalla API:n lähdekoodissa jokaiseen kohtaan, jossa arvoja asetetaan Loopback –rajapintaan, Eth0 –rajapinta. API:n lähdekoodi sijaitsee repositorion polussa:

***/server/src/api/index.js***

Muokattavia kohteita on seuraavilla riveillä: *9,55,71,78,* ja *84*.

Kuviossa 34. nähdään API:n lähdekoodissa esimerkkirivi, jota korjataan.

```
7
8       function setLimit(json){
9           var command = 'sudo tc qdisc change dev lo root netem ';
```

Kuvio 34. API -lähdekoodin muokattava rivi

Kuviossa 35. puolestaan nähdään samainen rivi muutosten jälkeen.

```
7
8       function setLimit(json){
9           var command = 'sudo tc qdisc change dev eth0 root netem ';
10
```

Kuvio 35. API -lähdekoodin rivi

Seuraavaksi poistetaan Contriboard –yksilöinti, tällä tarkoitetaan ainoastaan sitä, että käyttöliittymän Welcome –näkymästä muutetaan teksti *”Welcome to Contriboard-noise”* tekstiksi *”Welcome to Noise Generator”* . Welcome –näkymän lähdekoodi sijaitsee repositorion polussa:

***/server/src/views/welcome/index.js***

Target –toiminto saadaan poistettua poistamalla kansio */server/src/views/target* . Myös layout –tiedostosta tulee poistaa rivi, joka viittaa kyseiseen kansioon. Layout –tiedosto sijaitsee polussa:

***/server/src/views/layout.jade***

Tiedostosta poistetaan rivi 26, joka nähdään kuviossa 36.

```
25              li
26                a(href='/gui/target') Set targets
```

Kuvio 36. Layout –tiedosto

Rate –parametrin poistamiseksi tulee muokata API:n lähdekoodia ja Manual Config –näkymää. API:n lähdekoodista poistetaan rivit 50-53.(Kuvio 37.)

```
50          /*Do not allow negative rate values*/
51          if (json.rate != "" && json.rate >= 0) {
52              command += 'rate ' + json.rate + 'bit ';
53          }
```

Kuvio 37. Rate -parametrin poistaminen

Manual Config –näkymän lähdekoodi sijaitsee polussa:

***/server/src/views/config_manual/index.js***

Tästä tiedostosta poistetaan rivit 54-57. (Kuvio 38.)

```
54        div(class='form-group')
55          label(class='control-label col-sm-2') Rate (bit)
56          div(class='col-sm-10')
57            input(type='number' name='rate')
```

Kuvio 38. Manual Config -näkymän muokkaaminen

Kansiosta poistetaan ylimääräiset skriptit, Haproxy –kansio ja Docker –asetustiedostot. Lähdekoodia ei haluta päivittää entisen repositorion tilalle vaan sille luodaan uusi repositorio. Repositorion vaihto tapahtuu komennolla:

***git remote set-url https://github.com/mikkopoyhonen/noisetester***

Tämän jälkeen lähdekoodi hyväksytään ja työnnetään repositorioon komennoilla:

***git commit –a –m ”Changes to api,views,layout”***

***git push***

Tässä vaiheessa vaaditaan kirjautumista Github –käyttäjätunnuksella ja salasanalla.

Muokatut lähdekoodin osat ovat esitetty liitteissä 1-5.

## 7.5 Käyttöliittymän ja käyttäjärajapinnan toteutus

Käyttöliittymä, API ja osa niiden rakentamiseen ja käyttämiseen tarvittavista komponenteista ja työkaluista on jaettu julkisessa Github –repositoriossa. Repositorion kloonaamiseksi tulee ensimmäisenä Linux –palvelimelle asentaa Git –sovellus. Git –sovellus ladataan ja asennetaan komennolla:

***yum install git*** – Fedora

***apt-get install git*** - Debian

Java Script –käyttöliittymän rakentamiseksi tarvittavat työkalut ja ajurit asennetaan komennoilla:

***yum install Openssl-devel***

***yum install Gcc-C++ make***

Node.js –pakettien rakentamiseksi tarvitaan myös NPM –pakettienhallintatyökalut. NPM –työkalut ovat myöskin jaettu Github –palvelussa ja se saadaan kloonattua komennolla:

***git clone https://github.com/isaacs/npm.git***

Tällä komennolla repositorio kloonataan työkansioon. Tässä tapauksessa työkansion polku on */etc/ .*

Tämän jälkeen siirrytään kloonattuun kansioon, jossa NPM asennetaan make –työkalun avulla komennolla:

***make install***

Myös itse Node.js -sovelluskehys kloonataan julkisesta Github –repositoriosta komennolla:

***git clone https://github.com/nodejs/node.gi***

Seuraavaksi siirrytään kloonattuun Node –kansioon, jossa tarkistetaan, että Node.js –sovelluskehyksen versio on oikea komennolla:

***git checkout v0.12.2***

Tämän jälkeen asennus konfiguroidaan Node –kansiossa komennolla:

***./configure***

Jonka jälkeen Node asennetaan komennoilla:

***make***

***make install***

Kaikki työkalut ovat nyt asennettu, joten voidaan siirtyä rakentamaan itse käyttöliittymä ja ohjelmointirajapinta. Tiedostot ovat jaettu Github –palvelussa ja kloonaaminen tapahtuu:

***git clone https://github.com/mikkopoyhonen/noise-builder***

Kloonaamisen jälkeen siirrytään sijaintiin */etc/noisetester/server/*

Kansiossa suoritetaan komento:

***npm install***

Tämä komento asentaa *package.json* –tiedostossa määritetyt kirjastot ja työkalut, joita tarvitaan käyttöliittymän ajamisessa. Näihin kuuluvat seuraavat:

***"express": "4.12.4",***

***"body-parser": "1.12.4",***

***"shelljs": "0.5.1",***

***"jade": "1.10.0"***

Kun kirjastot ja työkalut ovat asennettu, voidaan palvelu käynnistää siirtymällä kansioon, jossa *index.js* –tiedosto sijaitsee. Tässä tapauksessa tiedoston polku on */etc/noisetester/server/src* . Tässä kansiossa suoritetaan komento:

***node index.js***

Tämä aloittaa verkkosivujen jakamisen portissa 80. Käyttäjälle annetaan kuvion 39. mukainen tuloste.

```
[ec2-user@ip-172-31-41-254 src]$ sudo node index.js
Running on port: 80
```

Kuvio 39. Tuloste palvelun käynnistämisestä

Itse Traffic Control –työkalu kuuluu oletuksena useimmissa Linux –distribuutioissa olevaan iproute2 –työkalupakettiin, joten sitä ei tarvitse erikseen asentaa.

## 7.6 Automatisoinnin toteutus

Linux shell –skripti voidaan luoda millä tahansa tekstieditorilla. Tiedoston nimeksi valitaan *startup.sh*. Ensimmäisenä skriptin tulee ladata ja asentaa tarvittavat ohjelmat ja työkalut palvelun rakentamiseksi. Tämä tapahtuu komennoilla:

***sudo yum install -y gcc-c++ make &***

***wait $!***

***echo Installed gcc-c++ and make***

***sleep 2***

***sudo yum install -y openssl-devel &***

***wait $!***

***echo Installed openssl-devel***

***sleep 3***

***sudo yum install -y ppp &***

***wget http://poptop.sourceforge.net/yum/stable/rhel6/x86_64/pptpd-1.4.0-1.el6.x86_64.rpm***

***sudo yum -y localinstall pptpd-1.4.0-1.el6.x86_64.rpm***

Komentojen välissä käytetään *wait $!* –komentoa, joka määrää, että prosessin on loputtava ennen uuden aloittamista. Myös *sleep* –komennolla voidaan viivästää seuraavan komennon suorittamista. Oletuksena kaikkia tarvittavia komentoja ei voida suorittaa *sudo* –komennon avulla, tämän takia *sudoers* –tiedostoon joudutaan lisäämään *Secure Path* –kohtaan polku */bin:/usr/local/bin/* . Tekstin korvaaminen tiedoston */etc/sudoers/* sisällä tapahtuu komennolla:

***sudo sed -i 's/secure_path = \/sbin:\/bin:\/usr\/sbin:\/usr\/bin/secure_path = \/sbin:\/bin:\/usr\/sbin:\/usr\/bin:\/usr\/local\/bin/g' /etc/sudoers/g***

Merkki \ sulkee pois function merkiltä /, jolloin se tulkitaan tekstinä.

Github –repositoriot kloonataan kuten manuaalisessakin toteutuksessa komennolla:

***sudo git clone https://github.com/isaacs/npm.git***

***sudo git clone https://github.com/nodejs/node.git***

***sudo git clone https://github.com/mikkopoyhonen/noisetester.git***

Tämän jälkeen skriptin täytyy suorittaa paketeille samat toiminnot, jotka niille suoritetaan manuaalisessa toteutuksessa. Skriptin sisällä täytyy myös työkansiota vaihtaa, kuten normaalistikin.

PPtPd –palvelun konfigurointia ei voida suorittaa täysin automaattisesti, koska IP-osoitteet saattavat vaihtua ympäristöstä riippuen. Konfiguroinnissa kysytään käyttäjää syöttämään käytetyt IP-osoitteet, jotka lisätään *pptpd.conf* ja *chap-secrets* –tiedostoihin viimeiselle riville. Syötteen pyytäminen käyttäjältä tapahtuu seuraavasti:

***read -p "Enter your IP address which you want to use for PPTP prefer elastic ip in amazon EC2 : " iplocal***

***echo "localip $iplocal" >> /etc/pptpd.conf***

***read -p "Enter ip address pool allocated to machines, for example 192.168.0.1-20 : " ipremote***

***echo "remoteip $ipremote" >> /etc/pptpd.conf***

***echo "Added $IPREMTOE as remote ip!"***

***read -p "Enter username for pptp : " uname***

***read -p "Enter your password for pptp : " pword***

***echo "$uname pptpd  $pword *" >> /etc/ppp/chap-secrets***

Myös *options.pptpd* –tiedostoon syötetään nimipalvelimen tiedot ja *sysctl* –tiedoston IP-forward –asetus otetaan käyttöön  sed –komennoilla seuraavasti:

***sudo sed -i 's/#ms-dns 10.0.0.1/ms-dns  8.8.8.8/g' /etc/ppp/options.pptpd***

***sudo sed -i 's/#ms-dns 10.0.0.2/ms-dns  8.8.4.4/g' /etc/ppp/options.pptpd***

***sudo sed -i 's/net.ipv4.ip_forward = 0/net.ipv4.ip_forward = 1/g' /etc/sys-ctl.conf***

Palvelun käynnistämisessä, qdisc –oletusarvojen asettamisessa ja IPtablesin konfiguroinnissa käytetään samoja komentoja, kuin manuaalisessa toteutuksessa.

Tekstitiedostosta luodaan suoritettava skriptitiedosto komennolla:

***sudo chmod +x startup.sh***

Skripti suoritetaan kansiossa komennolla:

***sudo ./startup.sh***

Joissain tapauksissa skripti joudutaan ajamaan kahdesti

Shell –skripti on esitetty kokonaisuudessaan liitteessä 7.

# 8 TOIMINNAN TODENNUS JA TESTAUS

## 8.1 Yhteys asiakaslaitteelta

Kun Point-to-Point –yhteys on luotu, tulisi uusi ppp0 –verkkorajapinta näkyä *ifconfig* –komennolla Linux käyttöjärjestelmissä. Windows –käyttöjärjestelmissä rajapinnat nähdään *ipconfig* –komennolla. Kuviossa 40. nähdään ifconfig –komennon tuloste.

```
ppp0      Link encap:Point-to-Point Protocol
          inet addr:192.168.0.1  P-t-P:172.31.199.17  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1400  Metric:1
          RX packets:84500 errors:0 dropped:0 overruns:0 frame:0
          TX packets:55045 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:108119486 (108.1 MB)  TX bytes:15709641 (15.7 MB)
```

Kuvio 40. Ifconfig –tuloste

Kun Point-To-Point –yhteys ja oletusreitti on saatu asetettua, voidaan testata traceroute –työkalulla kulkeeko liikenne asiakaslaitteelta testikohteeseen välityspalvelimen kautta. Kuviossa 41. traceroute –tuloste.

```
lubuntu@lubuntu:/etc/ppp/peers$ traceroute contriboard.n4sjamk.org
traceroute to contriboard.n4sjamk.org (54.194.192.73), 30 hops max, 60 byte packets
 1  172.31.199.17 (172.31.199.17)  96.565 ms  113.228 ms  113.249 ms
 2  ec2-79-125-0-240.eu-west-1.compute.amazonaws.com (79.125.0.240)  130.105 ms ec2-79-1
36.eu-west-1.compute.amazonaws.com (79.125.0.136)  113.369 ms
```

Kuvio 41. Traceroute –tuloste

## 8.2 Käyttöliittymä

Käyttöliittymän toimintaa testataan muuttamalla verkon rajoituksia ja samalla seuraamalla asiakaslaitteelta lähetettyjen ICMP echo –viestien viivettä. ICMP echo –viestejä lähetetään ping –komennolla. Ensin ping –komennon tulostetta seurataan ilman rajoituk-

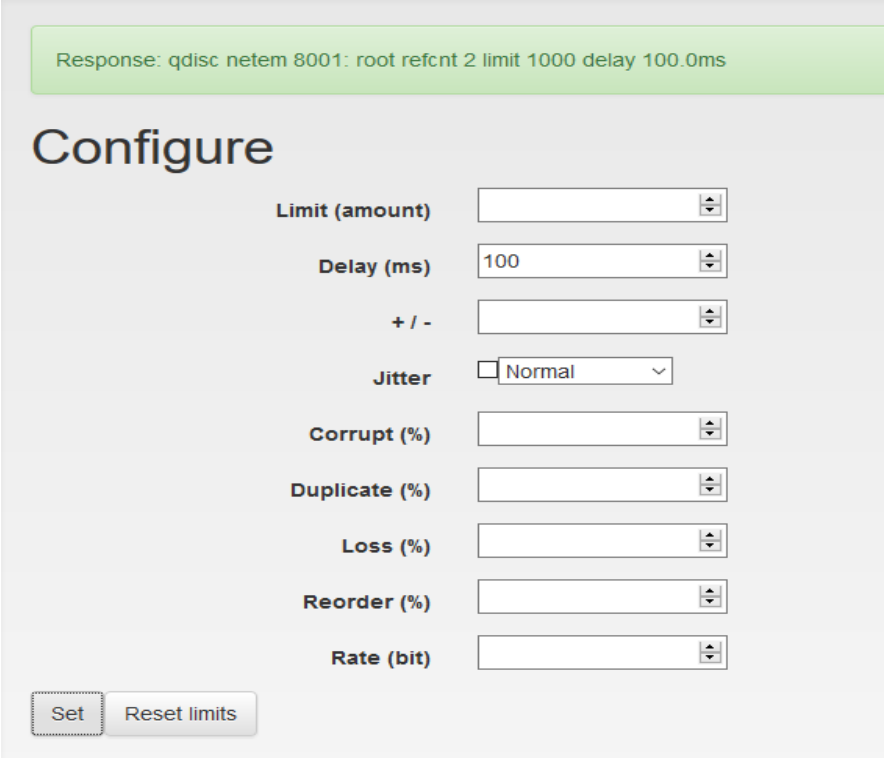
sia ja kesken komennon suorittamisen *Delay* –arvoksi muutetaan 100 millisekuntia. Kuvissa 42. nähdään ping –komennon tuloste, kun *Delay* –arvoa muutetaan kesken komennon suorittamisen.

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=58 time=101 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=58 time=95.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=58 time=95.1 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=58 time=94.4 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=58 time=94.5 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=58 time=96.3 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=58 time=194 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=58 time=233 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=58 time=241 ms
```

Kuvio 42. Ping -komennon tuloste

Huomataan, että *Delay* –arvon muuttaminen nosti myös ICMP echo –viestien viivettä seitsemännen viestin kohdalla.

Muut ominaisuudet testataan ottamalla ne käyttöön ja seuraamalla yläreunan *Response* –tulostetta.(Kuvio 43.)




Kuvio 43. Arvojen muuttaminen käyttöliittymässä

## 8.3 Välityspalvelimen suorituskykytestaus

Välityspalvelimen vaikutusta verkon nopeuteen testataan speedtest-cli –ohjelmalla. Speedtest –ohjelma asennetaan komentokehotteen kautta Linux –asiakaslaitteelle. Asentaminen tapahtuu komennoilla:

***wget https://raw.github.com/sivel/speedtest-cli/master/speedtest_cli.py***

***chmod a+rx speedtest_cli.py***

***mv speedtest_cli.py /usr/local/bin/speedtest-cli***

***chown root:root /usr/local/bin/speedtest-cli***

Testi käynnistetään komennolla

***speedtest-cli***

Testin tulokset nähdään kuviossa 44.

```
lubuntu@lubuntu:/etc/locust-contriboard$ speedtest-cli
Retrieving speedtest.net configuration...
Retrieving speedtest.net server list...
Testing from Amazon Technologies (52.50.160.104)...
Selecting best server based on latency...
Hosted by Three Ireland (Dublin) [1.78 km]: 106.586 ms
Testing download speed.....................................
Download: 21.91 Mbit/s
Testing upload speed.....................................
Upload: 3.85 Mbit/s
```

Kuvio 44. Speedtest –tulokset

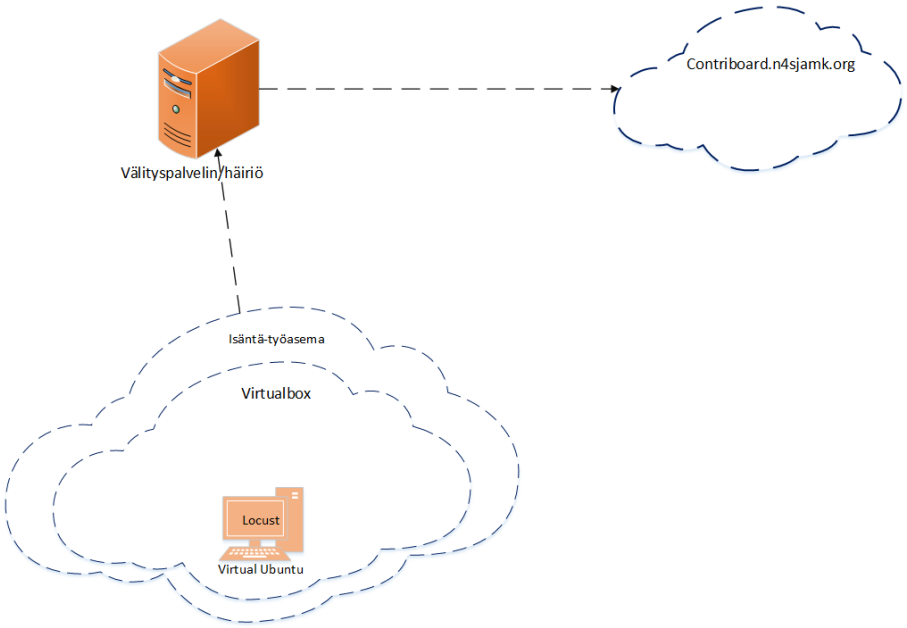
Tässä tapauksessa asiakasyhteys tapahtui 4G –verkon yli. Latausnopeudeksi saatiin 21.91 megabittiä sekunnissa ja lähetysnopeudeksi 3.85 megabittiä sekunnissa. Viive asiakaslaitteen ja testikohteen välillä mittaushetkellä on 106 millisekuntia. Viivettä tässä toteutuksessa aiheuttaa fyysinen etäisyys välityspalvelimeen.

Tuloksien perusteella todetaan, että verkon nopeus on riittävä ja viive siedettävä toiminnallisten –ja suorituskykytestejen suorittamiseksi. Suorituskykyä kuitenkin on mahdollista parantaa asentamalla palvelu esimerkiksi omaan lähiverkkoon virtuaaliselle tai fyysiselle palvelimelle.

## 8.4 Locust testaus

Tuotteen toiminnan testausta todennetaan suorittamalla suorituskykytestejä tuotteen läpi Contriboard –palveluun. Testi suoritetaan myös ilman generoitua häiriötä, jotta testaustuloksia voidaan verrata. Tämä testaustapa on mustalaatikko –orientoitunut, koska palvelulle lähetetään pyyntöjä ja seurataan sen vastauksia ottamatta kantaa palvelun sisällä tapahtuviin prosesseihin.

Suorituskykytestaukseen käytetään Locust –ohjelmaa. Locust ohjelmaan on olemassa valmis Contriboard –testiskenaario Github –palvelussa. Kuviossa 45. nähdään suorituskykytestauksen topologia.




Kuvio 45. Suorituskykytestauksen topologia

Locust vaatii järjestelmältä seuraavat työkalut:

***Python***

***gcc-c++ make***

Locust asennetaan komennolla:

***pip install locustio***

tai

***easy_install locustio***

Varsinainen Locust –testiskenaario kloonataan N4SJAMK Github –repositoriosta komennolla:

***git clone https://github.com/N4SJAMK/locust-contriboard***

Tämän jälkeen kansiolle ja sen sisällölle tulee antaa täydet käyttöoikeudet kaikille käyttäjille, koska Locust –komento ei hyväksy *Sudo* –asetusta .

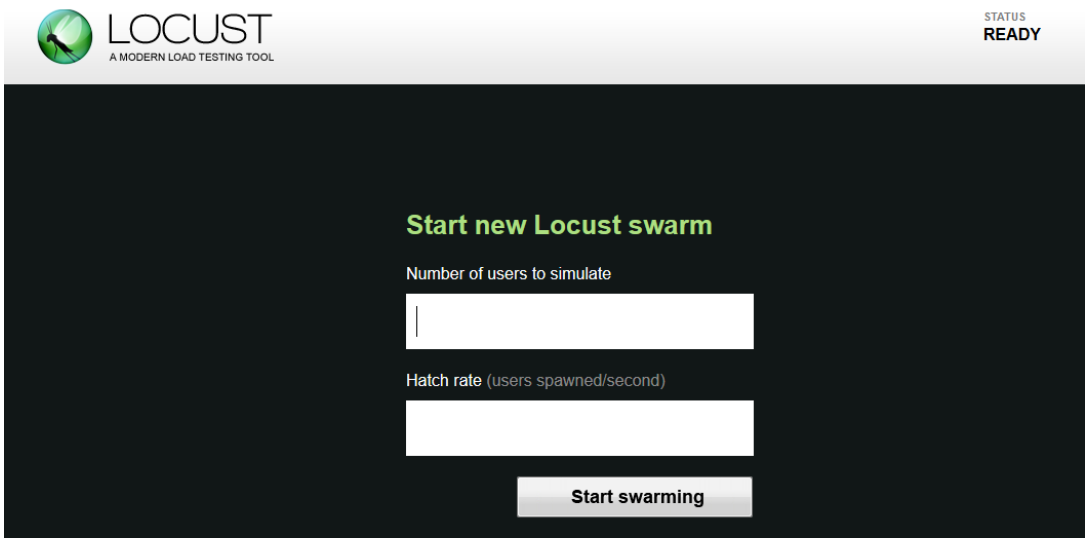
Oikeudet määritetään komennolla:

***chmod 777 /etc/locust-contriboard***

Varsinainen prosessi käynnistetään kansiossa *locust-contriboard* komennolla

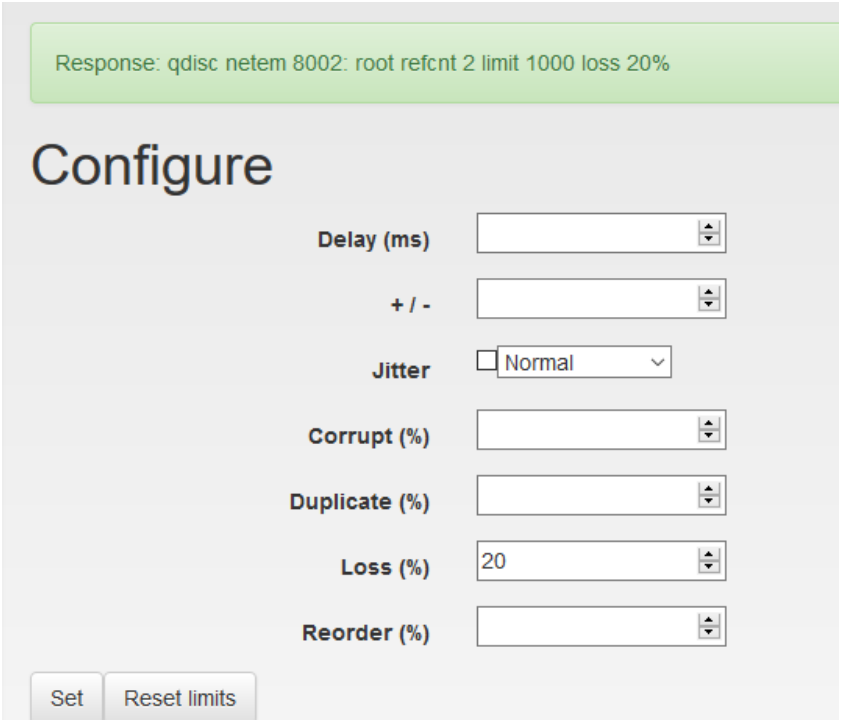
***locust -f locust-tests-new.py -H http://contriboard.n4sjamk.org/ TeamboardUser***

Tämän jälkeen testit suoritetaan ja monitoroidaan verkkoselaimella palvelimen tai työaseman osoitteesta portista *8089.* Kuviossa 46. nähdään Locust –käyttöliittymä.




Kuvio 46. Locust –käyttöliittymä

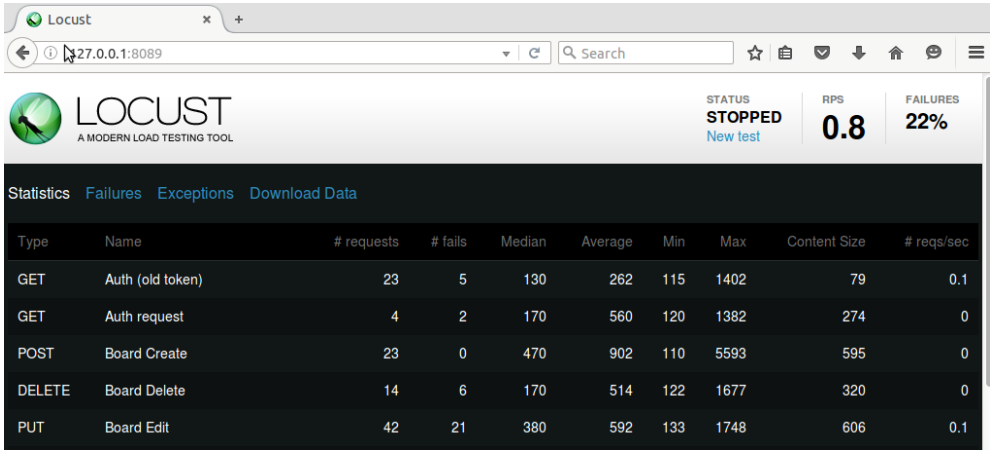
Tässä testissä käyttäjien määräksi asetetaan viisi kappaletta ja *”Hatch rate”* asetetaan yhteen. Tässä skenaariossa häiriötä simuloidaan ainoastaan pakettien pudottamisella. Kuviossa 47. nähdään asetetut arvot.




Kuvio 47. Testauksessa käytetyt arvot

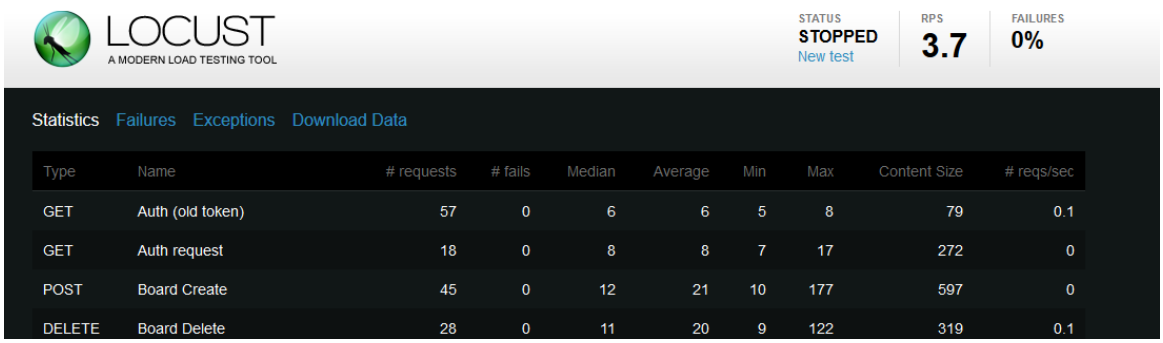
## 8.5 Tulokset

Testaustuloksia voidaan analysoida nopeasti graafisesta käyttöliittymästä, joka esittää palvelimelle lähetetyt pyynnöt, pyyntöjen määrän sekuntia kohti ja virheiden määrän prosentteina. Kuviossa 48. nähdään ensimmäisen testin tulokset, jossa käytettiin *20% Packet Loss* –asetusta.




Kuvio 48. Testaustulokset 20% Packet loss

Vastaavasti kuviossa 49. nähdään testitulokset testistä, jossa ei käytetty häiriötä.




Kuvio 49. Testitulokset ilman häiriötä

Locust –ohjelma antaa tulosteen myös komentokehotteeseen palvelimelle. Komentokehotteen raportit ovat esitetty liitteessä 5.

Testitulosten virheraportti on esitetty kuviossa 50.

```
Error report
 # occurences       Error
-------------------------------------------------------------------------------------------------------------------
---------------
 4                  DELETE Board Delete: "CatchResponseError('Board remove failed, code: 0',)"
 1                  POST Board Share: "CatchResponseError('Share board failed, code: 0',)"
 1                  GET Board export as image: "CatchResponseError('Export failed, code 0',)"
 16                 POST Ticket Create: "CatchResponseError('Creating new ticket failed, code: 404',)"
 1                  GET Board export as plaintext: "CatchResponseError('Export failed, code 404',)"
 2                  DELETE Board Delete: "CatchResponseError('Board remove failed, code: 404',)"
 3                  GET Version Api: "CatchResponseError('Version api failed, code: 0',)"
 26                 PUT Ticket Move: "CatchResponseError('Failed to move ticket, code: 404',)"
 1                  GET Board export as csv: "CatchResponseError('Export failed, code 0',)"
 1                  GET Auth (old token): "CatchResponseError('Get Auth after logout != 401, code: 200',)"
 4                  GET Auth (old token): "CatchResponseError('Get Auth after logout != 401, code: 0',)"
 5                  GET Board Get Single: "CatchResponseError('Getting board by id failed, code 404',)"
 10                 GET Board Get Tickets: "CatchResponseError('Getting board tickets failed, code 404',)"
 1                  GET Board Get Events: "CatchResponseError('Getting board events failed, code 0',)"
 2                  DELETE Ticket Delete: "CatchResponseError('Ticket deletion failed, code: 404',)"
 5                  GET Board Get All: "CatchResponseError('Getting boards failed, code 0',)"
 6                  DELETE Ticket Delete: "CatchResponseError('Ticket deletion failed, code: 0',)"
 7                  PUT Board Edit: "CatchResponseError('Board edit failed, code: 404',)"
 1                  GET Version Img: "CatchResponseError('Version img failed, code: 0',)"
 5                  GET Board Get Single: "CatchResponseError('Getting board by id failed, code 0',)"
 14                 PUT Board Edit: "CatchResponseError('Board edit failed, code: 0',)"
 12                 POST Ticket Comment: "CatchResponseError('Ticket comment failed, code: 0',)"
 15                 POST Ticket Comment: "CatchResponseError('Ticket comment failed, code: 404',)"
 5                  PUT Ticket Move: "CatchResponseError('Failed to move ticket, code: 0',)"
 2                  GET Board Get Tickets: "CatchResponseError('Getting board tickets failed, code 0',)"
 2                  GET Auth request: "CatchResponseError('Get Auth failed, code: 0',)"
 3                  POST Logout: "CatchResponseError('Logout failed, code: 0',)"
 18                 POST Ticket Create: "CatchResponseError('Creating new ticket failed, code: 0',)"
 1                  PUT Ticket Modify Content: "CatchResponseError('Ticket modify failed, code: 0',)"
 1                  GET Board export as image: "CatchResponseError('Export failed, code 404',)"
 23                 PUT Ticket Modify Content: "CatchResponseError('Ticket modify failed, code: 404',)"
```

Kuvio 50. Testitulosten virheraportti

Virheraportista voidaan lukea yleisimmät huonon verkkoyhteyden aiheuttamat virheet. Näitä ovat Contriboardin osalta seuraavat:

**tiketin siirtäminen**

**tiketin muokkaaminen**

**tiketin luominen**

Samaan aikaan häiriöttömällä yhteydellä toteutetussa testissä ei ilmennyt virheitä, vaikka palvelu joutui käsittelemään samaan aikaan häiriöllistä liikennettä, joka vastaa kymmenen käyttäjän lähettämää liikennemäärää. Näin ollen voidaan todeta, että yksittäiset huonoilla verkkoyhteyksillä palvelua käyttävät käyttäjät eivät vaikuta muihin käyttäjiin ruuhkauttamalla palvelun verkkorajapintaa.

# 9 POHDINTA

## 9.1 Työn lopputulos

Työn lopputuloksena on palvelu, jonka avulla voidaan simuloida huonoja verkko-olosuhteita käyttäjämäärästä ja testauskohteesta riippumatta. Välityspalvelimen käyttö onnistuu helposti yksinkertaisen web-käyttöliittymän kautta, mutta arvot voidaan asettaa myös JSON –formaatissa esimerkiksi testausautomaatio –työkalujen kautta.

Palvelua voidaan isännöidä niin fyysisillä, kuin virtuaalisillakin Linux –palvelimilla. Palvelua voidaan käyttää myös monilla eri asiakaslaitteilla, kuten:

***-IOS matkapuhelimet***

***-Linux –käyttöjärjestelmät***

***-Windows –käyttöjärjestelmät***

Palvelun käyttöönotto onnistuu myöskin kätevästi Linux shell –skriptin avulla, mutta asiakaslaitteiden konfigurointia ja ympäristön pystytystä ei ole automatisoitu.

Työssä toteutetut suorituskykytestit toteutettiin lähinnä havainnollistamaan palvelun toimintaa ja toteamaan, että palvelu toimii suunnitelman mukaisesti. Locust –ohjelman antamista raporteista voidaan lukea suoraan yleisimmät häiriöiden aiheuttamat virheet, mutta niiden ilmenemistä ei päästä näkemään käytännössä. Tämän takia palvelu soveltuu paremmin toiminnallisiin testeihin, joissa käyttäjä testaa manuaalisesti tai monitoroi testaus automaatio -työkalun suorittamaa testiä ja tekee toiminnan perusteella havaintoja ohjelman toiminnasta huonoilla verkko-olosuhteilla.

## 9.2 Hyöty

Työn varsinainen hyöty saadaan itse tuotteesta. Kuka tahansa voi pystyttää oman välityspalvelimen nopeasti ja ottaa sen osaksi omaa testausketjua. Koska palvelu käyttää

Point-To-Point –tunnelia, voidaan palvelua käyttää kaikissa mahdollisissa testauskohteissa, kuten verkkopalvelut, verkkopelit ja erilaiset video ja ääni –palvelut.

Simuloimalla huonoja verkko-olosuhteita palvelun testausvaiheessa, voidaan palvelun toimivuutta parantaa juurikin huonoilla verkko-olosuhteilla palvelua käyttäville käyttäjille. Joissakin tapauksissa voidaan myös löytää verkko-olosuhteiden aiheuttamia kriittisiä vikoja ja puutteita ohjelmasta tai ympäristöstä.

## 9.3 Tuotteen jatkokehitys

Vaikka tuotteen toteutus onnistui aikataulun ja laajuuden puitteissa täydellisesti, on aina tarvetta jatkokehitykselle ja tuotteen jalostamiselle. Muun muassa palvelun käynnistämistä olisi mahdollista helpottaa entisestään esimerkiksi Ansible –työkalun avulla. Ansiblen avulla myös Amazon –ympäristön rakentaminen ja asiakaslaitteiden konfigurointi voitaisiin automatisoida täysin.

Tuotteen käyttöä varten olisi myös hyvä alkaa kehittämään testiskenaarioita, jossa palvelua käytetään automatisoidusti testiautomaatiotyökalujen kautta.

Mikäli palvelun käyttäjät kokevat tietoturvatason liian alhaiseksi, on mahdollista Point-to-Point –protokollan sijasta ottaa käyttöön L2TP ja IPsec –protokollat. L2TP/IPsec on myös laajalti tuettu useissa käyttöjärjestelmissä kuten:

***Windows***

***Unix –käyttöjärjestelmät***

***IOS***

***Android***

Myöskään suuremman salauksen aiheuttama prosessorikuorma ei nykypäivän laitteilla aiheuta suuria muutoksia palvelun suorituskyvyssä.

Palvelu olisi mahdollista rakentaa myös yhdellä verkkorajapinnalla. Yhden verkkorajapinnan toteutuksessa palvelimelle saapuva liikenne ohjattaisiin esimerkiksi HAProxy –ohjelmalla virtuaaliseen Loopback –rajapintaan, jossa suoritetaan liikenteelle halutut rajoitukset. HAProxyn avulla hallintayhteys voidaan erotella rajoitettavasta liikenteestä, joten hallintayhteys ei katkea vaikka liikennettä testauskohteen ja asiakaslaitteen välillä rajoitettaisiin.

# LÄHTEET

Alani, M. 2014. Guide to OSI and TCP/IP Models.

Amazon Web Services, 2016. Palvelun verkkosivut Viitattu 20.2.2016 https://aws.amazon.com/ec2/details/.

Deal, R. 2006. The Complete Cisco VPN Configuration Guide.

Digital Ocean 2016. Palvelun verkkosivut. Viitattu 11.3.2016. https://www.digitalocean.com/help/.

Fewster, M. & Graham, D. 1999. Software test automation, effective use of test execution tools.

Fewster, M. & Graham, D. 1999. Software test automation, effective use of test execution tools.

Furht, B. & Escalante, A. 2011. Handbook of cloud computing.

Gajda, W. 2013. Git Recipes: A ProblemSolution Approach.

Github. 2015. Palvelun verkkosivut. Viitattu 8.2.2016. https://github.com/.

Google. 2016. Palvelun AppEngine dokumentaatio. Viitattu 12.4.2016. https://cloud.google.com/appengine/.

ISTQB. 2016. Viitattu 8.2.2016. Organisaation verkkosivujen dokumentaatio. http://istqbexamcertification.com/what-is-integration-testing/.

Jäsberg, J. 2011. Web-sovellusten testauksen automatisointi, opinnäytetyö. Jyväskylän Ammattikorkeakoulu, tekniikan ja liikenteen ala.

Kavis, M. 2014. Architecting the Cloud: Design Decisions for Cloud Computing Service Models.

Koirala, S. & Sheikh, S. 2012. Software Testing: Interview Questions.

Linux Foundation. 2009. Työkalun dokumentaatio. Viitattu 12.4.2016. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

Linux. 2016. Linux-verkkosivun wiki-osio. Viitattu 13.2.2016 https://www.linux.fi/wiki/Iptables.

Microsoft. 2016. Palvelun Azure dokumentaatio. Viitattu 12.4.2016. https://azure.microsoft.com/en-gb/overview/what-is-azure/.

Myers, G. j., Sandler, C. & Badgett, T. 2012. Art of Software testing, third edition.

N4S-ohjelma. 2016. N4S esittely yrityksen Digile verkkosivuilla. Viitattu 13.4.2016. http://www.n4s.fi/fi/.

Pekkinen, J. 2011. Yksikkötestauksen mahdollisuus Finncomm Oy:ssä. Opinnäytetyö, Seinäjoen ammattikorkeakoulu, tekniikan ja liikenteen ala.

Rhodes, B. & Goerzen, J. 2010. Foundations of Python Network Programming: The Comprehensive Guide to Building Network Applications with Python, Second Edition.

Shotts, W. E. JR. 2000-2016. Linux-ohjesivusto. Viitattu 8.2.2016. http://linuxcommand.org/wss0010.php.

What is an API? 2012. Dokumentaatio yrityksen 3scale sivustoilla. Viitattu 23.2.2016. http://www.3scale.net/wp-content/uploads/2012/06/What-is-an-API-1.0.pdf.

Wiley, J. & Sons, 2011. Introduction to Cloud Computing.

# LIITTEET

## Liite 1. API –lähdekoodi

```
module.exports = function(app){
var shell = require('shelljs');

// Timers
var timeouts = [];
var loopouts = [];

function setLimit(json){
var command = 'sudo tc qdisc change dev eth0 root netem ';

/*Do not allow negative delay values*/
if (json.limit != "" && json.limit >= 0) {
command += 'limit ' + json.limit + ' ';
}

/*Do not allow negative delay values*/
if (json.delay != "" && json.delay >= 0) {
command += 'delay ' + json.delay + 'ms ';
}

/*Don't execute unless the user has inputted a value for delay*/
if (json.delayvariance && json.delay != "") {
command += json.delayvariance + 'ms ';
}

if(parseInt(json.jitter) === 1){
command += 'distribution ' + json.distribution + ' ';
}

/*Do not allow loss-values over 100% or under 0%*/
if (json.loss != "" && json.loss >= 0 && json.loss <= 100) {
command += 'loss ' + json.loss + '% ';
```

```
}
/*Do not allow duplicate-values over 100% or under 0%*/
if (json.duplicate != "" && json.duplicate >= 0 && json.duplicate <= 100) {
command += 'duplicate ' + json.duplicate + '% ';
}

/*Do not allow reorder-values over 100% or under 0%*/
if (json.reorder != "" && json.reorder >= 0 && json.reorder <= 100) {
command += 'reorder ' + json.reorder + '% ';
}

/*Do not allow corrupt-values over 100% or under 0%*/
if (json.corrupt != "" && json.corrupt >= 0 && json.corrupt <= 100) {
command += 'corrupt ' + json.corrupt + '% ';
}

if (command != 'sudo tc qdisc change dev eth0 root netem ') {
shell.exec(command, {silent:true});
}
}

function resetLimit() {
for (var i = 0; i < timeouts.length; i++) {
clearTimeout(timeouts[i]);
}

for (var z = 0; z < loopouts.length; z++) {
clearInterval(loopouts[z]);
}

timeouts = [];
loopouts = [];
var command = 'sudo tc qdisc change dev eth0 root netem limit 1000 delay 0ms 0ms cor-
rupt 0% duplicate 0% reorder 0% loss 0% rate 0bit';
shell.exec(command, {silent:true});
}

app.put('/limit', function (req, res) {
resetLimit();
```

```
setLimit(req.body);
var status = shell.exec('sudo tc qdisc show dev eth0', {silent:true}).output;
res.json(status);
});

app.put('/limit/reset', function(req, res) {
resetLimit();
var status = shell.exec('sudo tc qdisc show dev eth0', {silent:true}).output;
res.json(status);
});

app.put('/target', function (req, res) {
var command = undefined;

command = "line=\"$(grep -n '\\sbackend contriboardclient' /etc/haproxy/haproxy.cfg | cut -d : -f 1)\" ; line=$((line+1)) ; sed -i \"${line}s/.*/server target "+req.body.clien-turl+"/\" /etc/haproxy/haproxy.cfg";
shell.exec(command, {silent:true});

command = "line=\"$(grep -n '\\sbackend contriboardapi' /etc/haproxy/haproxy.cfg | cut -d : -f 1)\" ; line=$((line+1)) ; sed -i \"${line}s/.*/server target "+req.body.apiurl+"/\" /etc/haproxy/haproxy.cfg";
shell.exec(command, {silent:true});

command = "line=\"$(grep -n '\\sbackend contriboardio' /etc/haproxy/haproxy.cfg | cut -d : -f 1)\" ; line=$((line+1)) ; sed -i \"${line}s/.*/server target "+req.body.iourl+"/\" /etc/haproxy/haproxy.cfg";
shell.exec(command, {silent:true});

var status = shell.exec('sudo service haproxy restart', {silent:true}).output;
res.json(status);
});

app.put('/script', function(req, res){
resetLimit();
var loop = req.body.loop;
var jsonscript = JSON.parse(req.body.jsonscript);
var seconds = Object.keys(jsonscript);
console.log('New script with timers at: ' +seconds.toString());
seconds.forEach(function(i){
timeouts.push(setTimeout(function(){
```

```
console.log('');
console.log('[S]['+i+' sec] Delayed trigger fired');
setLimit(jsonscript[i]);
if(loop){
loopouts.push(setInterval(function(){
console.log('');
console.log('[L]['+i+' sec] Delayed trigger fired');
setLimit(jsonscript[i]);
},1000*Math.max.apply(Math,seconds)));
}
},i*1000));
});
res.json('New script with timers at: ' + seconds.toString());
});
}
```

## Liite 2. Welcome -näkymän –lähdekoodi

```
extends ../layout
block content
h1 Welcome
div
p This is Noise Generator
```

## Liite 3. layout_jade –lähdekoodi

```
doctype html
html
head
title Noise API
script(src='https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js')
link(rel='stylesheet' href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/boot-
strap.min.css')
link(rel='stylesheet' href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/boot-
strap-theme.min.css')
link(rel='stylesheet' href='/static/stylesheets/style.css')
```

```
script(src='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/js/bootstrap.min.js')
script(src='/static/js/utility.js')

body
//#{authe = state.authenticated}
div(class='wrapper')
nav(class='navbar navbar-inverse navbar-static-top')
div(class='container')
a(class='navbar-brand' href='/') Noise Manager
ul(class='nav navbar-nav')
li
a(href='/') Home
li
a(href='/gui/config/manual') Manual Config
li
a(href='/gui/config/script') Scripted config
div(class='well')
div(role='alert' id='status' style='display: none')
block content
```

## Liite 4. config_manual –lähdekoodi

```
extends ../layout

block content

h1 Configure

div
form(role='form' method='PUT' class='form-horizontal' id="configureform")
div(class='form-group')
label(class='control-label col-sm-2') Limit (amount)
div(class='col-sm-10')
input(type='number' name='limit')

div(class='form-group')
label(class='control-label col-sm-2') Delay (ms)
```

```
div(class='col-sm-10')
input(type='number' name='delay')

div(class='form-group')
label(class='control-label col-sm-2') + / -
div(class='col-sm-10')
input(type='number' name='delayvariance')

div(class='form-group')
label(class='control-label col-sm-2') Jitter
div(class='col-sm-10')
input(type='checkbox' name='jitter' value='1')

select(name='distribution')
option(value='normal') Normal
option(value='pareto') Pareto
option(value='paretonormal') Paretonormal

div(class='form-group')
label(class='control-label col-sm-2') Corrupt (%)
div(class='col-sm-10')
input(type='number' name='corrupt')

div(class='form-group')
label(class='control-label col-sm-2') Duplicate (%)
div(class='col-sm-10')
input(type='number' name='duplicate')

div(class='form-group')
label(class='control-label col-sm-2') Loss (%)
div(class='col-sm-10')
input(type='number' name='loss')

div(class='form-group')
label(class='control-label col-sm-2') Reorder (%)
div(class='col-sm-10')
input(type='number' name='reorder')

div(class='form-group')
```

```
div(class='col-sm-10')
input(class='btn btn-default' type='button' onclick='ajaxRequest("PUT", "/limit", "con-
figureform")' value='Set')
input(class='btn btn-default' type='button' onclick='ajaxRequest("PUT", "/limit/reset",
"configureform")' value='Reset limits')
```

# Liite 5. Locust –testitulokset

```
Error report
 # occurences      Error
-------------------------------------------------------------------------------------------------------------
---------------
 4                 DELETE Board Delete: "CatchResponseError('Board remove failed, code: 0',)"
 1                 POST Board Share: "CatchResponseError('Share board failed, code: 0',)"
 1                 GET Board export as image: "CatchResponseError('Export failed, code 0',)"
 16                POST Ticket Create: "CatchResponseError('Creating new ticket failed, code: 404',)"
 1                 GET Board export as plaintext: "CatchResponseError('Export failed, code 404',)"
 2                 DELETE Board Delete: "CatchResponseError('Board remove failed, code: 404',)"
 3                 GET Version Api: "CatchResponseError('Version api failed, code: 0',)"
 26                PUT Ticket Move: "CatchResponseError('Failed to move ticket, code: 404',)"
 1                 GET Board export as csv: "CatchResponseError('Export failed, code 0',)"
 1                 GET Auth (old token): "CatchResponseError('Get Auth after logout != 401, code: 200',)"
 4                 GET Auth (old token): "CatchResponseError('Get Auth after logout != 401, code: 0',)"
 5                 GET Board Get Single: "CatchResponseError('Getting board by id failed, code 404',)"
 10                GET Board Get Tickets: "CatchResponseError('Getting board tickets failed, code 404',)"
 1                 GET Board Get Events: "CatchResponseError('Getting board events failed, code 0',)"
 2                 DELETE Ticket Delete: "CatchResponseError('Ticket deletion failed, code: 404',)"
 5                 GET Board Get All: "CatchResponseError('Getting boards failed, code 0',)"
 6                 DELETE Ticket Delete: "CatchResponseError('Ticket deletion failed, code: 0',)"
 7                 PUT Board Edit: "CatchResponseError('Board edit failed, code: 404',)"
 1                 GET Version Img: "CatchResponseError('Version img failed, code: 0',)"
 5                 GET Board Get Single: "CatchResponseError('Getting board by id failed, code 0',)"
 14                PUT Board Edit: "CatchResponseError('Board edit failed, code: 0',)"
 12                POST Ticket Comment: "CatchResponseError('Ticket comment failed, code: 0',)"
 15                POST Ticket Comment: "CatchResponseError('Ticket comment failed, code: 404',)"
 5                 PUT Ticket Move: "CatchResponseError('Failed to move ticket, code: 0',)"
 2                 GET Board Get Tickets: "CatchResponseError('Getting board tickets failed, code 0',)"
 2                 GET Auth request: "CatchResponseError('Get Auth failed, code: 0',)"
 3                 POST Logout: "CatchResponseError('Logout failed, code: 0',)"
 18                POST Ticket Create: "CatchResponseError('Creating new ticket failed, code: 0',)"
 1                 PUT Ticket Modify Content: "CatchResponseError('Ticket modify failed, code: 0',)"
 1                 GET Board export as image: "CatchResponseError('Export failed, code 404',)"
 23                PUT Ticket Modify Content: "CatchResponseError('Ticket modify failed, code: 404',)"
```

| Name | # reqs | # fails | Avg | Min | Max | Median | req/s |
|---|---|---|---|---|---|---|---|
| GET Auth (old token) | 23 | 5(17.86%) | 262 | 115 | 1402 | 130 | 0.10 |
| GET Auth request | 4 | 2(33.33%) | 560 | 120 | 1382 | 170 | 0.00 |
| POST Board Create | 23 | 0(0.00%) | 901 | 110 | 5593 | 470 | 0.00 |
| DELETE Board Delete | 14 | 6(30.00%) | 514 | 122 | 1677 | 170 | 0.00 |
| PUT Board Edit | 42 | 21(33.33%) | 591 | 133 | 1748 | 380 | 0.10 |
| GET Board Get All | 45 | 5(10.00%) | 414 | 116 | 1476 | 150 | 0.00 |
| GET Board Get Events | 3 | 1(25.00%) | 204 | 122 | 252 | 240 | 0.00 |
| GET Board Get Single | 22 | 10(31.25%) | 275 | 115 | 797 | 140 | 0.10 |
| GET Board Get Tickets | 29 | 12(29.27%) | 496 | 122 | 1834 | 340 | 0.00 |
| POST Board Share | 22 | 1(4.35%) | 611 | 108 | 1461 | 470 | 0.00 |
| GET Board export as csv | 1 | 1(50.00%) | 786 | 786 | 786 | 790 | 0.00 |
| GET Board export as image | 2 | 2(50.00%) | 1366 | 658 | 2075 | 660 | 0.00 |
| GET Board export as json | 2 | 0(0.00%) | 886 | 169 | 1604 | 170 | 0.00 |
| GET Board export as plaintext | 3 | 1(25.00%) | 263 | 138 | 452 | 200 | 0.00 |
| GET Login | 31 | 2(6.06%) | 421 | 199 | 1513 | 210 | 0.10 |
| POST Logout | 25 | 3(10.71%) | 372 | 115 | 1258 | 210 | 0.10 |
| POST Register | 5 | 0(0.00%) | 399 | 297 | 660 | 330 | 0.00 |
| POST Ticket Comment | 66 | 27(29.03%) | 487 | 119 | 1652 | 240 | 0.00 |
| POST Ticket Create | 110 | 34(23.61%) | 358 | 116 | 1589 | 160 | 0.20 |
| DELETE Ticket Delete | 11 | 8(42.11%) | 300 | 115 | 1243 | 140 | 0.00 |
| PUT Ticket Modify Content | 70 | 24(25.53%) | 523 | 219 | 1780 | 310 | 0.00 |
| PUT Ticket Move | 95 | 31(24.60%) | 485 | 222 | 1934 | 290 | 0.10 |
| PUT User Edit | 24 | 0(0.00%) | 265 | 117 | 811 | 140 | 0.00 |
| PUT User Password Change | 11 | 0(0.00%) | 606 | 292 | 1604 | 630 | 0.00 |
| GET Version Api | 20 | 3(13.04%) | 146 | 112 | 230 | 130 | 0.00 |
| GET Version Img | 13 | 1(7.14%) | 281 | 121 | 1402 | 130 | 0.00 |
| Total | 716 | 200(27.93%) | | | | | 0.80 |

# Liite 6. Startup.sh skripti

```
#!/bin/bash

cd /etc

sudo yum install -y git

sudo yum install -y gcc-c++ make &
wait $!
echo Installed gcc-c++ and make
sleep 2
sudo yum install -y openssl-devel &
wait $!
echo Installed openssl-devel
sleep 3
sudo yum install -y ppp &
wait $!

wget http://poptop.sourceforge.net/yum/stable/rhel6/x86_64/pptpd-1.4.0-1.el6.x86_64.rpm &
wait $!

sudo yum -y localinstall pptpd-1.4.0-1.el6.x86_64.rpm &

wait $!

sudo git clone https://github.com/mikkopoyhonen/noisetester.git


sudo sed -i 's/secure_path = \/sbin:\/bin:\/usr\/sbin:\/usr\/bin/secure_path =
\/sbin:\/bin:\/usr\/sbin:\/usr\/bin:\/usr\/local\/bin/g' /etc/sudoers

sudo git clone https://github.com/nodejs/node.git &
wait $!
echo Cloned node.git
sleep 2
echo entering nodejs workfolder

cd node
git checkout v0.12.2
sudo ./configure
make &
```

```
wait $!
sudo make install &
wait $!

sleep 3

sudo git clone https://github.com/isaacs/npm.git &
wait $!
echo cloned NPM

echo entering npm workfolder

cd npm
sudo make install &
wait $!

cd /etc

echo returned to /etc, now pulling Noise-tester


echo waiting for npm to finish installing
cd /etc/noisetester/server


sleep 20

npm install &
wait $!




read -p "Enter your IP address which you want to use for PPTP prefer elastic ip in amazon EC2 : " iplocal
echo "localip $iplocal" >> /etc/pptpd.conf

read -p "Enter ip address pool allocated to machines, for example 192.168.0.1-20 : " ipremote
echo "remoteip $ipremote" >> /etc/pptpd.conf

echo "Added $IPREMTOE as remote ip!"

read -p "Enter username for pptp : " uname
echo "Hi, $uname. Let us be friends!"


read -p "Enter your password for pptp : " pword
```

```
echo "$uname pptpd $pword *" >> /etc/ppp/chap-secrets

sudo sed -i 's/#ms-dns 10.0.0.1/ms-dns 8.8.8.8/g' /etc/ppp/options.pptpd

sudo sed -i 's/#ms-dns 10.0.0.2/ms-dns 8.8.4.4/g' /etc/ppp/options.pptpd


sudo sed -i 's/net.ipv4.ip_forward = 0/net.ipv4.ip_forward = 1/g' /etc/sysctl.conf

sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo service iptables save &
wait $!
sudo service iptables restart &
wait $!

sudo /sbin/sysctl -p

sudo service pptpd stop

sleep 1

sudo tc qdisc add dev eth0 root netem

sudo service pptpd start &
wait $!

sudo chkconfig pptpd on

cd /etc/noisetester/server/src

sudo node index.js
```