jamk.fi

IPv6-Liikennegeneraattori

Matias Hämäläinen

Opinnäytetyö
Toukokuu 2015

Tietotekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Hämäläinen Matias	Opinnäytetyö	25.05.2015
	Sivumäärä 114	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
IPv6-Liikennegeneraattori

Koulutusohjelma
Tietotekniikan koulutusohjelma

Työn ohjaaja(t)
Saharinen Karo
Häkkinen Antti

Toimeksiantaja(t)
Hallberg Jani
Jyväskylän ammattikorkeakoulu

Tiivistelmä

Opinnäytetyö toteutettiin Jyväskylän ammattikorkeakoulun Lutakon toimipisteeseen. Työn tavoitteena oli luoda IPv6 -protokollalla toimiva pakettigeneraattori, jonka pääasiallinen tarkoitus on verkon kuormittaminen suurella pakettimäärällä, ja näin mahdollistaa esimerkiksi palvelun laadun testaus SpiderNet ympäristössä.

Ohjelmointikieleksi valikoitui Python, ja työn keskeisiä vaatimuksia olivat muun muassa graafinen käyttöliittymä sekä laitteen rajapinnoista kulkevan liikenteen määrän esittäminen visuaalisessa muodossa graafin avulla.

Työn toteutus ja testaus suoritettiin aluksi virtuaaliympäristössä käyttäen kahta Ubuntu -päätelaitetta ja yhtä VyOs -käyttöjärjestelmällä toimivaa reititintä. Ohjelman toiminta todennettiin myös SpiderNet – verkossa.

Lopputuloksena saatiin ohjelma, joka sisältää kaikki alkuperäiset vaatimukset sekä hieman ylimääräisiä ominaisuuksia kuten komentolinjan toiminnallisuuden. Ohjelma mahdollistaa verkkoon konfiguroitujen QoS-asetusten testauksen kuormittamalla verkkoa yli sen kantokyvyn, joten sitä voidaan käyttää aiheeseen liittyvissä laboratorioharjoituksissa.

Avainsanat (asiasanat)
Python, IPv6, GUI, multithreading, socket

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Description

Author(s)	Type of publication	Date
Hämäläinen Matias	Bachelor's thesis	25.05.2015
		Language of publication: Finnish
	Number of pages 114	Permission for web publication: x

Title of publication
IPv6 traffic generator

Degree programme
Bachelor's Degree in Information technology

Tutor(s)
Saharinen Karo
Häkkinen Antti

Assigned by
Hallberg Jani
JAMK University of Applied Sciences

Abstract

The thesis was assigned by JAMK University of Applied Sciences. The purpose of the thesis was to provide JAMK with a packet generator that is capable of generating IPv6 traffic. It is mainly used to test QoS implementations in an IPv6 network.

The program was created by using Python as the programming language. The key requirements were a GUI and the ability to represent the traffic flow through the device's interfaces in visual format.

The program was tested in a virtual environment consisting of two Ubuntu clients and VyOs router. It was also tested in a physical environment existing in SpiderNet.

The finished product provides all key requirements and some extra features, such as the ability to run the program by using only the command line. The program enables the user to test QoS settings in an IPv6 network by overloading it with packets, so it can be used in related laboratory exercises.

Keywords/tags (subjects)
Python, IPv6, GUI, multithreading, socket

Miscellaneous

Sisältö

Kuviot

Taulukot

Lyhenteet

BSD	Berkeley Software Distribution
DAD	Duplicate Address Detection
DHCP	Dynamic Host Configuration Protocol
DHCPv6	Dynamic Host Configuration Protocol version 6
ESP	Encapsulating Security Payload
EUI	Extended Unique Indetifier
GUI	Graphical User Interface
IANA	Internet Assigned Numbers Authority
ICMPv6	Internet Control Message Protocol version 6
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ISP	Internet service provider
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
LXDE	Lightweight X11 Desktop Environment
MAC	Media Access Control
NAT	Network Address Translation
OSI-malli	Open Systems Interconnection Reference Model
POSIX	Portable Operating System Interface for Unix
QoS	Quality of Service
RAM	Random Access Memory
RFC	Request for Comments
RIR	Regional Internet Registry
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UDS	Unix Domain Sockets

1. Työn lähtökohdat

1.1 Työn toimeksiantaja

Jyväskylän ammattikorkeakoulu, eli lyhyemmin JAMK, tarjoaa useita eri koulutuslinjoja kampuksillaan, jotka sijaitsevat Jyväskylässä, sekä Saarijärven Tarvaalassa. JAMK:ssa opiskelee yli 8 500 opiskelijaa yli 70 maasta ja työllistymisprosentti JAMK:sta valmistuneilla opiskelijoilla on noin 80%. Lisäksi kaikkiin tutkintoihin kuuluu viisi opintopistettä yrittäjyyskoulutusta. (ks. Miksi JAMK 2015).

Opinnäytetyön toimeksiantajana toimi JAMK:n Lutakon kampus, joka toimii osoitteessa Piippukatu 2 ja työ toteutettiin tietotekniikan koulutusohjelman käyttöön.

1.2 Työn tavoitteet

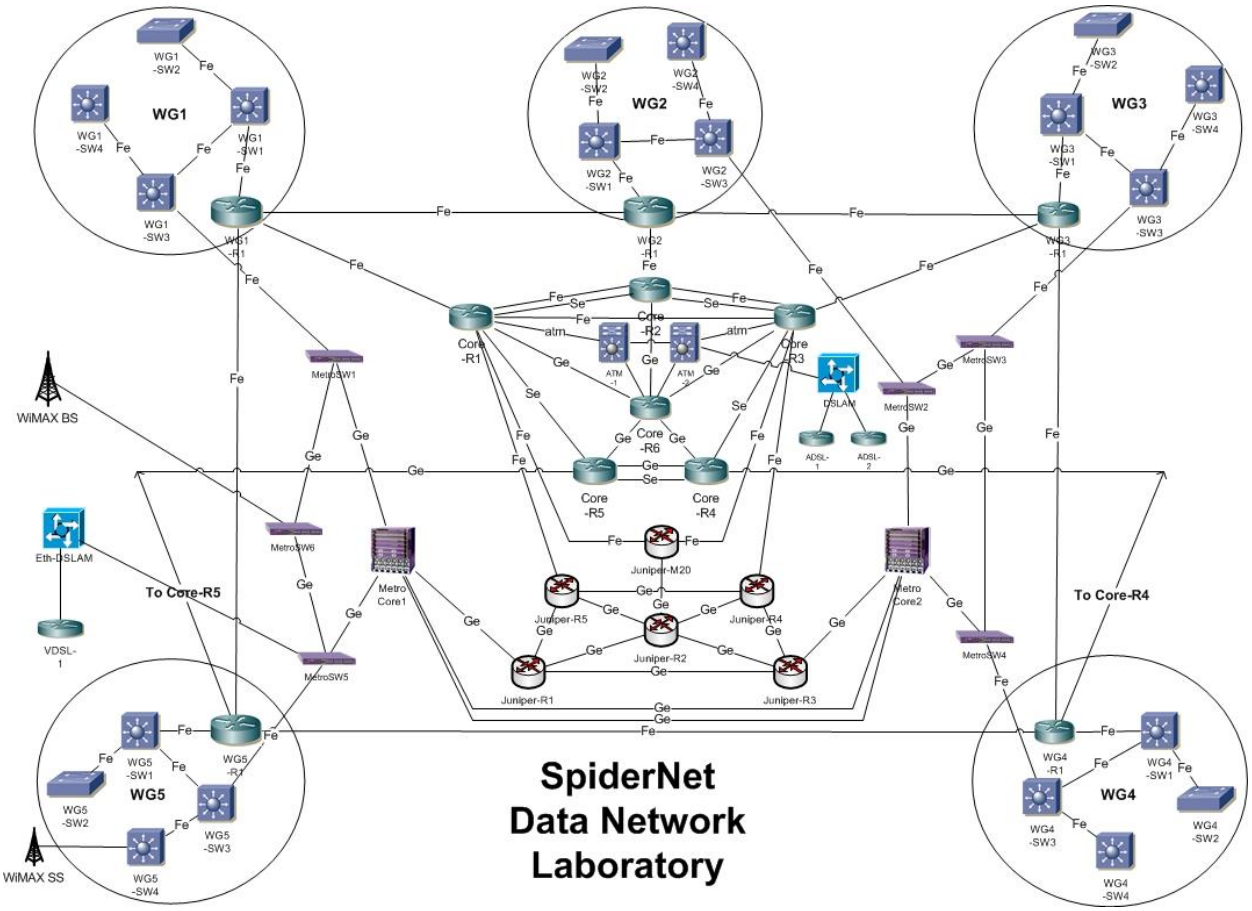
Opinnäytetyön tavoitteena oli perehtyä Python-ohjelmointikieleen Linux-ympäristössä ja sen pohjalta rakentaa IPv6:lla toimiva liikennegeneraattori. Ohjelman vaatimuksiin kuului graafisen käyttöliittymän ja graafin lisäksi socket-infrastruktuurilla toimiva verkkorajapinta palvelin/pääteohjelma-mallilla. Erityisesti ohjelman suunnittelussa tuli ottaa huomioon palvelimen ja pääteohjelman helppokäyttöisyys sekä selkeys, johon kuului lukuisten eri virheilmoitusten sisällyttäminen käyttöliittymään.

Ylimääräisenä tavoitteena oli luoda ohjelmalle komentoriviltä toimiva käyttöliittymä, joka pystyy näyttämään olemassa olevat komennot sisältävän ohjetekstin sekä toimimaan niin pääteohjelmana kuin palvelimenakin.

1.3 Toteutusympäristö

Toteutusympäristönä toimi tietotekniikan koulutusohjelman käytössä oleva SpiderNet-ympäristö. Ympäristöä on kehitetty yli kymmenen vuotta ja sen kehitys jatkuu edelleen. Se sisältää paljon verkko-operaattoreiden ja palveluntarjoajien käyttämää teknologiaa suljetussa ympäristössä. SpiderNettiä käytetään yleensä tietoverkkoteknologian kursseilla, mutta toisinaan sitä käytetään myös tutkimus- ja kehitysprojekteissa. (JAMK SpiderNet. 2009).

SpiderNet koostuu useista Ciscon, Juniperin sekä MetroEthernetin laitteista. Tarkempi topologia löytyy Kuviosta 1, josta nähdään verkkoon kuuluvat laitteet ja niiden kytkennät. (JAMK SpiderNet. 2009).



KUVIO 1. SpiderNetin topologia

2. Ohjelmointikieli ja moduulit

2.1 Python

Python -ohjelmointikielen ensimmäinen versio julkaistiin 1980-luvun loppupuolella. Tämän jälkeen Pythonista on julkaistu kaksi suurempaa versiota joista aikaisempi 2.0 julkaistiin vuonna 2000 ja joka toi kieleen muun muassa roskienkeruumekanismin ja Unicode-tuen, sekä vuonna 2008 versio 3.0 joka uudistusten takia ei ole yhteensopiva 2.x – versioiden kanssa. (Should I use Python 2 or Python 3 for my development activity? 2014)

Vaikka varsinainen kehitystyö Python 2:lle on jo lopetettu, on se vielä yleisesti käytetyin Pythonin versio. Tämä johtuu siitä, että on olemassa useita laajoja kirjastoja joihin ei ole vielä onnistuttu luomaan Python 3 tukea ja siitä, että Python 2 on vielä useissa käyttöjärjestelmissä asennettu oletuksena. (Should I use Python 2 or Python 3 for my development activity? 2014)

2.2 PyQt

PyQt on yksi kahdesta suosituimmasta Python sidoksesta järjestelmäriippumattomaan Qt-kirjastoon GUI/XML/SQL C++ rakenteessa. PyQt:n kehitti Riverbank Computing Limited, vaikka Qt:tä itseään kehitetään osana Qt-projektia. PyQt tarjoaa sidokset Qt 4:lle sekä Qt 5:lle. (Boddie 2014).

PyQt on saatavilla kahdessa versiossa: PyQt4:ssä, joka rakentuu Qt4.x:ää ja Qt5.x:ää vasten, sekä PyQt5:ssä, joka rakentuu vain Qt5:tä vastaan. Molemmat näistä versioista voidaan rakentaa Python 2:lla ja 3:lla. (Boddie 2014)

PyQt sisältää tavallisen GUI-työkalupakin lisäksi muun muassa seuraavat ominaisuudet: socket, threading, Unicode, SQL-database, SVG, OpenGL, XML sekä

regular expressions. Lisäksi PyQt:een kuuluu toimiva verkkoselain ja suuri määrä GUI-widgettejä. (Boddie 2014)

2.3 Qwt

Qwt-kirjasto sisältää GUI-komponentteja ja apuluokkia, jotka ovat yleisesti hyödyllisiä teknistä ohjelmointia varten. Kaksiulotteisen rakenteen lisäksi se tarjoaa asteikot, liukusäätimet, kellotaulut, lämpömittarit, kompassit ja useita muita tapoja hallita ja näyttää arvoja näytöllä. (Rathmann 2014)

Qwt 6.1:stä on mahdollista käyttää kaikissa ympäristöissä, joista löytyy Qt:n versio 4 tai korkeampi. (Rathmann 2014)

Qwt:n huono puoli on, että se toimii pelkästään ohjelmissa, jotka käyttävät Qt-pohjaista graafista käyttöliittymää. Se on kuitenkin erittäin tehokas datan esittämisessä, koska sen reaaliaikainen päivittäminen on erittäin helppoa.

Qwt pystyy käyttämään Qt-kirjastossa tulevia muotoiluja, kuten esimerkiksi värejä, joita on mahdollista käyttää kaikissa Qwt-kirjaston graafiin liitettävissä elementeissä.

2.4 Socket

Pythonin *Socket*-moduuli mahdollistaa kommunikoinnin verkon yli käyttämällä BSD – *socket*-rajapintaa. Se sisältää *socket*-luokan, joka hoitaa datakanavan, sekä funktioita, jotka liittyvät verkossa suoritettaviin tehtäviin. Tällaisia tehtäviä ovat muun muassa palvelimen nimen muuttaminen osoitteeksi sekä datan muokkaaminen lähettämistä varten. (Addressing, Protocol Families and Socket Types 2010)

Socket on yksi päätepiste kommunikaatiokanavassa prosessien välillä joko lokaalisti, tai verkkokanavan yli. *Socketeilla* on pääasiallisesti kaksi ominaisuutta, joilla voidaan hallita tapaa lähettää tietoa. Nämä tavat ovat osoiteperhe(*address family*), joka hallitsee OSI-mallin verkkokerroksella käytettävää protokollaa, sekä *socket type*, joka määrittelee kuljetuskerroksella käytettävän protokollan. (Addressing, Protocol Families and Socket Types 2010)

Python tukee kolmea osoiteperhettä. Näistä yleisimmin käytettyä, AF_INET, käytetään IPv4-osoitteistuksessa. Tämä osoiteperhe on vielä tällä hetkellä käytetyin. (Addressing, Protocol Families and Socket Types 2010)

AF_INET6-Osoiteperhettä käytetään IPv6-osoitteistukseen.

AF_UNIX-osoiteperhe on tarkoitettu UDS:lle, joka on protokolla prosessien väliselle kommunikoinnille POSIX-yhteensopivissa järjestelmissä. Yleisesti UDS suo käyttöjärjestelmälle mahdollisuuden siirtää tietoa suoraan prosessilta prosessille käyttämättä verkkokirjastoja. Koska UDS ja AF_INET kuuluvat samaan moduuliin, voidaan samaa koodia käyttäen hyötyä tehokkaasta prosessienvälisestä kommunikoinnista sekä lähettää dataa verkon yli. (Addressing, Protocol Families and Socket Types 2010)

Neljännen kerroksen socket

Neljännen kerroksen *socketista* puhuttaessa puhutaan *socketista*, joka käsittelee liikennettä OSI-mallin neljännellä eli kuljetuskerroksella. Vaikka neljännen kerroksen protokollia on useampia, tarkoitetaan tässä yhteydessä joko TCP tai UDP protokollaa. (Mecki 2013)

TCP/UDP-mallin yhteys tunnistetaan listalla, joka koostuu viidestä arvosta:

```
{<protocol>, <src addr>, <src port>, <dest addr>, <dest port>}
```

protocol arvo määrittelee protokollan, eli se on yhteydestä riippuen joko UDP tai TCP. *src addr* sisältää lähettävän osoitteen ja *src port* lähettävän prosessin portin. *dest addr* puolestaan sisältää kohdeosoitteen ja *dest port* vastaanottavan prosessin portin. (Mecki 2013)

Jokainen uniikki yhdistelmä näistä viidestä arvosta määrittelee yhteyden. Tästä johtuen kahdella yhteydellä ei voi olla täsmälleen samaa yhdistelmää näistä viidestä arvosta, sillä silloin laite ei kykenisi erottamaan yhteyksiä toisistaan. (Mecki 2013)

Kannan protokolla määritellään, kun se luodaan käyttämällä *socket()*-funktiota. Lähdeosoite sekä portti asetetaan *bind()*-funktiossa. Kohdeosoite ja portti asetetaan *connect()*-funktiossa. UDP:llä toimivan *socketin* ei kuitenkaan välttämättä tarvitse käyttää *connect()*-funktiota, sillä UDP on yhteydetön protokolla. (Mecki 2013)

SO_REUSEADDR toiminta

SO_REUSEADDR on tärkeä optio ohjelmissa, joissa *socket* täytyy sen sulkeuduttua saada nopeasti uudestaan päälle saman IP-osoitteen ja portin yhdistelmään. (Mecki 2013)

Socketilla on olemassa puskuri lähetettäviä paketteja varten ja vaikka *send()*-funktio onnistuu, ei se tarkoita automaattisesti sitä, että paketti on lähetetty laitteelta ulos, vaan sitä, että se on lisätty lähtevän liikenteen puskuriin. UDP kannoissa paketti lähetetään yleensä todella nopeasti sen puskuriin lisäämisen jälkeen, mutta TCP-pakettien kanssa siinä voi mennä huomattavasti pidemmän aikaa. Tämän takia on mahdollista että TCP-socketia suljettaessa lähetyspuskurissa on vielä paketteja odottamassa lähetystä, mutta ohjelma lukee ne jo lähetetyiksi koska *send()* kutsu onnistui. Jos TCP-socket sulkeutuisi heti sitä siltä pyydettäessä, katoaisi kaikki puskurissa oleva data, eikä ohjelma edes huomaisi sitä. Koska TCP:stä puhuttaessa puhutaan luotettavasta protokollasta, eikä tällainen datan häviäminen ole kovin luotettavaa, menee *socket* sulkeutumispyynnön saapuessa TIME_WAIT -tilaan ja odottaa että puskuri tyhjentyy, tai aikarajoitus ylittyy ennen lopullista sulkeutumista. (Mecki 2013)

Aikaa, jonka kernel odottaa ennen *socketin* sulkemista sen puskurin sisällöstä riippumatta, kutsutaan nimellä viipymisaika (*Linger Time)*. Viipymisaika on suurimmalla osalla käyttöjärjestelmistä konfiguroitavissa ja se on oletusarvoltaan melko pitkä (minuutista kahteen minuuttiin). *Socket* on koko tämän ajan TIME_WAIT -tilassa. (Mecki 2013)

Jos SO_REUSEADDR ei ole käytössä järjestelmä olettaa TIME_WAIT-tilassa olevan *socketin* olevan yhä sidottu lähdeosoitteeseen ja porttiin ja kaikki yritykset sitoa uusi *socket* samaan osoitteeseen ja porttiin epäonnistuvat niin kauan, että vanha *socket* on suljettu lopullisesti. Koska tässä voi kestää pahimmillaan useita minuutteja riippuen järjestelmän asetuksista, on lähes välttämätöntä käyttää SO_REUSEADDR asetusta, joka mahdollistaa *socketin* sitomisen samaan osoitteeseen ja porttiin kuin mihin TIME_WAIT -tilassa oleva *socket* on sidottu. (Mecki 2013)

2.5 Threading

Threading-moduuli on korkeamman tason rajapinta alemman tason *Thread*-moduulin päällä. Moduulin tarkoitus on tehdä useiden säikeiden hallinnoimisesta helpompaa. (Threading 2015)

Moduuli tarjoaa funktioita, joiden avulla voidaan muun muassa luoda uusia säikeitä, nimetä ne ja siirtää niille argumentteja, nostaa tapahtumia, hakea halutun säikeen nimi ja luoda lukkoja. (Threading 2015)

threading.Event() Funktio, joka palauttaa uuden tapahtumaobjektin. Tapahtuma hallinnoi lippua, joka voidaan asettaa *True*-tilaan *set()*-metodilla ja tyhjätä takaisin False-tilaan metodilla *clear()*. *wait()*-metodi estää niin kauan, että lippu on asetettu *True*-tilaan.

threading.Local() Luokka, joka esittää säikeelle lokaalia dataa. Jos instanssi on luotu säiekohtaiseksi jokaisessa säikeessä, sisältävät jokaisen säikeen kyseiset instanssit eri arvon.

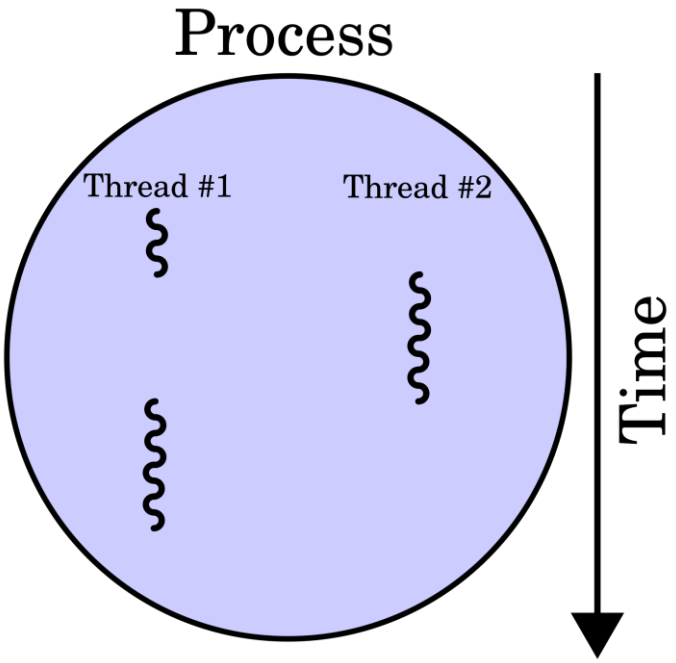
threading.Lock() Luo uuden primitiivisen lukko-objektin. Kun yksi säie on hankkinut sen, kaikki muiden säikeiden yritykset saada lukko estävät niin kauan, että se vapautuu.

threading.Thread	Luo uuden hallinnoitavan säikeen. Uusi säie aloitetaan *start()*-metodilla, joka herättää *run()*-metodin uuteen hallinnoitavaan säikeeseen.
Join([timeout])	Odottaa, kunnes säie on sammunut. Tarkoittaa sitä, että kutsuva säie estää kaikki kutsut niin kauan, että säie, jonka *join()*-metodia kutsuttiin, sammuu. Sammuminen voi tapahtua joko normaalisti, virheen sattuessa tai valinnaisen aikarajan ylittyessä.
setDaemon()	Säie on mahdollista merkata taustaprosessiksi komennolla *setDaemon(True)*. Taustaprosessit sulkeutuvat ohjelman sammuessa äkillisesti eivätkä niiden käyttämät resurssit välttämättä vapaudu sulkeutumisen yhteydessä.

Säie

Säie on tietojenkäsittelyopissa pienin osa, jota vuorontaja voi hallinnoida. Suurimmassa osassa tapauksia säie on osa olemassa olevaa prosessia, eli se käyttää sen prosessin resursseja johon se kuuluu. Yhteen prosessiin voi myös kuulua useita säikeitä jolloin puhutaan monisäikeistämisestä.

Kuviossa 2 havainnollistetaan, miten vuoronantaja käyttää prosessin resursseja kahden säikeen välillä vuorotellen. Ajan kuluessa Y-akselin suuntaisesti ensimmäisenä vuorossa on säie #1 (Thread #1). Kuvion 2 piirtohetkellä säikeellä #1 on enää hetki aikaa olla suorittavassa tilassa, jonka jälkeen suoritusvuoro siirtyy säikeelle #2 (Thread #2). Säie #2 suorittaa ennalta määritellyn ajan, jonka jälkeen vuoronantaja vaihtaa suorittavan vuoron takaisin säikeelle #1.



KUVIO 2. Säikeiden ja vuoronantajan toiminta prosessissa

Monisäikeistämisen edut ja haitat

Yksi tärkeimpiä monisäikeisyyden ominaisuuksia on responsiivisuus. Jos yksisäikeisessä ohjelmassa säie on pidemmän aikaa estävässä tilassa, esimerkiksi suorittamassa raskasta tehtävää, koko ohjelma voi näyttää jähmettyneeltä. Tällaisia kutsuja varten, jotka voivat pitää säikeen pitkänkin aikaa estävässä tilassa, voidaan luoda työskentelevä säie jolle pääsäie siirtää raskaan tehtävän suorittamisen samalla, kun se itse pysyy responsiivisena käyttäjän syötteelle. (Jenkov 2015)

Monisäikeistäminen myös mahdollistaa laitteen resurssien paremman optimoinnin minimoimalla ajan, jonka suoritin joutuu olemaan ns. tyhjäkäynnillä. Tällaiseen tilanteeseen voidaan päätyä esimerkiksi, kun ohjelma haluaa lukea levyltä useita kookkaita tiedostoja ja sen jälkeen prosessoida ne. Levyltä voidaan lukea vain yhtä tiedostoa kerrallaan, joten ohjelma joutuu odottamaan levyn syötettä niin kauan, että se on valmis ja voi vasta sen jälkeen aloittaa tiedoston prosessoinnin. Tämä täytyy tehdä erikseen jokaiselle levyltä pyydetylle tiedostolle. (Jenkov 2015)

Kuviossa 3 on tilanne, jossa levyltä pyydetään kaksi tiedostoa (tiedostot A ja B), joiden molempien luku kestää viisi sekuntia ja prosessointi kaksi sekuntia. Yhteensä tähän prosessiin menee siis aikaa 14 sekuntia. (Jenkov 2015)

```
  5 seconds reading file A
  2 seconds processing file A
  5 seconds reading file B
  2 seconds processing file B
----------------------
 14 seconds total
```

KUVIO 3. Tiedostojen luku ja prosessointi tapa A

Kuviossa 4 käsitellään sama tilanne, mutta ensimmäisen tiedoston luvun jälkeen CPU aloittaakin tiedosto A:n käsittelyn uudessa säikeessä samanaikaisesti, kun se odottaa että tiedosto B on luettu. Yhteensä tähän prosessiin menee tätä prosessointijärjestystä käyttämällä 12 sekuntia. (Jenkov 2015)

```
  5 seconds reading file A
  5 seconds reading file B + 2 seconds processing file A
  2 seconds processing file B
----------------------
 12 seconds total
```

KUVIO 4. Tiedostojen luku ja prosessointi tapa B

Kuviossa 4 esitetty ohjelma on mahdollista kirjoittaa myös yhdellä säikeellä, mutta silloin täytyy seurata jokaisen tiedoston luku- ja prosessointitilaa. Monisäikeistämisellä on mahdollista luoda työskentelevä säie jokaista haluttua tiedostoa varten. Jokainen näistä säikeistä on estävässä tilassa niin kauan, että tiedoston luku on saatu päätökseen samanaikaisesti, kun muut säikeet voivat käyttää CPU:ta prosessoimaan niitä osia tiedostosta, jotka ne ovat jo lukeneet. Tämän seurauksena CPU:n ja levyn tyhjäkäyntiaika saadaan minimoitua. Lisäksi se on helpompi ohjelmoida, sillä jokaisen säikeen täytyy seurata vain yhtä tiedostoa. (Jenkov 2015)

Monisäikeisyydellä on kuitenkin omat ongelmansa. Vaikka Kuviossa 4 olevassa, hyvin yksinkertaisessa esimerkissä, monisäikeistäminen onnistuu helposti ja sillä voidaan optimoida ohjelman suorituskykyä, aiheuttaa useiden säikeiden samanaikainen ajaminen ongelmia ajastamisen ja yhteisten resurssien käytön kanssa. Myös vikojen paikallistaminen monisäikeisessä ohjelmassa on huomattavasti haastavampaa erilaisten skenaarioiden eksponentiaalisen kasvun takia. (Ignatchenko 2010)

3. Verkkoprotokollat

3.1 Ipv6

IPv6, tai joskus IPng (Internet Protocol next generation), on uusin IETF:n kehittämä versio IP:stä. Sen tarkoituksena on tarjota ratkaisu kauan odotetulle IPv4-osoitteiden ehtymiselle sekä tarjota useita parannuksia vanhaan protokollaan. (Just how many IPv6 addresses are there? Really? 2012)

IPv6 laajentaa osoitekentän 32 bitistä 128 bittiin. Teoreettisesti käytettävissä olevien uniikkien osoitteiden enimmäismäärä on IPv6:sta käytettäessä 2^128 eli 340,282,366,920,938,463,463,374,607,431,768,211,456. Tässä ei kuitenkaan oteta huomioon kahta tosiasiaa. (Just how many IPv6 addresses are there? Really? 2012)

IANA on julkaissut vain osan IPv6 -osoiteavaruudesta julkiseen osoitteistukseen. RFC 2374:ssä määriteltiin (Nykyään vanhentunut RFC 3587:n tultua) että kaikissa julkisissa IPv6 -osoitteissa ensimmäisen kolmen bitin tulee olla asetettuna arvoon 001. Käytännössä tämä siis tarkoittaa, että jokainen julkinen IPv6-osoite:

a) alkaa joko numerolla kaksi tai kolme ja

b) Ensimmäinen heksadesimaaliryhmä on neljä heksadesimaalia pitkä

(Just how many IPv6 addresses are there? Really? 2012)

Jokainen pätevä IPv6 -osoite alkaa siis heksadesimaaliryhmällä, joka on muotoa 2xxx: tai 3xxx:. Lisäksi tulee ottaa huomioon, että jokainen julkisesti tarjolla oleva IPv6 -aliverkko on peitteeltään 64 bittiä. Kun nämä asiat otetaan huomioon, jää jäljelle 2^62 pisteestä pisteeseen tarkoitettua osoitetta. Tämä ei kuitenkaan tee varsinaista eroa tarjolla olevien IP -osoitteiden määrään, sillä tästäkin altaasta riittäisi satoja miljoonia osoitteita jokaista ihmistä kohti. (Just how many IPv6 addresses are there? Really? 2012)

IPv6-autokonfiguraatio ja osoitetyypit

IPv4:llä osoitteiden jakaminen tapahtuu useimmiten käyttämällä DHCP:tä. Myös IPv6:lle on tehty vastaava protokolla nimeltä DHCPv6, mutta tämän lisäksi sillä on myös tilaton autokonfiguraatiomekanismi, jollaista ei IPv4:llä ole. (Donzé 2004)

DHCP ja DHCPv6 ovat tilallisia protokollia, koska ne ylläpitävät taulukkoa ennaltamääritellyillä palvelimilla. Tilaton autokonfiguraatioprotokolla ei tarvitse lainkaan palvelinta tai välitintä, koska sillä ei ole tilaa mitä ylläpitää. (Donzé 2004)

Jokainen IPv6:lla toimiva järjestelmä (Reitittimiä lukuun ottamatta) pystyy luomaan oman globaalin IPv6-täsmälähetysosoitteen. Täsmälähetysosoite tarkoittaa tässä yhteydessä uniikkia rajapintaa. Osoitetyypeillä on hyvin määritellyt kohdealueet: *global, site-local* sekä *link-local.* (Donzé 2004)

link-local-osoitteet toimivat pelkästään kahden laitteen välisellä linkillä. Reitittimet, jotka voisivat ohjata tästä osoitteesta lähteviä paketteja eteenpäin, eivät saa tehdä sitä, sillä *link-local*-osoitteen uniikkiutta ei voida varmentaa. (Donzé 2004)

site-local-osoitteet ovat puolestaan rajoitettu organisaatioverkon sisäisiksi, eli reunareitittimet eivät saa ohjata *site-local*-kohdeosoitteella lähetettyjä paketteja ulos organisaatioverkosta. IETF pohtii tällä hetkellä tapaa joko poistaa tai korvata *site-local*-osoitteet. (Donzé 2004)

global-osoitteet ovat koko Internetin laajuisia. Tällaisilla kohdeosoitteilla varustetut paketit voidaan reitittää mihin päin Internetiä tahansa. (Donzé 2004)

link-local osoitteen luominen ja vahvistus toimii seuraavasti:

1. Luodaan tunniste, joka on oletettavasti uniikki linkkivälille
2. Luodaan väliaikainen osoite
3. Tämän osoitteen ainutlaatuisuus linkkivälillä varmennetaan

4. Jos uniikki, osoite vaiheesta kaksi asetetaan rajapinnan osoitteeksi, jos ei, niin vaaditaan manuaalista toimintaa

IPv6:ssa käytetty 128 bittiä pitkä osoite koostuu kahdesta osasta: aliverkkoetuliitteestä, joka edustaa verkkoa mihin rajapinta on liitetty, sekä lokaalista tunnisteesta. Tunniste saadaan yleensä rajapinnan MAC-osoitteesta käyttämällä IEEE:n luomaa EUI-64 algoritmia. EUI-64 standardi kertoo kuinka IEEE 802(eli MAC) osoite voidaan pidentää 48-bitistä 64-bittiin lisäämällä 16-bittiä (0xFFFE) IEEE 802-osoitteen 24.n bitin kohdalle. (Donzé 2004)

MAC-osoite 00-0C-29-C2-52-FF saadaan siis muutettua käyttämällä EUI-64 algoritmia muotoon 00-0C-29-FF-FE-C2-52-FF. Käyttämällä IPv6-merkintätapaa saadaan tämä puolestaan muotoon 000C:29FF:FEC2:52FF. RFC3513:ssa määritellään otsikon *2.5.1 Interface Identifiers* alla että EUI-64 muotoiset tunnisteet luodaan invertoimalla niin kutsuttu “u” bitti (universaali/lokaali bitti IEEE EUI-64 sanastossa), kun muodostetaan rajapintatunnistetta IEEE EUI-64 tunnisteista. Jos ”u” bitti on asetettu arvoon yksi (1) on kyseessä globaali osoite, jos se puolestaan on nolla (0) on osoite lokaali. ”u” bitti on ensimmäisen tavun toinen bitti, joten se tarkoittaa että yllä olevan esimerkin osoite muuttuu muotoon 020c:29ff:fec2:52ff. (Donzé 2004)

Autokonfiguraation toisessa vaiheessa asetetaan luodun tunnisteen eteen tunnettu etuliite fe80::/64. Tämän seurauksena saadaan siis esimerkin tapauksessa osoite fe80::20c:29ff:fec2:52ff. Osoite liitetään rajapintaan ja merkataan alustavaksi, sillä sen ainutlaatuisuus on vielä varmentamatta. (Donzé 2004)

Kolmatta vaihetta kutsutaan lyhenteellä DAD. Järjestelmä lähettää ICMPv6 paketteja linkille, jossa varmennus tapahtuu. Paketit sisältävät *Neighbor Solicitation* viestejä. Näiden pakettien lähdeosoite on määrittelemätön ”::” ja kohdeosoitteena on aiemmin määritelty alustava osoite. Jos toisella verkossa olevalla laitteella on tuo alustava osoite käytössä, lähettää se takaisin *Neighbor Advertisement* viestin. Tällöin laite tietää että sen haluama alustava osoite on toisen laitteen käytössä, eikä sitä voida asettaa rajapinnan osoitteeksi. (Donzé 2004)

Jos osoite ei ole minkään muun laitteen käytössä, poistetaan osoitteesta merkkaus, jossa se määritellään alustavaksi, ja asetetaan se rajapinnan viralliseksi osoitteeksi. (Donzé 2004)

Tavallisesti globaalit etuliitteet jaetaan asiakkaille palveluntarjoajien toimesta. Globaali osoite voidaan siis luoda käyttämällä EUI-64 tunnistetta ja globaalia etuliitettä, mikä on helppoa, mutta ei kovin turvallista yksityisyyden kannalta koska EUI-64 tunniste pysyy samana paikasta ja verkosta riippumatta. (Donzé 2004)

RFC 3041:n osiossa 3.2.1 määritellään kuinka voidaan luoda satunnaisesti globaali IPv6 -tunniste rajoitetulla eliniällä. Koska IPv6 -arkkitehtuuri sallii useita jälkiliitteitä rajapintaa kohden, asetetaan yhdelle rajapinnalle kaksi globaalia osoitetta. Yksi joka on luotu MAC-osoitteen perusteella, ja yksi satunnaisella tunnisteella. (RFC3041 2001, 8)

IPv6 -otsikko

IPv6-osoite koostuu kahdeksasta 16-bittisestä heksadesimaaliryhmästä. Osoitteet ovat yleensä hyvin pitkiä ja vaikeasti luettavissa, mutta protokollaan on sisäänrakennettu pari tapaa lyhentää osoite helpommin ymmärrettäväksi. Jos heksadesimaaliryhmän alussa on nollia, ne voidaan poistaa [1]. Lisäksi jos osoitteessa on yksi tai useampi pelkkiä nollia sisältävä heksadesimaaliryhmä peräkkäin, voidaan ne korvata kahdella kaksoispisteellä eli ’::’[2]. Tällainen lyhennys voidaan kuitenkin tehdä vain yhdessä kohtaa osoitetta. Esimerkki löytyy taulukosta 1 (Thomas)

Taulukko 1. IPv6-osoitteen lyhentäminen

Osoitteen editointi	Heksadesimaaliryhmien jono
Ei mitään	2001:dead:beef:0030:0000:0000:0000:0001
Nollien karsiminen[1]	2001:dead:beef:30:0:0:0:1
Tyhjien ryhmien karsiminen [2]	2001:dead:beef:30::1

Osoitteen muoto ja sen koko eivät ole ainoat asiat, jotka erottavat IPv6:n IPv4:stä, sillä näiden kahden protokollan kehykset ovat täysin erilaiset. IPv6-otsikon koostumus löytyy liitteestä 1. (RFC2460 1998, 3)

Versio	4 bitin IP-versionumero, joka on asetettu arvoon 6
Liikenneluokka	8 bitin liikenneluokka-kenttä, jonka tarkoitus on mahdollistaa liikenteen luokka- ja prioriteettierottelu reitittäville ja lähettäville laitteille.
Virtaetiketti	20 bitin virtaetikettikenttää käytetään merkitsemään laitteelta lähtevien pakettien järjestys tapauksissa, joissa reitittävältä laitteelta halutaan erityistä käsittelyä pakettien saapuessa. Tällaisia tilanteita voisivat olla esimerkiksi tarve tavallisesta poikkeavaan palvelun laatu- tai reaaliaikapalveluun.
Hyötykuorman Pituus	16-bitin järjestysluku. Sisältää IPv6-hyötykuorman, eli tätä otsikkoa seuraavan paketin osan pituuden. Kaikki paketissa olevat laajennetut otsikot sisällytetään hyötykuorman pituutta laskettaessa tässä kentässä näkyvään arvoon.
Seuraava Otsikko	8-bitin valitsin. Määrittää IPv6-osoitetta seuraavan otsikon tyypin. Kenttä käyttää samoja arvoja kuin IPv4
Hyppyrajoitus	8-bitin järjestysluku. Jokaisen hypyn jälkeen tämän kentän arvoa vähennetään yhdellä. Paketti pudotetaan jos tämän kentän arvo on nolla.
Lähdeosoite	128-bitin osoite jossa määritellään paketin lähettäjä.
Kohdeosoite	128-bitin osoite jossa määritellään paketin tarkoitettu vastaanottaja.

IPv6 -laajennusotsikot

IPv6:tta käytettäessä valinnainen tieto Internetkerroksista koodataan erilliseen otsikkoon, joka voidaan asettaa IPv6-otsikon ja ylemmän kerroksen otsikon väliin. Tällaiset laajennusotsikot erotetaan jokainen erillisellä Seuraava Otsikko-arvolla. Laajennusotsikoita voi olla yksi, useita tai ei yhtään. Taulukossa 2 nähdään tilanne, jossa laajennusotsikoita ei ole lainkaan. Taulukossa 3 laajennusotsikoita on yksi, ja taulukossa 4 kaksi. (RFC2460, 5)

Taulukko 2. Laajennusotsikot 1

IPv6 -otsikko **Seuraava Otsikko = TCP**	**TCP-otsikko + kuorma**

Taulukko 3. Laajennusotsikot 2

IPv6 -otsikko **Seuraava Otsikko = Reititys**	**Reititysotsikko** **Seuraava Otsikko = TCP**	**TCP-otsikko + kuorma**

Taulukko 4. Laajennusotsikot 3

IPv6 -otsikko **Seuraava Otsikko = Reititys**	**Reititysotsikko** **Seuraava Otsikko = Osa**	**Osaotsikko** **Seuraava Otsikko = TCP**	**Osa TCP otsikosta + kuorma**

Yhtä poikkeusta lukuun ottamatta laajennusotsikoita ei tarkastella tai prosessoida yhdelläkään laittella ennen kuin paketti saavuttaa laitteen, joka on määritelty IPv6 -otsikon kohdeosoitekentässä. Kohdelaite joko lukee laajennusotsikot, jos niitä on, tai siirtyy ylemmän tason otsikkoon. Laajennusotsikot täytyy käsitellä täsmälleen siinä

järjestyksessä, kun ne ovat paketissa, sillä lukuohjelma siirtyy seuraavaan otsikkoon vain, jos luettavassa otsikossa niin määritellään. (RFC2460, 5)

Aikaisemmin mainittu poikkeus tähän sääntöön on Hyppy-Hypyltä-Asetukset –otsikko, joka sisältää tietoa, joka täytyy tarkastella ja käsitellä jokaisella paketin matkalla olevalla laitteella, joihin myös lähettävä ja vastaanottava laite kuuluvat. Hyppy-Hypyltä-Asetukset-otsikon tulee olla heti IPv6-otsikon jälkeen. (RFC2460, 5)

Taulukosta 5 voidaan nähdä laajennusotsikot, joita IPv6-paketissa voidaan käyttää

Taulukko 5 Laajennusotsikot

Laajennusotsikko	Tyyppi	Määrittely
Hyppy-Hypyltä-Asetukset	0	Asetukset, jotka jokaisen matkalla olevan laitteen tulee tarkastella
Kohdeasetukset (ennen reititysotsikkoa)	60	Asetukset, jotka vain kohdelaitteen tarvitsee tarkastella
Reititys	43	Tavat joilla määritellään reitti tietosähkeille (käytetään mobiilin IPv6:n kanssa)
Osa (Fragment)	44	Sisältää parametrejä tietosähkeiden paloittelusta
Autentikointiotsikko	51	Sisältää tietoa jota käytetään paketin osien todentamiseen
Encapsulating Security Payload (ESP)	50	Kuljettaa kryptattua dataa turvallisille yhteyksille
Kohdeasetukset (ennen ylemmän tason otsikkoa)	60	Asetukset, jotka vain kohdelaitteen tarvitsee tarkastella
Mobiili (Ilman ylemmän tason otsikkoa)	135	Parametrejä käytetään mobiilin IPv6 kanssa

IPv4-osoitteiden ehtyminen

IPv4-osoitteita on yhteensä (4 294 967 296 kappaletta, eli 2^32). IPv4:n RFC julkaistiin vuoden 1981 syyskuussa eli aikana, jolloin vain harva omisti PC:n, kannettavat tietokoneet olivat vielä lastenkengissä ja ensimmäinen toimiva älypuhelin oli vielä yli vuosikymmenen päässä (Sager 2012). Internetyhteyteen kykenevien laitteiden määrän vähäisyyden ja niiden kalleuden johdosta ei tuota hieman yli neljän miljardin osoitteen määrää nähty millään lailla rajoittavana tekijänä, sillä kukaan ei uskonut, että teknologia kehittyisi näin paljon seuraavien vuosikymmenten aikana. Pelkän teknologisen kehityksen lisäksi Internetin palveluissa on tapahtunut suurta kehitystä viimeisen kolmen vuosikymmenen aikana kuten mm. sähköposti, WWW, pikaviesti- ja suoratoistopalvelut, verkkokauppajätit, P2P-teknologia sekä pilvipalvelut. Kaikki tämä on johtanut Internet-osoitteiden kysynnän valtavaan kasvuun ja täten vapaiden IP-osoitteiden hitaaseen ehtymiseen. (Pierre 2011)

IPv4-osoitteiden ehtyminen ei kuitenkaan tarkoita sitä, että Internet yhtäkkiä lakkaa toimimasta. Se tarkoittaa sitä, että palveluntarjoajien osoitealtaat ehtyvät, eikä niillä enää ole osoittaa vapaita IPv4-osoitteita. Tämä tapahtuu kuitenkin useissa vaiheissa, jotka eivät välttämättä tapahdu alla olevan listan järjestyksessä (IPv4 exhaustion details)

- IANA:n IPv4-osoiteallas ehtyy (3. helmikuuta 2011)
- RIR:en osoitealtaat ehtyvät
- Laajenevien verkkojen (ISP:t, yritykset yms.) osoitealtaat ehtyvät

Pahimmillaan tilanne, jossa kaikki IPv4-osoitteet olisivat käytössä, voisi johtaa siihen että uusien laitteiden ei olisi enää mahdollista saada IP-osoitetta ja näin ollen eivät kykenisi liittymään verkkoon lainkaan. Tämä saattaisi olla jo arkipäivää, ellei tätä skenaariota olisi onnistuttu viivyttämään NAT:n tehokkaalla käytöllä. NAT

mahdollistaa useiden laitteiden näkymisen julkiseen verkkoon yhtenä IPv4-osoitteena. Tämän ansiosta palveluntarjoajat voivat osoittaa yhdelle asiakkaalle, jolla on useita Internetiin yhdistettäviä laitteita, vain yhden IP-osoitteen. (IEEE-USA White Paper, Next Generation Internet 2009, 10)

IPv6 vs. IPv4

Vaikka puhutuin hyöty IPv6-teknologiaan siirtymisessä onkin osoitealtaan laajentuminen, tarjoaa se lisäksi useita teknologisia muutoksia, jotka on vartavasten suunniteltu toimimaan maailmanlaajuisessa verkossa. Näihin kuuluvat mm. (Beal 2014)

- NAT:lle ei ole enää tarvetta
- Konfiguroi itsensä automaattisesti
- Parempi monilähetysreititys
- Yksinkertaisempi otsikon muoto
- Yksinkertaistettu, tehokkaampi reititys.
- Sisäänrakennettu palvelun laadun hallinta (QoS), toisin sanoen virtaetiketti (Flow Label)
- Sisäänrakennettu autentikointi ja yksityisyyden tuki
- Joustavat lisäosat ja laajennukset
- Helpompi ylläpitää (Ei välttämätöntä tarvetta DHCP:lle)

3.2 UDP

UDP on yksi Internetin perustavanlaatuisista protokollista. UDP suunniteltiin vuonna 1980 ja sen toiminnallisuus on määritelty RFC 768:ssa.

UDP käyttää yksinkertaista, yhteydetöntä tiedonsiirron mallia minimaalisilla protokollamekanismeilla. Siinä ei ole minkäänlaisia kädenpuristuskeskusteluja, mistä johtuen yhteys on haavoittuvainen, jos alla oleva verkko on epäluotettava. Pakettien toimituksesta, järjestämisestä ja kopioiden lähettämiseltä suojauksesta ei ole minkäänlaisia takeita. UDP tarjoaa tarkistussumman datan eheyden varmentamiseksi, sekä porttinumerot tietosähkeiden ohjausta varten. (RFC 768 1980, 1)

Taulukossa 6 esitetään UDP -otsikon koostumus. Otsikosta löytyy lähdeportti, kohdeportti, otsikon pituus, tarkistussumma sekä otsikossa kulkeva data. Koska datan määrä ei ole vakio, ei sille voida määritellä kiinteitä arvoja ja täten sisällyttää tarkalleen taulukkoon, joten taulukko on jätetty lopusta avoimeksi ja se esittää vain mistä datatavut alkavat. (RFC 768 1980, 1)

Taulukko 6. UDP-otsikko

bitit	0 - 15	16 - 31
0	Lähdeportti	Kohdeportti
32	Pituus	Tarkistussumma
64	Datatavut	

Kentät

Lähdeportti määrittelee tarvittaessa lähettävän prosessin osoitteen, eli se ei ole pakollinen. Tässä tapauksessa lähdeportiksi asetetaan oletusarvo eli nolla. Vastaanottava laite voi olettaa, että tilanteessa jossa lähettävä laite haluaa vastauksen, lähetetään se lähdeporttikentän määrittelemään porttiin. (RFC 768 1980, 1)

Kohdeportti toimii yhteistyössä kohde IP:n kanssa. IP-osoitteen määrittäessä vastaanottavan laitteen, määrittelee kohdeportti vastaanottavan prosessin. (RFC 768 1980, 2)

Pituus kertoo kyseessä olevan käyttäjätietosähkeen pituuden. Tähän lasketaan myös otsikko ja data. (RFC 768 1980, 2)

Tarkistussumma lasketaan koko paketista. Se on vapaaehtoinen, eli se voidaan ohittaa ja kentän kaikki bitit asettaa nollaksi. UDP-otsikon tarkistussumma on kuitenkin ainoa tapa tarkistaa hyötykuorman oikeellisuus siirron jälkeen. (Internetix oppimateriaalit TCP- ja UDP-protokollat 1980)

4. Toteutus

4.1 Käyttöjärjestelmän valinta

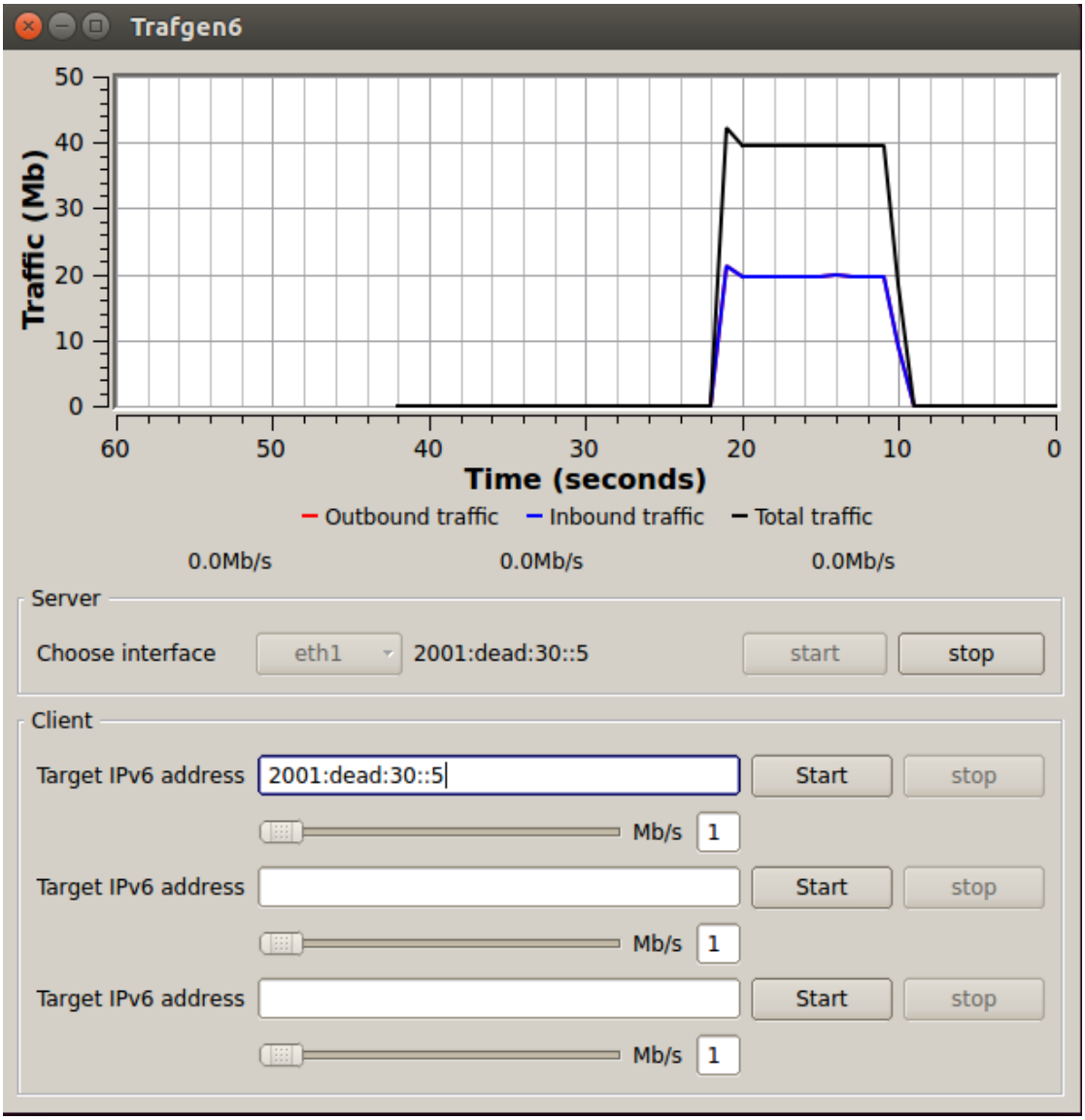
Vaikka suurin osa työstä suoritettiinkin Ubuntu-jakelulla, tultiin lopulta siihen tulokseen että huomattavasti kevyempi Lubuntu on parempi vaihtoehto. Lubuntu perustuu uuteen kevyempään graafiseen käyttöliittymään (LXDE), josta on karsittu mahdollisimman paljon Ubuntu-jakelun ylimääräisiä ominaisuuksia säilyttäen silti kaikki perustavanlaatuiset tärkeät osat. Kevyt käyttöliittymä laskee käyttöjärjestelmän rautatason vaatimuksia. Lubuntua pyörittävän laitteen RAM vaatimus minimiasetuksilla pelkän käyttöjärjestelmän pyörittämiseen on 256MB, mutta hieman raskaampien paikallisten ohjelmien, kuten Libre Office pyörittämiseen suositellaan 512MB.

Koska opinnäytetyön tarkoituksena on tarjota verkontestaustyökalu, täytyi käyttökokemus optimoida niin, että ohjelma toimii myös mahdollisimman vanhalla raudalla pyörivissä laitteissa. Tämä on erityisen tärkeää ympäristöissä, joissa laitteiden uusiminen tulisi erittäin kalliiksi

4.2 Graafinen käyttöliittymä

Graafinen käyttöliittymä toteutettiin PyQt-kirjastolla. PyQt valittiin siksi, että se on yhteensopiva graafiin käytetyn kirjaston kanssa. Käyttöliittymän layout koostuu neljästä osasta. Ohjelmassa on pystysuuntainen päälayout, johon on sisällytetty kolme ruudukkolayouttia. Lisäksi luotiin kaksi *QGroupBox*:ia palvelinta ja pääteohjelmaa varten, jotka sisällytin horisontaalisiin layoutteihin. (ks. Liite 2 osa 1)

Kuviossa 5 nähdään ohjelman graafisen käyttöliittymän ulkoasu. Graafi on ylimpänä, ja suoraan graafin alla on palvelin, eli Server laatikko. Alimpana on pääteohjelman osio, joka on myös asetettu laatikkoon.



KUVIO 5. Käyttöliittymän ulkoasu

Pääteohjelman graafisen käyttöliittymän elementit on luotu käyttämällä for-silmukkaa, joka on asetettu käymään läpi kolme iteraatiota. Yhden iteraation aikana luodaan jokainen elementti vuorotellen, jotka lisätään omiin kirjastoihinsa. (ks. Liite 2 osa 2)

Tämän jälkeen käydään läpi toinen for-silmukka, joka lisää käyttöliittymän pääteohjelmalle luotuun ruudukkolayouttiin riveille *i* ja *i+1* (ohjelmassa käytetään tässä silmukassa arvoja 1,3 ja 5) jokaisesta elementeille luodusta kirjastosta

järjestysluvulla *x* (Määritelty alussa luvuksi 0 ja kasvaa jokaisella iteraatiolla yhdellä) löytyvän elementin. (ks. Liite 2 osa 3)

Sitten otetaan kaikki luodut layoutit ja lisätään ne päälayoutiin siinä järjestyksessä, missä ne tulevat käyttöliittymää avatessa näkymään. (ks. Liite 2 osa 4)

Koska graafisessa käyttöliittymässä on haluttu yhdistää jokaisen pääteohjelman liukusäädin niiden oikealla puolella olevaan tekstikenttään, täytyy jokaiselle komponentille luoda oma signaalinsa. Tämä toteutetaan luomalla ensin for-silmukalla kaksi funktiota (*slidechanged()* ja *textchanged())* ja lisäämällä molemmat omaan kirjastoonsa (*funcdic1{}* ja *funcdic2{}*), minkä jälkeen luodaan signaalit, jotka kutsuvat kyseisiä funktioita. (ks. Liite 2 osa 5)

Application()-luokan alle on luotu kaksi *threading.Event()*-muuttujaa palvelimen ja pääteohjelman sammuttamista varten. Näitä muuttujia kutsutaan graafisen käyttöliittymän *stop* –napeista, sekä silloin, kun käyttöliittymä halutaan sammuttaa. Tämä takaa että säikeet eivät ole enää toiminnassa ohjelman sulkeutuessa säästäen käyttäjän useilta virheviesteiltä, jotka johtuvat taustaprosessien äkillisestä sammumisesta. (ks. Liite 2 osa 6)

4.3 Komentorivi

Koska verkontestaustyökalun tulee toimia etenkin palvelinympäristössä, toteutettiin ohjelma niin, ettei sen käyttäminen vaadi lainkaan graafista käyttöliittymää. Komentoriviltä pystyy toteuttamaan samat toiminnot kuin graafisella käyttöliittymällä graafia lukuun ottamatta. Graafin sijaan komentolinja näyttää liikenteen määrän pelkästään numeraalisilla arvoilla.

Komentorivi on toteutettu käyttämällä A*rgparse*-moduulia, jolla voi helposti ja yksinkertaisesti luoda ohjetekstiruudun ja toteuttaa optioiden käytön. Ohjelma tulostaa ohjetekstin, jos yhtään argumenttia ei ole annettu. (ks. Liite 3 osa 1)

Ennen komentorivin luontia ohjelma käy läpi funktion *valid_interfaces()*, joka etsii automaattisesti kaikki laitteen rajapinnat joihin on konfiguroitu pätevä IPv6 –osoite. (ks. Liite 3 osa 2)

Komentoriville on koodattu erillinen palvelin ja client komponentti kuin graafiselle käyttöliittymälle. Jos graafisen käyttöliittymän optiota ei ole annettu, mutta joko palvelimen tai pääteohjelman on, käynnistää ohjelma halutun komponentin uuteen säikeeseen. Tässä tapauksessa myös liikennelaskuri avataan uuteen säikeeseen samalla, kun pääsäie jää responsiiviseksi ohjelman sulkemista varten. (ks. Liite 3 osa 3)

4.4 Palvelin

UDP ja TCP palvelin socket

Palvelin on työn kannalta välttämättömyys. Se mahdollistaa pakettien vastaanottamisen ja niiden takaisin lähettämisen. Palvelimia on karkeasti jaoteltuna kahta tyyppiä, eli ne toimivat joko UDP tai TCP protokollalla. UDP on näistä protokollista kevyempi ja "tyhmempi", eikä se tarjoa samaa suojaa liikenteen perillepääsyn varmentamiseksi kuin TCP. Työn luonteen vuoksi TCP:n ominaisuudet ovat tarpeettomia ja jopa haitallisia, joten UDP on tätä käyttötarkoitusta varten ehdottomasti parempi.

UDP – palvelin

UDP-palvelimen koodi löytyy liitten 4 osasta 1

UDP palvelin eroaa TCP-vastineestaan sen yksinkertaisuuden takia. Se ei vaadi uuden socketin luomista jokaista sisään tulevaa yhteyttä varten, joten se voi toimia yhdessä säikeessä ja silti vastata kaikkiin sisään tuleviin paketteihin.

socketia luodessa määritellään että se toimii IPv6:lla ja että sen kuljetustason protokolla on UDP. Socket asetetaan myös tilaan, jossa uuden socketin voi sitoa heti

vanhan mentyä TIME_WAIT tilaan käyttämällä SO_REUSEADDR optiota. Vielä ennen socketin sitomista ennalta määriteltyyn porttiin ja IPv6 -osoitteeseen asetetaan sen aikakatkaisuksi 0.1 sekuntia jotta palvelin pysyy responsiivisena käyttöliittymän komentoihin.

TCP – palvelin

TCP-palvelimen koodi löytyy liitten 4 osasta 2

TCP-palvelin saattaa paketit UDP palvelinta luotettavammin perille, mutta se ei sovellu kyseiseen käyttötarkoitukseen siksi, että se lähettää aika-ajoin ylimääräisiä varmistuspaketteja. Tämä aiheuttaa sen, että liikenteen määrää on hankala säädellä tarkasti. Koodista näkee että TCP-pohjainen palvelin on UDP-vastinettaan huomattavasti monimutkaisempi sillä TCP-palvelimen toiminta vaatii uuden säikeen luomisen jokaista asiakasta varten.

Koodissa määritellään palvelimen IPv6 -osoite, jonka ohjelma osaa automaattisesti tunnistaa. Portti on määritelty koodiin käsin, eikä sitä voi graafista käyttöliittymää käytettäessä muuttaa.

Socketia luodessa määritellään sen kerros kolmen protokollaksi IPv6 ja kerros neljän protokollaksi UDP. Socket asetetaan myös tilaan, jossa se voi luoda uuden socketin heti vanhan sulkeuduttua.

Socket pysyy TCP-palvelimessa estävässä tilassa ja responsiivisuuden säilyttämiseksi käytetään *select()*-funktiota. *select()*-funktio toimii niin, että sitä kutsuttaessa ohjelma katsoo onko sillä antaa syötettä joka asetetaan määriteltyyn muuttujaan. Tämän jälkeen ohjelma jatkaa suoraan seuraavaan kohtaan, eikä jää odottamaan.

Jos *select()*-funktiolla on antaa uusi yhteys, ottaa ohjelma sen arvot ja käynnistää uuden säikeen jossa dataa vastaanotetaan ja lähetetään takaisin joko niin kauan kuin sitä saapuu, tai niin kauan että palvelin sammutetaan käyttäjän toimesta.

4.5 Pääteohjelma

Pääteohjelma on toteutettu samalla socket-tyypillä kuin palvelin, mutta muuten se eroaa siitä huomattavasti. Koska tarkoituksena on mahdollistaa useiden pääteohjelmien toiminta samanaikaisesti, mutta myös pystyä erottamaan nämä toisistaan, täytyy käyttää *threading.Local()* muuttujaa. Lisäksi jokaiselle pääteohjelmasäikeelle annetaan oma tunnusluku joka sisällytetään arvoon *cn*.

Pääteohjelman aloittamista varten on luotu for-silmukka, joka luo jokaiselle aiemmin luodulle painikkeelle oman *connect* signaalin. Tämä tarkoittaa sitä, että nappia painaessa lähetetään signaali, joka sisältää napin sijainnin *self.buttondic1[]* kirjastossa indeksinumeron muodossa. Signaali yhdistyy *connect*-funktioon, joka puolestaan ohjaa lähetetyn indeksinumeron säikeen aloittavalle funktiolle *self.threadcontrol().* (ks. Liite 5 osa 1)

self.threadcontrol()-funktio tunnistaa aloitettavan säikeen tyypin *threadtype* arvosta. Tässä tapauksessa arvo on 2, joten tiedetään, että halutaan aloittaa pääteohjelmasäie. (ks. Liite 5 osa 2)

Yhteyttä aloitettaessa tulee ottaa huomioon tilanteet, joissa se jostain syystä epäonnistuu. Syitä mahdolliselle epäonnistumiselle on useita. esimerkiksi annettu osoite voi olla väärä, tai jopa kokonaan eri muotoa kuin IPv6 -osoitteen kuuluu olla. Tästä syystä ohjelma tarkastaa jokaisen yhteyden virheiden varalta ja virheen sattuessa lähettää virhesignaalin ja sulkee säikeen. Virheviestit ovat:

'Invalid address'

Annettu osoite ei ole IPv6 -muotoa.

'No IP address given'

IP-osoitetta ei ole annettu lainkaan.

'Name or service not known'

Osoitetta yritetään selvittää nimen perusteella käyttäen DNS:ää, mutta annetulla nimellä ei löydy osoitetta.

'Connection failed'

Yhteyden luominen epäonnistui, eli palvelin ei joko ole käynnissä tai IPv6 -osoite ei ole palvelimen.

'Connection lost'

Yhteys palvelimeen katkesi

Pääteohjelman koodi löytyy liitten 5 osasta 3 ja *Oneround()*-funktion liitten 5 osasta 4

Pääteohjelma käyttää ajastinta takaamaan sen, että ohjelma lähettää oikean määrän paketteja sekunnissa. Ajastin luodaan käyttämällä *sched* kirjastoa. Sitä luodessa määritetään ajan määritys funktio, sekä viiveen määritys funktio, tässä tapauksessa käytetään *time.time* ja *time.sleep*-funktioita. Kun ajastin on luotu, voidaan siihen lisätä tapahtumia käyttämällä *enterabs()*-funktiota. Tapahtuman luominen vaatii seuraavan tapahtuman ajankohdan, prioriteetin, toimenpiteen sekä argumentit.

Ohjelma määrittelee ajastimen seuraavan tapahtuman hetken seuraavasti. *time.time() + 1.0 / (speed*128). speed* arvo saadaan graafisesta käyttöliittymästä, eli se on joku pyöreä arvo 1 ja 10 väliltä. Komentoriviä käytettäessä nopeudella ei ole rajoitteita, joten sitä käytettäessä myös desimaaliluvut kelpaavat. 128 on kiinteä luku, sillä paketit ovat kooltaan 1024 tavuisia (8192 bittisiä) ja jos sekunnissa lähetetään 128 tämän kokoista pakettia, on tulos 128*8192 = 1 048 576 bittiä, eli 1Mb. Lopullinen aika seuraavalle tapahtumalle tulee siis *speed* arvon ollessa 1 olemaan *time.time()* + 1/128.

Ajastimeen on asetettu toimenpiteeksi kutsua funktiota *oneround()*, joka sisältää paketin lähettämisestä vastaavan koodin. Funktio yrittää *try, except* lauseketta käyttämällä lähettää pakettia. Paketin lähetyksen onnistuessa käydään *try*:n alla

olevat komennot läpi, eli paketti lähetetään kohteeseensa, pääteohjelman *status*-arvo päivitetään, huomautetaan graafista käyttöliittymää pakettien lähetyksestä käyttämällä signaalia ja statuksen pääteohjelman statuksen mukaan joko luodaan uusi tapahtuma ajastimeen tai suljetaan socket ja huomautetaan tästä käyttöliittymää. Pääteohjelman *status*-arvo on oletuksena 1, mikä tarkoittaa että paketteja tulee lähettää. Jos pääteohjelma halutaan pysäyttää, voidaan *status*-arvo asettaa tilaan 0 painamalla toiminnassa olevan pääteohjelman *stop*-painiketta. Jos paketin lähetys ei onnistu, käy ohjelma *except*:n alla olevat komennot läpi, eli huomauttaa käyttöliittymää virheestä ja sulkee socketin.

4.6 Graafi

Ohjelma sisältää graafin, joka helpottaa liikenteen määrän hahmottamista. Ajattelin aluksi käyttää graafin luomiseen *matplotlib*-kirjastoa, mutta koska vaatimuksiin kuuluu graafin reaaliaikainen päivittyminen päädyin lopulta Qwt-kirjastoon. Qwt on myös yhteensopiva PyQt:n kanssa, mikä helpotti toteutusta huomattavasti.

Graafissa on x ja y akselit, jotka näyttävät aikaa ja liikenteen määrää. Y akselin mittayksikkönä on Mb ja x akselin mittayksikkönä sekunti. Graafissa on käytössä kolme kuvaajaa, jotka esittävät sisääntulevaa-, ulosmenevää- sekä kokonaisliikennettä. Lisäksi graafin alla näkyy näiden kolmen liikennetyypin määrä numeraalisessa muodossa

Graafi itsensä on luotu omaan luokkaansa nimeltä *PlotWidget()*. Luokka sisältää aloitusfunktion lisäksi neljä muuta funktiota (*setlandscape(), create_curves(), plot_data()* ja *netcalc()*), joista ensimmäisessä määritellään graafin tiedot, toisessa luodaan kuvaajat ja kiinnitetään ne graafiin, kolmannessa hoidetaan graafin reaaliaikainen skaalaus ja annetaan kuvaajille data, jonka mukaan ne piirretään graafin ruudukolle ja neljännessä mointoroidaan verkkorajapintoja ja haetaan niiltä dataa joka sitten esitetään graafilla. Aloitusfunktiossa puolestaan luodaan listat,

joihin *netcalc()*-funktio päivittää monitoroidun liikenteen määrän, kutsutaan muita funktioita ja yhdistetään *netcalc()*-funktiosta tuleviin signaaleihin (*plot = pyqtSignal()* ja *plotchanged = pyqtSignal(float, float, float)*), jotka luodaan heti luokan alla ennen *__init__()*-funktiota.

setlandscape()-funktiossa määritellään ensin taustan väri, oletuksena harmaa, valkoiseksi. Tämän jälkeen luodaan x- ja y akselit, nimetään ne ja määritetään niiden skaalat 60-0 ja 0-10. Sitten luodaan kuvaajia varten kuvatekstit ja asetetaan ne näkymään graafin alaosassa. Ohjelma osaa automaattisesti hakea kuvaajan nimen ja liittää sen kuvatekstiin, joten kuvaajat itsessään nimetään vasta seuraavassa funktiossa. Viimeiseksi luodaan graafin ruudukko ja liitetään se. (ks. Liite 6 osa 1)

create_curves()-funktiossa luodaan kuvaajat ja nimetään ne. Myös kuvaajan piirtämisessä käytetty väri ja tyyli valitaan tässä funktiossa. Lopuksi halutut kuvaajat liitetään graafiin. (ks. Liite 6 osa 2)

plot_data()-funktio antaa kuvaajille datan, jonka mukaan ne tulee piirtää. Tähän tarvitaan x- ja y-koordinaatit jokaiselle pisteelle jonka kautta kuvaajan tulee kulkea. Aluksi luodaan lista *xlist* x-koordinaatteja varten *self.outlist*:n perusteella. *self.outlist* on lista, johon lisätään joka sekunti uusi arvo ja jos arvoja on yli kuusikymmentä, poistetaan listan viimeinen arvo. *xlisti*:ä varten on luotu for-silmukka, joka jokaista *self.outlist*:sta löytyvää arvoa kohden käy läpi yhden iteraation. Jokaista läpikäytyä iteraatiota kohden kasvatetaan muuttujan n (aluksi 0) arvoa yhdellä ja lisätään listaan *xlist* arvo indeksinumerolla n ja arvolla n, eli n muuttujan ollessa 0, listaan lisätään indeksinumeroon 0 arvo 0. Kun muuttujan n arvoa kasvatetaan arvoon 1, lisätään indeksinumeroon 1 arvo 1. Tätä toistetaan niin kauan että on käyty läpi kaikki *self.outlist*:n sisältämät arvot. Kuvaajasta riippuen y-koordinaatteja varten käytetään jotakin seuraavista listoista: *self.outlist, self.inlist* tai *self.totallist.* Kaikille kuvaajille siis käytetään samaa x-koordinaattilistaa, mutta jokaisen kuvaajan y-arvo voi olla erilainen riippuen kuvaajan y-koordinaattilistan sisällöstä. *plot_data()*-funktion viimeinen toiminnallisuus on päivittää x-akselin skaalaa perustuen liikenteen

määrään. Alkuperäinen skaala on määritelty graafia luodessa olemaan 0-10, mutta liikenteen ylittäessä arvon 10, mutta ollessa myös alle 50, tunnistaa ohjelma sen ja nostaa skaalan 0-50. Jos taas liikenteen määrä nousee yli 50 Mb/s, nostetaan skaala välille 0-100. (ks. Liite 6 osa 3)

Funktio *netcalc()* sisältää liikennelaskurin, jolla mitataan laitteen rajapinnoista kulkevan liikenteen kokonaismäärä. Lisäksi *netcalc()* huolehtii graafin päivittämiseen käytetyn signaalin laukaisemisesta. Rajapintoja voidaan valvoa kahdella tavalla, joko avataan laitteen verkkoliikennedataa säilyttävät tiedostot manuaalisesti ja haetaan sieltä haluttu liikenne avainsanoja käyttäen, tai käytetään erillistä moduulia nimeltä *psutil* joka pystyy hakemaan verkkoliikenteen automaattisesti.

Linux säilyttää verkkoliikennedataa polussa /sys/class/net/ *iface* /statistics/ *t* _bytes, jossa *iface* tarkoittaa rajapintaa ja *t* liikenteen suuntaa. Ensimmäistä tapaa käytettäessä tuo tiedosto avataan jokaista rajapintaa kohden käyttämällä for-silmukkaa ja sieltä haetaan sen hetkinen siirretyn datan määrä, joka sisällytetään kirjastoon. Toimenpide toistetaan sekunnin päästä ja uuden ja vanhan datan erotuksesta saadaan sekunnin aikana rajapinnan läpi kulkeneen liikenteen määrä (ks. Liite 6 osa 4)

Ensimmäisen tavan etuna on se, että se ei vaadi ylimääräisen kirjaston asentamista, mutta on monimutkaisempi toteuttaa.

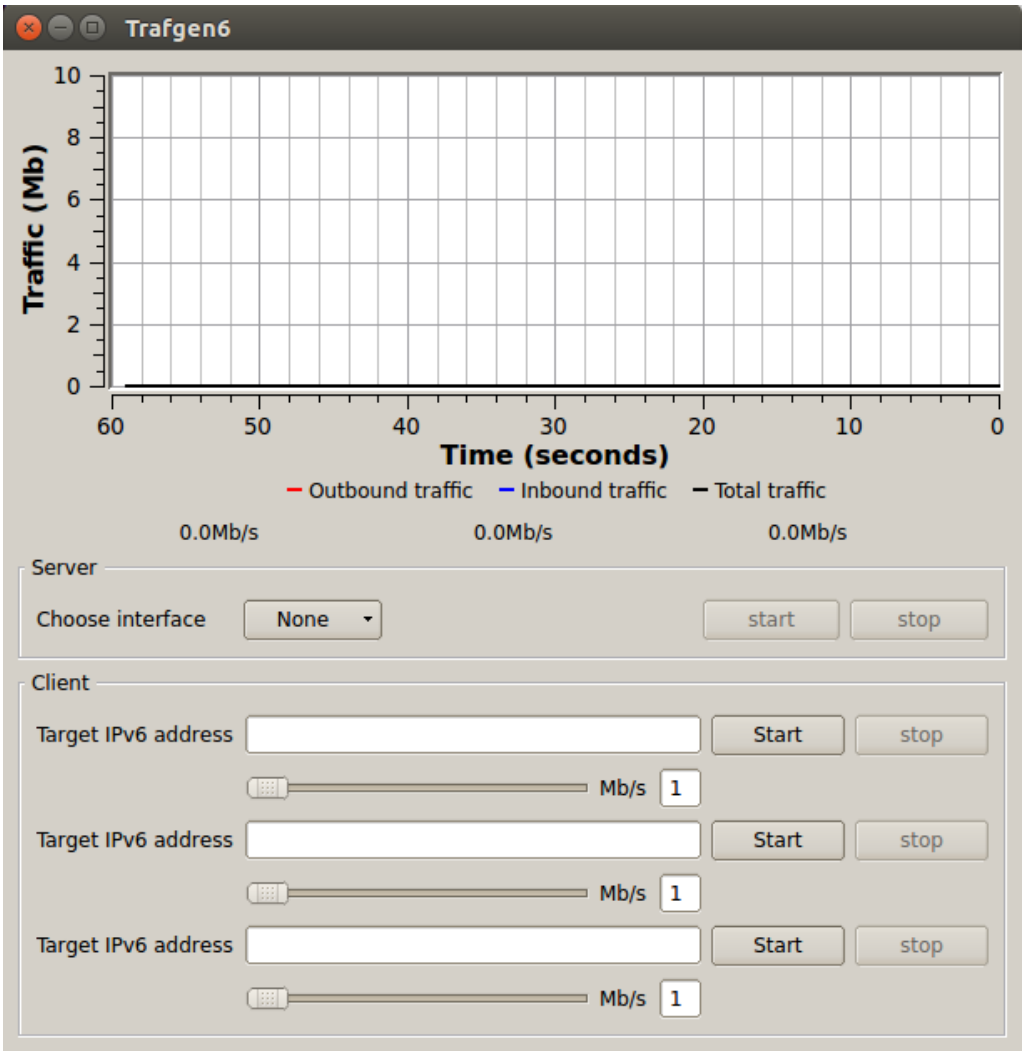
Toista tapaa käytettäessä ladataan *psutil*-kirjasto, jossa aiemmin käytetty toimenpide on sisäänrakennettuna. Komentoa 'counter = psutil.net_io_counters(pernic=True)[i]', jossa *i* on rajapinnan nimi, käyttäen voidaan hakea yhteen muuttujaan rajapinnan läpi lähetetyn ja saapuneen liikenteen määrä. Lähetetyn liikenteen arvo saadaan käyttämällä komentoa *counter.bytes_sent* ja saapuneen liikenteen vastaavasti käyttämällä komentoa *counter. bytes_recv*. Kun toimenpide suoritetaan uudestaan sekunnin jälkeen, saadaan sekunnin aikana kulkeneen liikenteen määrä vähentämällä ensimmäinen arvo toisesta. (ks. Liite 6 osa 5)

Koska liikenteen määrä täytyy esittää myös numeraalisena ja koska graafisen käyttöliittymän elementtejä ei pysty muokkaamaan pääsäikeen ja *Application()* luokan ulkopuolelta, täytyy liikenteen arvot lähettää jollakin tavalla pääsäikeelle. Tämä toteutetaan lähettämällä signaali *plotchanged = pyqtSignal(float, float, float)*, johon *Application()* luokan alla oleva self.*plot.plotchanged.connect(self.traffic_presenter)* yhdistyy. Luokka *PlotWidget()* on *Application()* luokkaa aloitettaessa *__init__()*-funktion alla sisällytetty muuttujaan *self.plot*, joten kaikki kyseisen luokan alla olevat signaalit löytyvät tästä muuttujasta. Lähetetty signaali sisältää kolme *float*-muuttuja, jotka signaaliin yhdistettäessä ohjataan funktiolle *self.traffic_presenter()*. Tämä funktio päivittää lähetetyt arvot graafiseen käyttöliittymään. (ks. Liite 6 osa 6)

5. Todennus

5.1 GUI:n todennus

Graafinen käyttöliittymä käynnistetään komennolla *Python trafgen6.py –g* joka aukaisee seuraavan näköisen ikkunan. Ikkuna sisältää graafin, johon liittyy myös liikenteen näyttäminen numeraalisissa arvoissa. Graafin alla on palvelinkomponentti jonka alta löytyy kolme pääteohjelmakomponenttia. Kuviossa 5 nähdään graafisen käyttöliittymän ulkoasu ilman muutoksia



KUVIO 6. GUI perusnäkymä

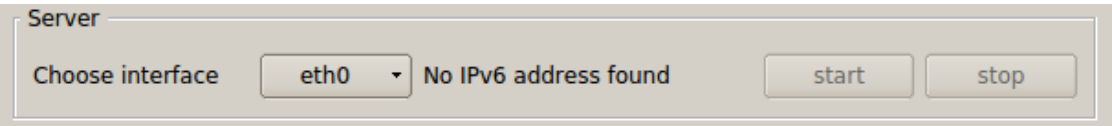
Palvelinkomponentin todennus

Graafisessa käyttöliittymässä palvelin on toteutettu siten, että se osaa automaattisesti hakea kaikki laitteen verkkorajapinnat silmukkaosoitetta lukuun ottamatta. Ohjelma osaa myös tarkistaa onko valittu rajapinta varustettu hyväksytyllä IPv6 -osoitteella. Seuraavissa kuvissa nähdään esimerkkejä mahdollisista tilanteista.

Kuviossa 7 avataan valikko tarjolla olevista rajapinnoista, mistä nähdään, että laitteella on käytössä rajapinnat *eth0* ja *eth1*. Tämä voidaan myös todentaa virtuaalikoneen asetusten *network* välilehdestä (ks Liite 8 kuvio 41) josta nähdään että vain *adapter 1* ja *adapter 2* ovat käytössä. Lisäksi komento *ifconfig –a* näyttää vain kaksi rajapintaa silmukkarajapinnan lisäksi. (ks Liite 8 kuvio 42)

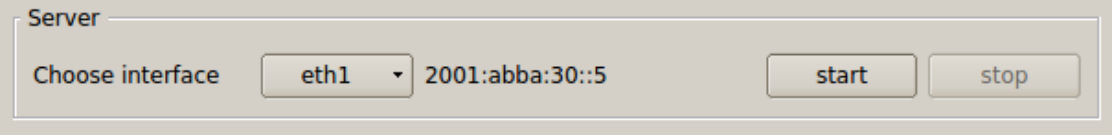
KUVIO 7. Palvelin - rajapinnan valintaruutu

Kuviossa 8 valitaan rajapinnaksi *eth0.* Koska kyseisellä rajapinnalla ei ole hyväksyttyä globaalia IPv6 -osoitetta, antaa ohjelma virheilmoituksen eikä aktivoi *start* -nappia.



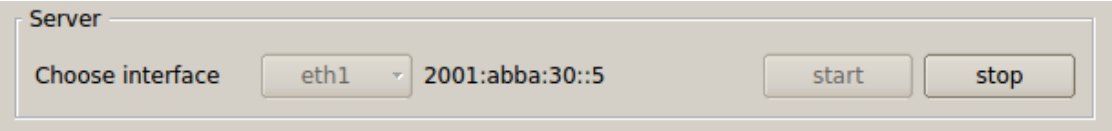
KUVIO 8. Palvelin, rajapinta eth0 – ei osoitetta

Seuraavaksi valitaan rajapinta *eth1.* Koska kyseiseen rajapintaan on konfiguroitu uniikki globaali IPv6 -osoite, näyttää ohjelma rajapinnan osoitteen ja antaa mahdollisuuden käynnistää palvelimen (ks. kuvio 9).



KUVIO 9 Palvelin, rajapinta eth1 - osoite löytyi

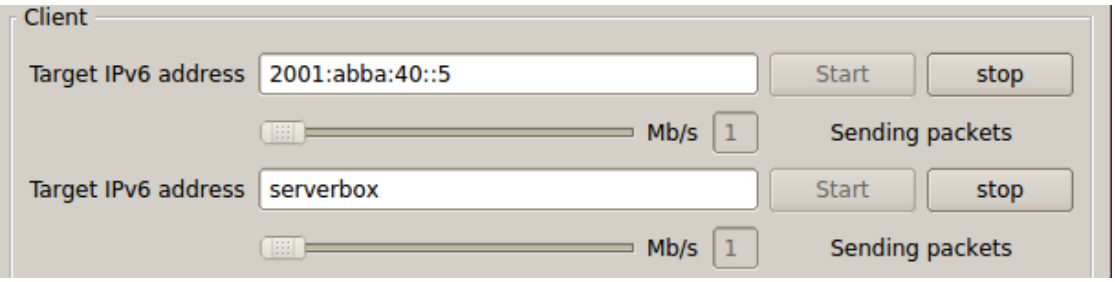
Start -napista painettaessa palvelin käynnistyy, *stop* -nappi aktivoituu ja rajapinnan valinta sekä *start* -nappi lukitetaan.



KUVIO 10 Palvelin käynnistetty

Pääteohjelmakomponentin todennus

Ohjelmassa on mahdollista hallinnoida kolmea pääteohjelmaa samanaikaisesti. Pääteohjelman käynnistys vaatii käynnissä olevan palvelimen IPv6 -osoitteen syöttämisen tekstikenttään. Ohjelma osaa käyttää DNS-hakua, joten IPv6 -osoitteen sijaan voidaan antaa myös kohteen nimi. Järjestelmään on asetettu kansioon */etc/hosts* rivi joka liittää osoitteen *2001:abba:40::5* nimeen *serverbox* (ks. Liite 8 kubio 43). Kuviossa 11 nähdään kuinka kahdella päätelaitteella lähetetään paketteja verkossa olevaan laitteeseen nimen ja IPv6 –osoitteen perusteella.

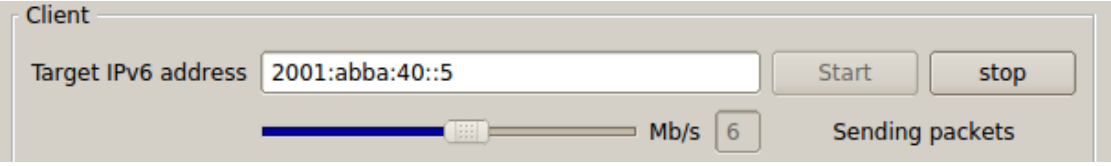


KUVIO 11. Pääteohjelma, pakettien lähetys IPv6 -osoitteella ja nimellä

Molempien pääteohjelmien tapauksessa pakettien lähetys onnistuu. *Start* nappi lukkiutuu ja *stop* nappi aktivoituu. Myös nopeuden säätö lukkiutuu pääteohjelman käynnistymisen jälkeen. Niin kauan, kun paketteja lähetetään, näkyy pääteohjelman alareunassa myös teksti ”Sending packets”.

Liikenteen lähettämisen nopeutta voidaan säätää joko liukusäätimestä vetämällä tai antamalla haluttu arvo tekstikenttään. Miniminopeus yhdelle pääteohjelmalle on 1Mb/s ja maksiminopeus 10Mb/s. Tekstikenttä hyväksyy vain numerot ja sitä käyttäessä tulee siinä valmiiksi oleva teksti ensin maalata, jotta annettu luku korvaa alla olevan. Tekstikenttä on nimittäin ohjelmoitu niin että siinä ei voi missään vaiheessa olla tyhjää arvoa.

Kuviossa 12 nähdään pääteohjelma lähettämässä paketteja säädetyllä nopeudella.



KUVIO 12. Pääteohjelman nopeuden säätö

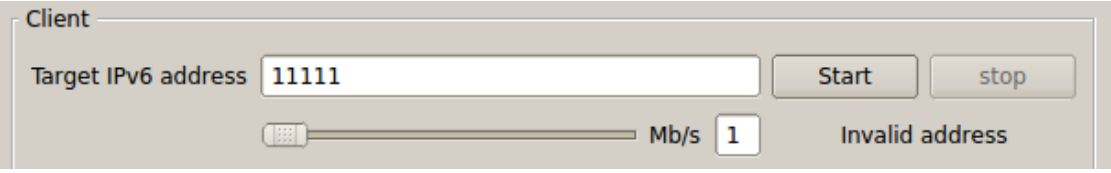
Pääteohjelman virheviestit

Koska pakettien lähetys saattaa epäonnistua useasta eri syystä, on pääteohjelmaan ohjelmoitu useita virheviestejä, jotka helpottavat epäonnistumisen syyn löytämistä.

Tekstikentän saattaa helposti jäädä tyhjäksi, jolloin ohjelma antaa virheviestin ”No IP address given” (ks. kuvio 13).

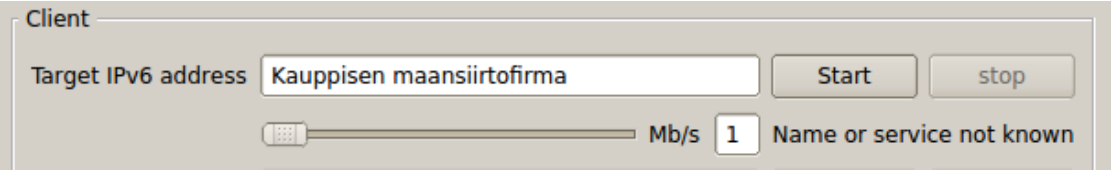
KUVIO 13. Virheviesti "No IP address given"

Pakettien lähetys saattaa myös epäonnistua siksi, että IPv6-osoite ei ole syystä tai toisesta pätevä. Tässä tapauksessa ohjelma antaa virheviestin ”Invalid address” (ks. kuvio 14).



KUVIO 14. Virheviesti "Invalid address"

Jos taas paketteja on yritetty lähettää nimen perusteella, mutta kyseistä nimeä ei löydy, antaa ohjelma virheviestin ”Name or service not known” (ks. kuvio 15).

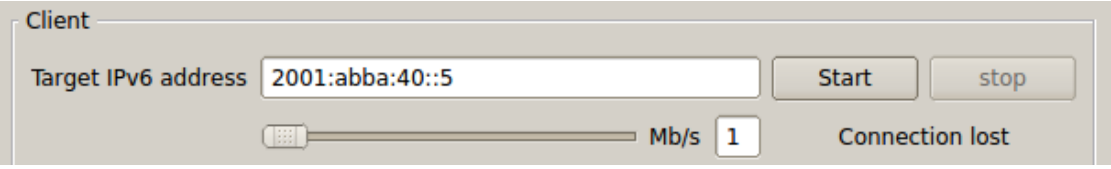


KUVIO 15. Virheviesti "Name or service not known"

Epäonnistuminen voi myös johtua siitä, että palvelin ei ole päällä, tai IPv6-osoite on väärä. Tällöin ohjelma antaa virheviestin ”Connection failed” (ks. kuvio 16).

KUVIO 16. Virheviesti "Connection failed"

Voi myös käydä niin, että kesken pakettien lähettämisen palvelin syystä tai toisesta sulkeutuu. Tässä tapauksessa ohjelma antaa virheviestin ”Connection lost” (ks. kuvio 17).



KUVIO 17. Virheviesti "Connection lost"

5.2 Komentolinjan todennus

Komentolinjan toimivuus voidaan testata komennolla *Python trafgen6.py*. Koska komento ei sisällä argumentteja, tulostaa ohjelma ohjetekstin, kuten Kuviosta 18 nähdään. Tulosteesta nähdään käytettävissä olevat komennot ja argumentit, joilla komennot voidaan suorittaa. Myös IPv6-osoitteella varustetut rajapinnat löytyvät automaattisesti tulosteesta, kuten Kuvioiden 18 ja 19 perusteella voidaan todeta.

```
root@nyoa-VirtualBox:/home/nyoa# python trafgen6.py
usage: trafgen6.py [-h] [-g | -s | -c] [-i {eth1} | -d TARGETIP] [-p PORT]
                   [-t TIME] [-r RATE]

Send Ipv6 traffic through your network

optional arguments:
  -h, --help            show this help message and exit
  -g, --gui             Start graphical user interface. Do not use other
                        arguments with this option
  -s, --server          Set the program to run in server mode
  -c, --client          Set the program to run in client mode
  -i {eth1}, --interface {eth1}
                        Specifies the interface for server
  -d TARGETIP, --destip TARGETIP
                        Sets the destination IPv6 address eg.2001:dead:beef::1
  -p PORT, --port PORT  Specifies the port used by server or
                        client | Default 5001
  -t TIME, --time TIME  Client only. Specifies in seconds how long traffic
                        will be sent | Default (0) is unlimited
  -r RATE, --rate RATE  Client only. Sets the rate at which traffic will be
                        sent (use plain numbers such as 1 or 1.5) | Default
                        1Mb/s
```

KUVIO 18. Komentolinja, ohjeteksti

Kuviota 19 ottaessa on myös rajapintaan *eth0* konfiguroitu IPv6-osoite, joten se näkyy ohjetekstin tulosteessa.

```
root@nyoa-VirtualBox:/home/nyoa# python trafgen6.py
usage: trafgen6.py [-h] [-g | -s | -c] [-i {eth1,eth0} | -d TARGETIP]
                   [-p PORT] [-t TIME] [-r RATE]
```

KUVIO 19. Komentolinja, ohjetekstin osoittamat rajapinnat

Komentolinja, palvelinkomponentin todennus

Komentolinjan palvelinkomponentti käynnistetään komennolla *Python trafgen6.py –s –i eth1*. Argumentiksi rajapinnalle eli *–i,* voi asettaa minkä tahansa ohjelman tarjoamista vaihtoehdoista. Kuviossa 20 nähdään komentolinjalta käynnistetty palvelinkomponentti johon lähetetään liikennettä pääteohjelmalta 10Mb/s. Palvelin tulostaa käynnistyessään rajapinnan IPv6-osoitteen, sekä portin jossa se toimii.

```
root@nyoa-VirtualBox:/home/nyoa# python trafgen6.py -s -i eth1
Server address:  2001:abba:30::5
Server port:  5001
Out 10.01Mb/s     In 10.02Mb/s     Total 20.02Mb/s
```

KUVIO 20. Komentolinja, palvelimen todennus

Optiolla *–p portti* voidaan vaihtaa porttia, jossa palvelin toimii. Tällöin täytyy kuitenkin muistaa asettaa sama portti pääteohjelmalle. Kuviossa 21 portiksi on asetettu 1337.

```
root@nyoa-VirtualBox:/home/nyoa# python trafgen6.py -s -i eth1 -p 1337
Server address:  2001:abba:30::5
Server port:  1337
Out 10.01Mb/s     In 10.02Mb/s     Total 20.03Mb/s
```

KUVIO 21. Komentolinja, palvelimen todennus vaihdetulla portilla

Komentolinja, pääteohjelmakomponentin todennus

Pääteohjelma käynnistetään komentolinjaa käyttäen komennolla *Python trafgen6.py –c –d kohdeosoite (-p portti –r nopeus -t aika)*. Kohdeosoitteeksi asetetaan käynnissä

olevan palvelimen IPv6-osoite. Jos palvelimen porttia on muutettu oletusportista, käytetään optiota *–p portti* oikean portin asettamiseksi. Kuviossa 22 nähdään pääteohjelma lähettämässä liikennettä palvelimelle nopeudella 7.5Mb/s.

```
root@nyoa-VirtualBox:/home/nyoa# python trafgen6.py -c -d 2001:abba:40::5 -r 7.5
Out 7.51Mb/s      In 7.51Mb/s      Total 15.02Mb/s
```

KUVIO 22. Komentolinja, pääteohjelman todennus

Kuviossa 23 pääteohjelma lähettää liikennettä palvelimelle jonka portti on oletusarvon sijaan 1337 nopeudella 10Mb/s.

```
root@nyoa-VirtualBox:/home/nyoa# python trafgen6.py -c -d 2001:abba:40::5 -r 10 -p 1337
Out 10.01Mb/s      In 9.99Mb/s      Total 20.0Mb/s
```

KUVIO 23. Komentolinja, pääteohjelman todennus vaihdetulla portilla

Pääteohjelma voidaan myös asettaa lähettämään liikennettä ennalta määritellyn ajan ennen sen automaattista sammumista käyttämällä optiota *–t aika*. Kuviossa 24 pääteohjelma on asetettu lähettämään liikennettä nopeudella 10Mb/s kymmenen sekunnin ajan.

```
root@nyoa-VirtualBox:/home/nyoa# python trafgen6.py -c -d 2001:abba:40::5 -r 10 -p 1337 -t 10
Out 9.99Mb/s      In 9.99Mb/s      Total 19.97Mb/s
Client socket closed after 10 seconds

Client stopped
root@nyoa-VirtualBox:/home/nyoa#
```

KUVIO 24. Komentolinja, pääteohjelman todennus aikarajalla

Komentolinja, pääteohjelman virheilmoitukset

Myös komentolinjalta käynnistettävä pääteohjelma on asetettu antamaan virheilmoituksia tilanteessa, jossa pakettien lähetys syystä tai toisesta epäonnistuu.

Jos IPv6-osoitetta ei ole annettu lainkaan, tulostaa ohjelma seuraavanlaisen virheviestin (ks. kuvio 25).

```
root@nyoa-VirtualBox:/home/nyoa# python trafgen6.py -c -d
usage: trafgen6.py [-h] [-g | -s | -c] [-i {eth1,eth0} | -d TARGETIP]
                   [-p PORT] [-t TIME] [-r RATE]
trafgen6.py: error: argument -d/--destip: expected one argument
```

KUVIO 25. Komentolinjan virheviestit, ei annettua osoitetta

Annettu osoite voi myös olla väärää muotoa, eli se ei ole pätevä IPv6 –osoite. Tällöin ohjelma tulostaa ”Invalid address” –virheilmoituksen (ks. kuvio 26).

```
root@nyoa-VirtualBox:/home/nyoa# python trafgen6.py -c -d 11111
Invalid address
```

KUVIO 26. Komentolinjan virheviestit, "invalid address"

Jos paketteja yritetään lähettää käyttämällä nimipalvelua, mutta käytettyä nimeä ei löydy, tulostaa ohjelma virheviestin ”Name or service unknown” (ks. kuvio 27).

```
root@nyoa-VirtualBox:/home/nyoa# python trafgen6.py -c -d kauppisen_maansiirtofirma
Name or service unknown

Client stopped
```

KUVIO 27. Komentolinjan virheviestit, "Name or service not known"

5.3 Verkkoinfrastruktuurin todennus

Verkkoinfrastruktuuri todennettiin JAMK:n SpiderNetissä käyttämällä reititin WG1-R1:stä sekä kytkimiä WG1-SW1 ja WG1-SW2. Palvelin ja pääteohjelma sijaitsivat erillisillä laitteilla, jotka olivat molemmat kytketty kytkimen WG1-SW2 portteihin erillisiin vlaneihin. Laitteiden osoitteet olivat seuraavat:

vastaanottava laite *2001:abba:30::5*

Lähettävä laite *2001:abba:40::5*

Reitittimen rajapintojen osoitteet puolestaan:

GI0/1.30 *2001:abba:30::1*

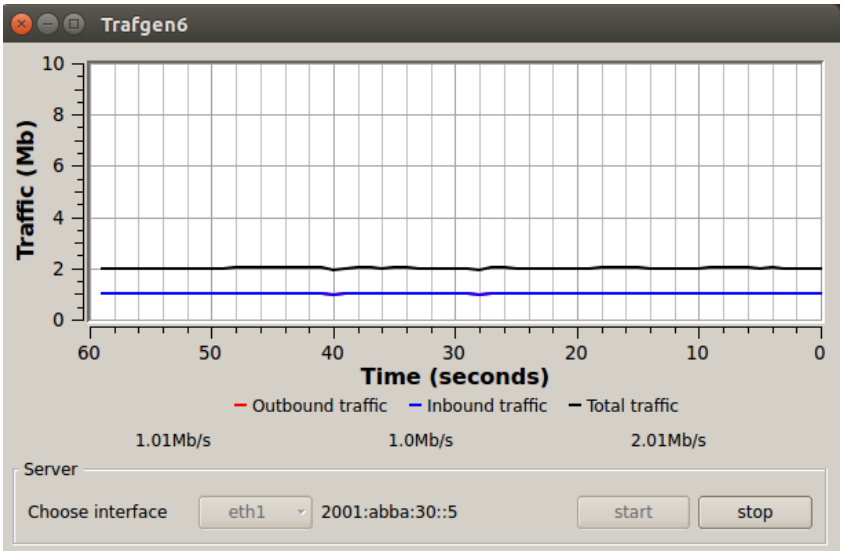
GI0/1.40 *2001:abba:40::1*

Kuviossa 28 nähdään traceroute vastaanottavalta laitteelta lähettävälle. Kuviosta voidaan todeta, että liikenne menee osoitteen *2001:abba:30::1* kautta. Kyseinen osoite on konfiguroitu reitittimen rajapintaan GI0/1.30, mikä tarkoittaa että liikenne kulkee reitittimen läpi, eli verkko toimii.

```
root@nyoa-VirtualBox:~# traceroute6 2001:abba:40::5
traceroute to 2001:abba:40::5 (2001:abba:40::5) from 2001:abba:30::5, 30 hops ma
x, 16 byte packets
 1  2001:abba:30::1 (2001:abba:30::1)  0.237 ms  0.868 ms  0.975 ms
 2  2001:abba:40::5 (2001:abba:40::5)  0.917 ms  0.371 ms  3.266 ms
```

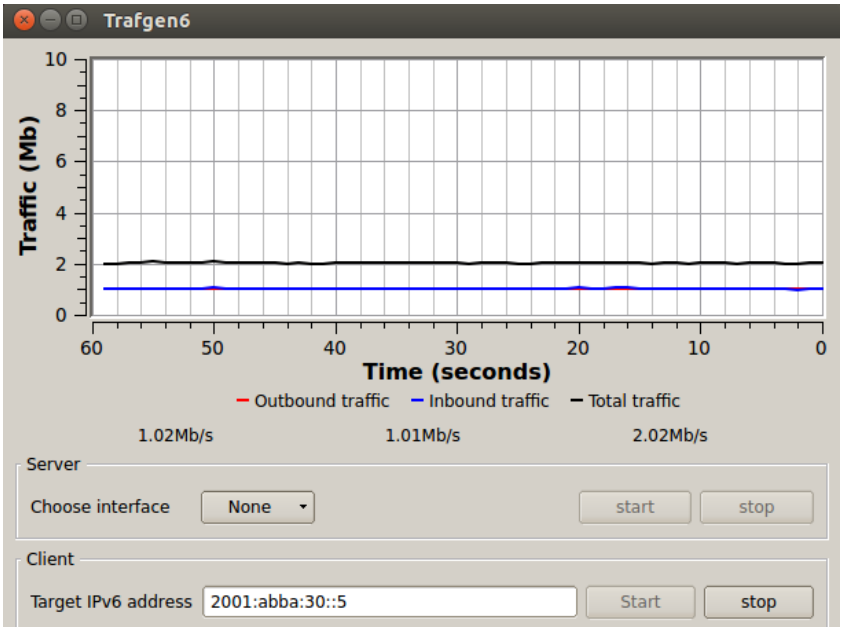
KUVIO 28. Traceroute palvelin -> pääteohjelma

Seuraavaksi käynnistetään palvelin ja aloitetaan liikenteen lähetys perusasetuksilla. Kuviosta 29 voidaan todeta, että palvelin toimii ja että se vastaanottaa liikennettä. Myös palvelimen rajapinta näkyy kuviosta (ks. kuvio 29).



KUVIO 29. Palvelin vastaanottamassa liikennettä

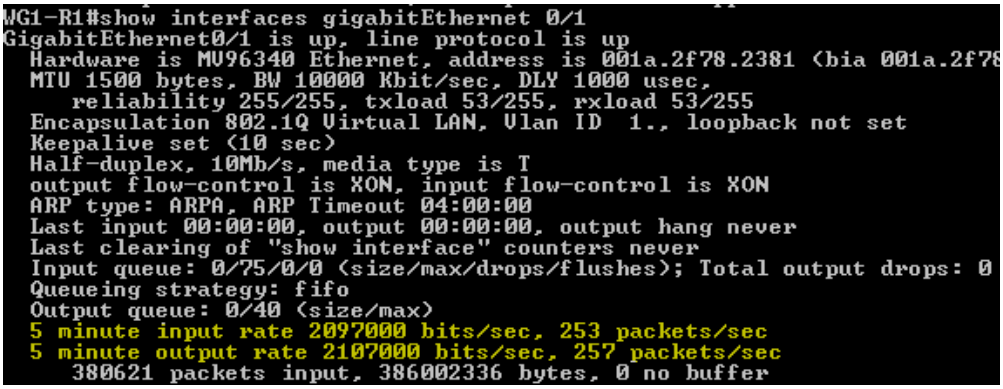
Pääteohjelma on tässä vaiheessa jo käynnistetty ja se lähettää liikennettä. Tämä voidaan todeta kuviosta 30, jossa näkyy liikenteen määrän lisäksi myös kohdeosoite.



KUVIO 30. Pääteohjelma lähettämässä liikennettä

Reitittimen läpi kulkeneen liikenteen määrä todennettiin käyttämällä komentoa *show interfaces gi0/1.* Komento näyttää paljon tietoa rajapinnasta, johon sisältyy myös viiden minuutin aikana rajapinnan läpi kulkeneiden pakettien summa. Koska tarkasteluajanjakso oli pitkä, otettiin varmuuden vuoksi todennukset useasta syötteestä monen minuutin väliajoin.

Kuvioissa 31, 32 ja 33 näkyy reitittimelle saapunut ja siltä lähtenyt liikenne (input/output). Kuviota 31 otettaessa liikennettä on lähetetty noin kahdeksan minuutin ajan. Tuloksia tarkasteltaessa tulee ottaa huomioon, että sisääntulevaan liikenteeseen lasketaan lähetetyn liikenteen lisäksi myös palvelimen kaikuna takaisin lähettämä liikenne, mikä tuplaa lähetetyn liikenteen määrän. Sama pätee myös saapuvaan liikenteeseen.



KUVIO 31. Liikenteen todennus reitittimellä 1

Kuviossa 32 on seuraava otos, joka on otettu joitakin minuutteja kuvion 31 tilanteen jälkeen. Kuviosta voidaan todeta, että pakettien määrä on pysynyt samana, mutta rajapintojen läpi kulkeneen datan määrä on kasvanut.

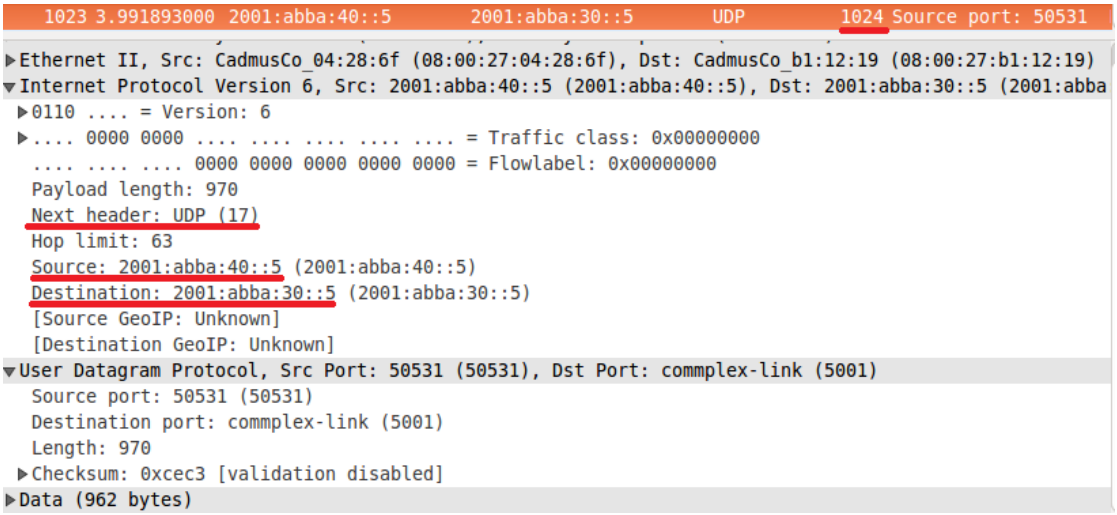
KUVIO 32. Liikenteen todennus reitittimellä 2

Kuviossa 33 nähdään tilanne muutaman minuutin päästä. Paketteja on kulkenut ulos reitittimeltä sama määrä kuin aiemmin, mutta sisääntulleiden pakettien määrä on laskenut yhdellä. Datan määrä on pysynyt ulospäin kulkevan liikenteen kohdalla ennallaan, mutta sisäänpäin tulevan datan määrä on laskenut hieman.

KUVIO 33. Liikenteen todennus reitittimellä 3

Lähetin lähettää paketteja noin 128 paketin sekuntivauhtia. Koska kuvioissa tämä liikenne näkyy kaksinkertaisena molempiin suuntiin, täytyy tuo luku kertoa kahdella eli 128 * 2 = 256. Kuvista voidaan kuitenkin todeta, että sisään on tullut sekunnin aikana 252-253 pakettia ja ulos on lähtenyt 257 pakettia, mikä on hyvin lähellä haluttua määrää. Keskimääräinen sisään tulleen paketin koko saadaan jakamalla bittimäärä/s pakettimäärällä/s, eli (2097000 + 2110000 + 2102000) : (253 + 253 + 252) = 8323b. Tulos on siis hieman suurempi kuin laskettu lähetetyn paketin koko joka on 1024 * 8 = 8192b. Lähetetyn paketin koko on samaa laskukaavaa käyttämällä hieman suurempi (8198b), joka on kuitenkin äärimmäisen lähellä oletettua paketin kokoa.

Kuviossa 34 nähdään paketin koko ja sen koostumus. Siitä voidaan todeta, että paketti on lähetetty käyttämällä IPv6:sta ja että sen kuljetustason protokolla on UDP. Paketin koko tavuina (1024) sekä IPv6-otsikon lähde- ja kohdeosoitekentät löytyvät kuviosta punaisella alleviivattuina.



KUVIO 34. IPv6 paketti – wireshark

5.4 Lähettimen todennus

Lähettimen testausta varten luotiin uusi funktio, joka mittaa lähettimen sekunnin aikana käymien iteraatioiden määrän. Koska jokaisen iteraation aikana lähetetään

yksi paketti, voidaan tulosten perusteella päätellä kuinka monta pakettia sekunnin aikana on lähetetty.

Funktiota kutsutaan uudessa säikeessä, jotta pakettien lähettämisestä vastaava funktio pystyy toimimaan samaan aikaan laskurin kanssa samalla jatkuvasti päivittäen laskurissa käytettävää *self.nn*-muuttujaa.

```
def trouble(self):
    while True:
        old = self.nn
        time.sleep(1)
        print self.nn - old, "Iterations/s"
```

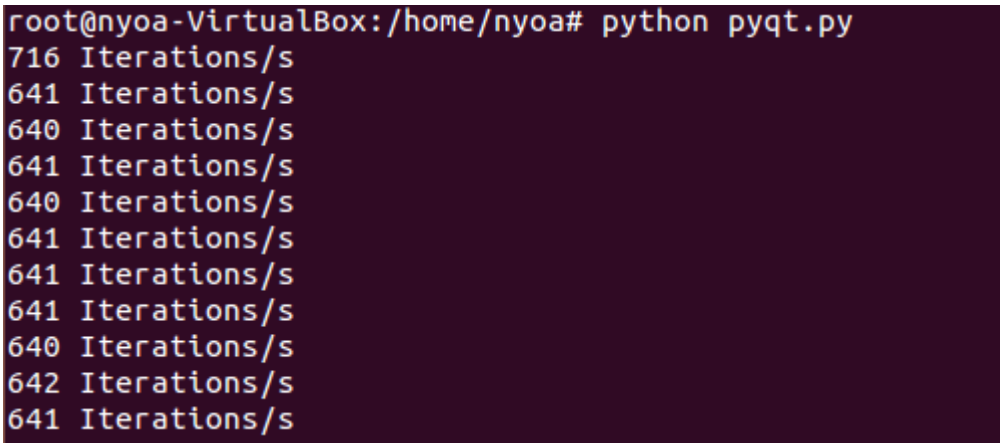
Funktio ottaa jokaisen kierroksen alussa sen hetkisen *self.nn*-arvon, joka määrittyy paketteja lähettävän funktion tekemien iteraatioiden mukaan. Tämän jälkeen funktio odottaa sekunnin ennen uuden *self.nn*-arvon ja juuri määritellyn *old*-arvon erotuksen, jonka loppuun on lisätty tekstinpätkän *”Iterations/s”,* tulostamista. Näin saadaan ohjelma tulostamaan sekunnin välein sinä aikana aikana tapahtuneiden iteraatioiden summa.

Ensimmäiseksi liikennegeneraattori asetettiin lähettämään paketteja 1Mb/s nopeudella VyOS- käyttöjärjestelmällä toimivan virtuaalisen reitittimen läpi. Tulokseksi saatiin oletetusti noin 128 iteraatiota sekunnissa (ks. kuvio 35).

```
root@nyoa-VirtualBox:/home/nyoa# python pyqt.py
127 Iterations/s
129 Iterations/s
128 Iterations/s
128 Iterations/s
128 Iterations/s
```

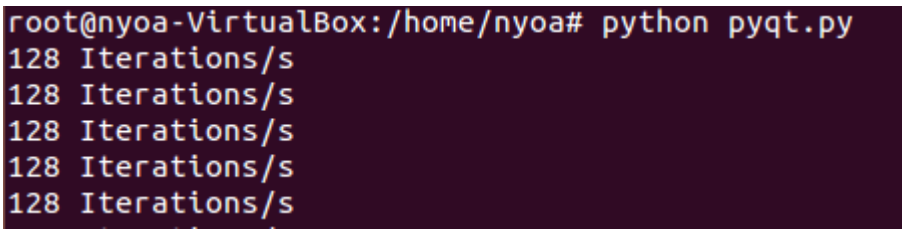
KUVIO 35. Lähettimen testaus VyOS:n läpi 1Mb/s

Sama testi toistettiin nopeudella 5Mb/s. Tulokseksi saatiin ensimmäisen sekunnin aikana tapahtuneita iteraatioita lukuun ottamatta noin 640 iteraatiota sekunnissa. Tulos on siis lähes täsmälleen viisinkertainen aiempaan testiin verrattuna (ks. kuvio 36).



KUVIO 36. Lähettimen testaus VyOS:n läpi 5Mb/s

Seuraavaksi testattiin iteraatioiden määrä silmukkaliikennettä lähettäessä. Liikennegeneraattori asetettiin ensin lähettämään liikennettä laitteen silmukkaosoitteeseen nopeudella 1Mb/s. Tulokseksi saatiin sama kuin VyOS:n läpi lähetettäessä, eli 128 iteraatiota sekunnissa (ks. kuvio 37).



KUVIO 37. Lähettimen testaus silmukkaosoitteeseen lähetettäessä 1Mb/s

Myös tämä testi toistettiin nopeudella 5Mb/s. Tulokset olivat myös tällä kertaa lähes identtiset VyOS-testin kanssa, eli noin 640 iteraatiota sekunnissa (ks. kuvio 38).

```
root@nyoa-VirtualBox:/home/nyoa# python pyqt.py
641 Iterations/s
640 Iterations/s
641 Iterations/s
640 Iterations/s
640 Iterations/s
```

KUVIO 38. Lähettimen testaus silmukkaosoitteeseen lähetettäessä 5Mb/s

Tuloksista voidaan päätellä että lähetin toimii halutulla tavalla ja että sen läpi käymien iteraatioiden määrä kasvaa halutussa suhteessa liikenteen määrään. Lähetin myös toimii samalla tavalla riippumatta siitä, mihin paketteja halutaan lähettää. Nämä todennukset olivat erityisen tärkeitä siksi, että silmukkaosoitteeseen lähetettäessä liikenteen määrä näkyy kaksinkertaisena mittausohjelmissa, vaikka iteraatioiden määrä ei muutu osoitetyypin muuttuessa.

5.5 Liikennelaskurin todennus

Liikennelaskurin todentaminen vaati vertailua ulkoisen, yleisesti toimivaksi todetun, laskurin kanssa. Tällaiseksi ulkoiseksi laskuriksi valittiin Linux-jakeluissa yleisesti käytetyn verkkorajapintojen valvontatyökalun Iftop. Iftop kykenee mittaamaan laitteen läpi kulkevaa liikennettä rajapintakohtaisesti ja näyttää sen komentorivillä kätevästi numeraalisessa muodossa.

Iftop käyttää liikenteen määrää laskiessaan hieman eri algoritmia kuin liikennegeneraattori. Liikennegeneraattori näyttää sekunnin välein kulkeneen liikenteen määrän, kun taas Iftop laskee noin viiden sekunnin keskiarvon. Lisäksi Iftop näyttää liikenteen positiivisen ja negatiivisen huippuarvon.

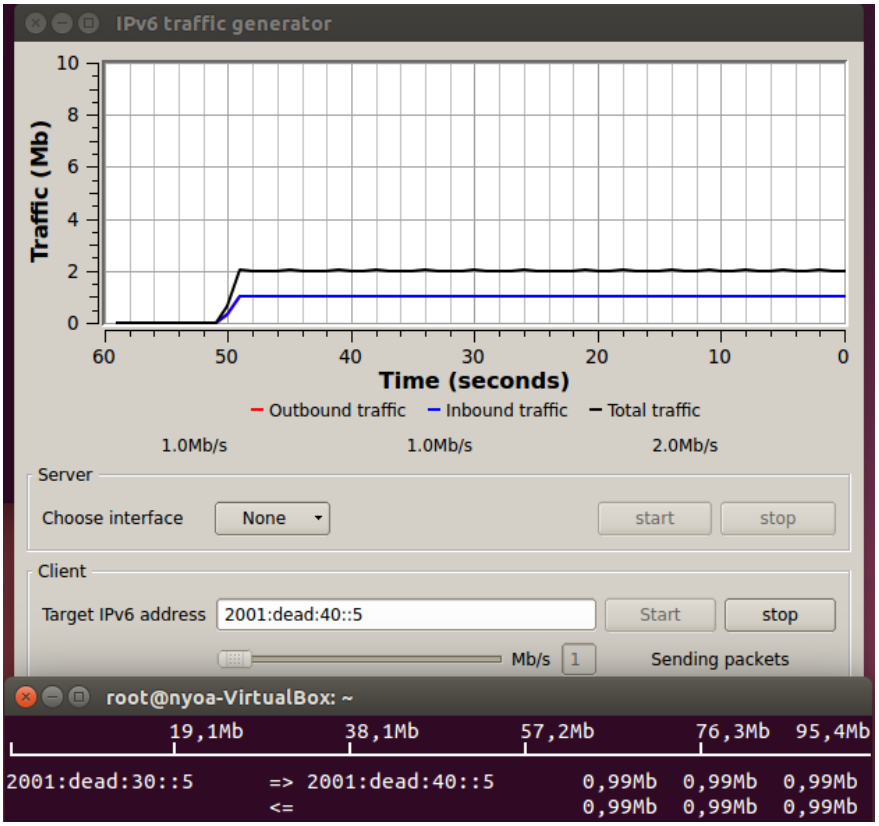
Iftop käynnistetään kirjoittamalla Linux-komentoriville *iftop,* mutta mittausta varten täytyi vielä määritellä mitattava rajapinta komennolla:

iftop –i eth1

Tämä käynnistää Iftop-ohjelman ja asettaa sen mittaamaan rajapintaa *eth1*, joka on kytkettynä VyOS:n

Vastaanottava laite, eli palvelin, sijaitsi osoitteessa 2001:dead:40::5, mikä voidaan todeta kuviosta 39 joko pakettigeneraattorin pääteohjelman tekstikentästä, tai Iftop-ohjelmasta. Lähettävän laitteen osoite oli puolestaan 2001:dead:30::5, mikä voidaan todentaa Iftop-ohjelmasta.

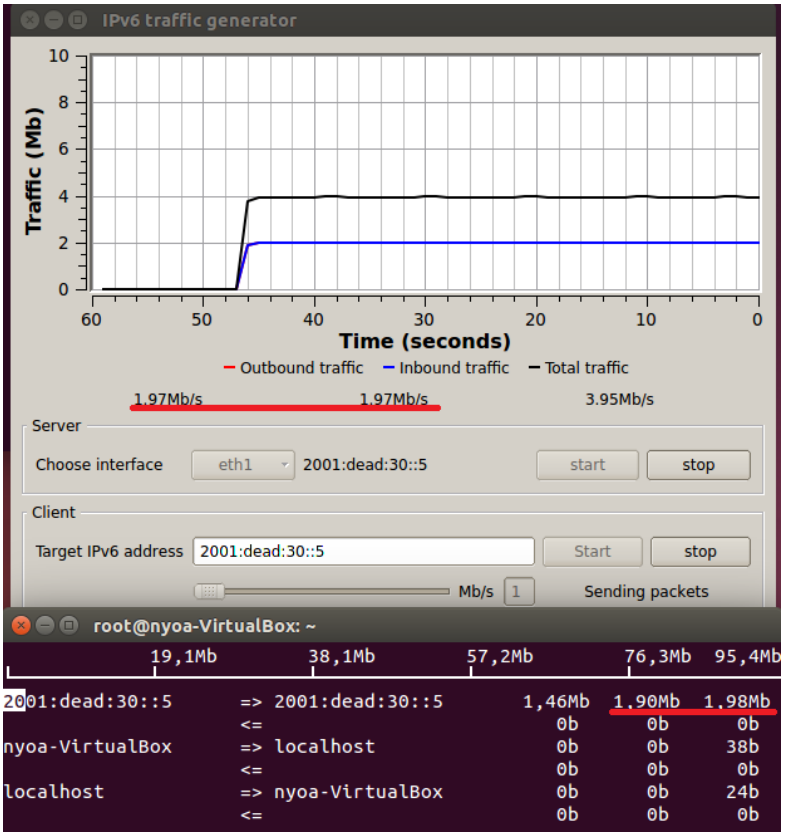
Pakettigeneraattori asetettiin lähettämään liikennettä nopeudella 1Mb/s osoitteeseen 2001:dead:40::5 ja graafista voidaan todeta, että kuvion ottohetkellä mittausta on kulunut noin viisikymmentä sekuntia. Pakettigeneraattorin laskuri näyttää liikenteen määräksi 1.0Mb/s ja Iftop 0.99Mb/s eli tulokset ovat lähes identtiset.



KUVIO 39. Liikennelaskurin vertaaminen ulkoisen ohjelman kanssa - VyOS

Seuraavassa suoritetussa mittauksessa liikennettä lähettiin laitteelle itselleen silmukkalähetyksenä (ks. kuvio 40). Koska liikenne ei koskaan poistunut laitteelta, täytyi Iftop avata kuuntelemaan *eth1*-rajapinnan sijaan lo-rajapintaa. Tämä suoritetaan komennolla *iftop –i lo.*

Kuviossa 40 on alleviivattu mittauksen kannalta oleelliset kohdat punaisella kynällä. Kuviota tarkasteltaessa voidaan todeta, että ohjelmalla käynnistetyn palvelimen osoite on 2001:dead:30::5 ja lähettäjän osoite 2001:dead:30::5, eli kyseessä on silmukkalähetys. Lähetyksen nopeudeksi on säädetty 1Mb/s, mutta niin liikennegeneraattorin laskuri, kuin ulkoinen ohjelma Iftop, näyttävät lähetetyn liikenteen määräksi noin 2Mb/s. Koska niin Iftop kuin liikennegeneraattori näyttivät liikenteen määrän yhtäläisesti kaksinkertaisena, voidaan olettaa, että tämä johtuu joko Linuxin tavasta merkitä liikennettä silmukkalähetystä lähettäessä, tai siitä että kyseessä on virtuaaliympäristössä ilman reititintä toteutettu mittaus.



KUVIO 40. Liikennelaskurin vertaaminen ulkoisen ohjelman kanssa - Silmukkalähetys

6. Pohdinta

Opinnäytetyön tavoitteena oli toteuttaa IPv6:lla toimiva liikennegeneraattori graafisella käyttöliittymällä ja graafilla. Mielestäni tavoitteissa onnistuttiin erittäin hyvin, varsinkin jos ottaa huomioon opinnäytetyön suorittajan lähtötilanteen, eli lähes nollakokemuksen koodaamisesta.

Vähäisen koodauskokemuksen takia useat osa alueet täytyi tehdä uudestaan käyttäen eri kirjastoja. Suurimpana esimerkkinä käyttöliittymän vaihto TkInter-kirjastosta PyQt4:ään, koska Qwt –kirjasto on yhteensopiva vain PyQt:n kanssa.

Pakettien kokoa määriteltäessä ei otettu huomioon paketille tapahtuvia muutoksia sen kulkiessa verkon läpi (mm.dot1q leimaus), mikä lisää paketin kokoa ja täten kasvattaa verkossa kulkevan liikenteen määrää verrattuna laitteilla näkyvään liikenteen määrään. Erittäin suureksi harmiksi todennuksia tehdessä huomattiin, että syystä tai toisesta dot1q leimattuja paketteja ei ollut mahdollista todentaa, sillä mittauksessa käytettävän laitteen verkkokortin ajureiden takia laite poisti dot1q leimat ennen kuin Wireshark pääsi niihin käsiksi. Paketteihin olisi mahdollista lisätä dot1q leima laitteelta lähtiessä, mikä poistaisi ongelman, mutta se vaatisi paketin rakentamisen alusta alkaen itse.

Ohjelman rakenteeseen jouduttiin tekemään useita pienehköjä muutoksia ongelmien ilmentyessä testauksen aikana, vielä aivan viime metreihin saakka. Palvelin oli esimerkiksi yritetty konfiguroida liian monimutkaisesti, joten se ei toiminut riittävän hyvin. Ratkaisu oli äärimmäisen yksinkertainen, eli non-blocking *socketin* sijaan käytettiinkin blocking (estävää) *socketia,* mutta niin, että siihen oli asetettu aikaraja käyttämällä *socket.settimeout()*-funktiota. Lisäksi pääteohjelman nopeuden määrittävät tekstikentät täytyi asettaa päivittymään vasta käyttäjän fokuksen poistuessa tekstikentästä, ei tekstin muuttuessa. Graafiseen käyttöliittymään täytyi luoda myös funktio, joka lähettää sulkeutumispyynnön käynnissä oleville säikeille, jos ohjelma suljetaan äkillisesti.

Loppujen lopuksi graafisesta käyttöliittymästä tuli siisti ja hyvin jaoteltu. Graafi näyttää asialliselta ja sen skaalautuvuus helpottaa liikenteen määrän hahmottamista huomattavasti. Lisäksi verkkoinfrastruktuuri toimii monisäikeistämisestä johtuvia ongelmia lukuun ottamatta erittäin hyvin. Komentolinjan elementit toimivat täysin moitteitta.

Lähteet

Addressing, Protocol Families and Socket Types. 2010. Pymotw.com Viitattu 5.4.2015. http://pymotw.com/2/socket/addressing.html

Beal. V. 2014. What is The Difference Between IPv6 and IPv4? Webodepia.com. Viitattu 2.4.2015. http://www.webopedia.com/DidYouKnow/Internet/ipv6_ipv4_difference.html

Sager. I. 2012. Before IPhone and Android Came Simon, the First Smartphone. Bloomberg.com. Viitattu 28.5.2015 http://www.bloomberg.com/bw/articles/2012-06-29/before-iphone-and-android-came-simon-the-first-smartphone

Boddie. D. 2014. About PyQt. Python.org. Viitattu 19.4.2015 https://wiki.Python.org/moin/PyQt

Donzé. F. 2004. IPv6 Autoconfiguration. Cisco.com. Viitattu 2.4.2015. http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_7-2/ipv6_autoconfig.html

IEEE-USA White Paper, Next Generation Internet: IPv4 Address Exhaustion, Mitigation Strategies and Implications for the U.S. 2009. Viitattu 15.4.2015. https://www.ieeeusa.org/policy/whitepapers/IEEEUSAWP-IPv62009.pdf

IETF RFC 3513. 2003. Internet Engineering Task Forcen määrittelemä standardi, Internet Protocol Version 6 (IPv6) Addressing Architecture. Viitattu 2.4.2015 ftp://ftp.rfc-editor.org/in-notes/rfc3513.txt

IETF RFC 768. 1980. Internet Engineering Task Forcen määrittelemä standardi, User Datagram Protocol . Viitattu 10.4.2015. http://tools.ietf.org/html/rfc768

IETF RFC 2460. 1998. Internet Engineering Task Forcen määrittelemä standardi, Internet Protocol Version 6 (IPv6) Addressing Specification. Viitattu 2.4.2015. https://tools.ietf.org/html/rfc2460

IETF RFC 3041. 2001. Internet Engineering Task Forcen määrittelemä standardi, Privacy Extensions for Stateless Address Autoconfiguration in IPv6. Viitattu 2.4.2015. https://tools.ietf.org/html/rfc3041

Ignatchenko. S. 2010. Single-Threading: Back to the Future? accu.org. Viitattu 15.4.2015 http://accu.org/index.php/journals/1634

IPv4 exhaustion details. Apnic.net. Viitattu 2.4.2015. https://www.apnic.net/community/ipv4-exhaustion/ipv4-exhaustion-details

JAMK SpiderNet. 2009. Lyhyt esittely JAMK:n SpiderNetin taustasta ja resursseista. Viitattu 1.5.2015. http://student.labranet.jamk.fi/spidernet/

Thomas. J. How to Simplify Shorten and Compress IPv6 Addresses. OmniSecu.com. Viitattu 5.4.2015. http://www.omnisecu.com/tcpip/ipv6/how-to-simplify-ipv6-addresses.php

Jenkov. J. 2015. Java Concurrency, Part 2 - Multithreading Benefits. Jenkov.com. Viitattu 22.4.2015 http://tutorials.jenkov.com/java-concurrency/benefits.html

Just how many IPv6 addresses are there? Really? 2012. Rednectar.net. Viitattu 2.4.2015. http://rednectar.net/2012/05/24/just-how-many-ipv6-addresses-are-there-really/

Koivisto. M. TCP- ja UDP-protokollat. Internetix.fi. Viitattu 10.4.2015 http://oppimateriaalit.internetix.fi/fi/avoimet/6tekniikkatalous/verkko/tcp_ja_udp_protokollat

Mecki. 2013. Socket options SO_REUSEADDR and SO_REUSEPORT, how do they differ? Do they mean the same across all major operating systems? Stackoverflow.com. Viitattu 18.4.2015. http://stackoverflow.com/questions/14388706/socket-options-so-reuseaddr-and-so-reuseport-how-do-they-differ-do-they-mean-t

Miksi JAMKiin? 2015. Jyväskylän ammattikorkeakoulun verkkosivut. Viitattu 1.5.2015 http://www.jamk.fi/fi/Koulutus/Miksi-JAMKiin/

Pierre. 2011. IPv4 Exhaustion and IPv6 Readiness - should you care, and if so, why? IP-performance.com. Viitattu 20.3.2015. http://www.ip-performance.co.uk/blog/ipv6readiness

Rathmann. U. 2014. Qwt User's Guide, Qwt - Qt Widgets for Technical Applications. sourceforge.net. Viitattu 6.5.2015. http://qwt.sourceforge.net/

Should I use Python 2 or Python 3 for my development activity? 2014. Wiki.Python.org. Viitattu 11.5.2015. https://wiki.Python.org/moin/Python2orPython3

Threading. 2015. Python.org. Viitattu 15.4.2015 https://docs.Python.org/2/library/threading.html

Liitteet

Liite 1. IPv6 Otsikko

Taulukko 7 IPv6-otsikko

<table>
<tr><td>bitit</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td>0</td><td colspan="4">Versio</td><td colspan="8">Liikenneluokka</td><td colspan="20">Virtaetiketti</td></tr>
<tr><td>32</td><td colspan="16">Hyötykuorman pituus</td><td colspan="8">Seuraava otsikko</td><td colspan="8">Hyppyrajoitus</td></tr>
<tr><td>64</td><td colspan="32" rowspan="4">Lähdeosoite</td></tr>
<tr><td>96</td></tr>
<tr><td>128</td></tr>
<tr><td>160</td></tr>
<tr><td>192</td><td colspan="32" rowspan="4">Kohdeosoite</td></tr>
<tr><td>224</td></tr>
<tr><td>256</td></tr>
<tr><td>288</td></tr>
</table>

Liite 2. GUI:n koodi

Osa 1

```
self.mainLayout = QVBoxLayout(self.mainWidget)
self.gLayout1 = QGridLayout()
self.gLayout2 = QGridLayout()
self.gLayout2.setColumnMinimumWidth(1, 15)
self.gLayout2.setColumnMinimumWidth(3, 180)
self.gLayout3 = QGridLayout()
self.gLayout3.setColumnMinimumWidth(4, 10)
self.gLayout3.setColumnMinimumWidth(2, 200)
self.serverBox = QGroupBox('Server')
self.clientBox = QGroupBox('Client')
```

Osa 2

```
for i in range (0,3):
  self.label1 = QLabel('Target IPv6 address')
  self.label5 = QLabel('Mb/s', self)

  self.label4 = QLabel('Connection failed', self)
  self.label4.setAlignment(Qt.AlignCenter)
  self.label4.hide()

  self.entry1 = QLineEdit(self)

  self.entry2 = QLineEdit(self)
```

```
        self.entry2.setMaxLength(2)
        self.entry2.setText("1")

        self.button1 = QPushButton('Start', self)

        self.button4 = QPushButton('stop', self)
        self.button4.setEnabled(False)

        self.slider1 = QSlider(self)
        self.slider1.setOrientation(Qt.Horizontal)
        self.slider1.setMinimum(1)
        self.slider1.setMaximum(10)
        self.slider1.setSingleStep(1)

        self.labeldic1.update({i:self.label1})
        self.labeldic4.update({i:self.label4})
        self.labeldic5.update({i:self.label5})
        self.entrydic1.update({i:self.entry1})
        self.entrydic2.update({i:self.entry2})
        self.buttondic1.update({i:self.button1})
        self.buttondic2.update({i:self.button4})
        self.sliderdic1.update({i:self.slider1})
```

Osa 3

```
    rows = [1,3,5]
    x = 0
```

```
for i in rows:
 self.gLayout3.addWidget(self.labeldic1[x],i,0)
 self.gLayout3.addWidget(self.labeldic5[x],i+1,3)
 self.gLayout3.addWidget(self.labeldic4[x],i+1,5,1,2)
 self.gLayout3.addWidget(self.entrydic1[x],i,1,1,4)
 self.gLayout3.addWidget(self.entrydic2[x],i+1,4)
 self.gLayout3.addWidget(self.buttondic1[x],i,5)
 self.gLayout3.addWidget(self.buttondic2[x],i,6)
 self.gLayout3.addWidget(self.sliderdic1[x],i+1,1,1,2)
 x += 1
```

Osa 4

```
self.mainLayout.addWidget(self.plot)
self.mainLayout.addLayout(self.gLayout1)
self.serverBox.setLayout(self.gLayout2)
self.mainLayout.addWidget(self.serverBox)
self.clientBox.setLayout(self.gLayout3)
self.mainLayout.addWidget(self.clientBox)
```

Osa 5

```
funcdic1 = {}
for x in range(0,3):
 def slidechanged(x=x):
    svalue = str(self.sliderdic1[x].value())
    self.entrydic2[x].setText(svalue)
 funcdic1.update({x:slidechanged})
```

```
funcdic2 = {}
for x in range(0,3):
  def textchanged(x=x):
    try:
      value = int(self.entrydic2[x].text())
      if int(value) > 0 and int(value) <= 10:
         self.sliderdic1[x].setValue(value)
      else:
         self.entrydic2[x].setText('10')
         self.sliderdic1[x].setValue(10)
    except:
         self.entrydic2[x].setText('1')
         self.sliderdic1[x].setValue(1)
  funcdic2.update({x:textchanged})

for i in range(0,3):
  self.buttondic1[i].clicked.connect(partial(self.threadcontrol, '2',$
  self.buttondic2[i].clicked.connect(partial(self.clientstop, '0', i))

self.sliderdic1[0].valueChanged.connect(lambda: funcdic1[0]())
self.sliderdic1[1].valueChanged.connect(lambda: funcdic1[1]())
self.sliderdic1[2].valueChanged.connect(lambda: funcdic1[2]())
self.entrydic2[0].editingFinished.connect(lambda: funcdic2[0]())
self.entrydic2[1].editingFinished.connect(lambda: funcdic2[1]())
self.entrydic2[2].editingFinished.connect(lambda: funcdic2[2]())
```

Liite 3. Komentorivin koodi

Osa 1

```
parser = argparse.ArgumentParser(description='Send Ipv6 traffic through your network')
group1 = parser.add_mutually_exclusive_group()
group2 = parser.add_mutually_exclusive_group()
mode = 0
list = []
nohelp = (False)
valid_interfaces()

group1.add_argument('-g', '--gui',
        action='store_true', dest='gui',
        help='Start graphical user interface. Do not use other arguments with this option')
group1.add_argument('-s', '--server',
        action='store_true', dest='server',
        help='Set the program to run in server mode')
group1.add_argument('-c', '--client',
        action='store_true', dest='client',
        help='Set the program to run in client mode')
group2.add_argument('-i', '--interface',
        action='store', dest='interface', choices=list,
        help='Specifies the interface for server')
group2.add_argument('-d', '--destip',
        action='store', dest='targetip',
        help='Sets the destination IPv6 address eg.2001:dead:beef::1')
parser.add_argument('-p', '--port',
        action='store', dest='port', type=int,
        help='Specifies the port used by server or client | Default 5001', default=5001)
parser.add_argument('-t', '--time',
        action='store', dest='time', type=float,
```

```
            help='Client only. Specifies in seconds how long traffic will be sent | Default (0) is unlimited', default=0)
    parser.add_argument('-r', '--rate',
            action='store', dest='rate', type=float,
            help='Client only. Sets the rate at which traffic will be sent (use plain numbers such as 1 or 1.5) | Default 1Mb/s', default=1)


    if len(sys.argv)==1:
        parser.print_help()
        sys.exit(1)
```

Osa 2

```
    def valid_interfaces():
        for i in ni.interfaces():
            try:
                ip = ni.ifaddresses(i)[ni.AF_INET6][0]['addr']
                if not ip.startswith('fe80') and not i.startswith('lo'):
                    list.insert(0,i)
            except KeyError:
                print 'Interface \"' + i + '\" does not have an IP address'
```

Osa 3

```
    # This part starts the GUI if the flag is set


    if args.gui:
        app = QApplication(sys.argv)
        app.setStyle('cleanlooks')
        mainwindow = Application()
        mainwindow.show()
        cr = app.exec_()
        mainwindow.exit()
        sys.exit(cr)
```

```
    # If the flag for server is set on and an interface is given the program starts in server
mode


    if args.server and args.interface:
        thread1 = Server(host=args.interface, port=args.port)
        thread1.start()
        mode = 1


    # If the flag for client is set on and a target ip address is given the program starts in cli-
ent mode


    if args.client and args.targetip:
        thread1 = Client(args.targetip, args.port, args.rate, args.time)
        thread1.start()
        mode = 2


    if mode != 0 and thread1.isAlive():
        thread2 = calculator()
        try:
            thread2.start()
        except:
            nohelp = (True)


    if threading.active_count() > 2:
        while threading.active_count > 2:
            try:
                time.sleep(0.01)
            except KeyboardInterrupt:
                thread1.join()
                if mode == 1:
                    print '\n' + 'Server stopped'
                elif mode == 2:
```

```
                print '\n' + 'Client stopped'
            thread2.join()
            sys.exit()
    elif nohelp == False:
        parser.print_help()
        sys.exit(1)
```

Liite 4. Palvelimen koodi

Osa 1

```
    def serverstart(self):
     self.stopserver.clear()
     self.buttonswitch("1",'null')
     self.host = self.intip
     self.port = 5001
     s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
     s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
     s.settimeout(0.01)
     s.bind((self.host, self.port))

     while not self.stopserver.isSet():
         try:
             message, address = s.recvfrom(1500)
             s.sendto(message, address)
         except:
             pass
     s.close()
     self.buttonswitch("0", 'null')
```

Osa 2

```
    def serverstart(self):
     self.buttonswitch("1")
```

```
 self.host = self.intip
 self.port = 5001
 s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
 s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
 s.bind((self.host, self.port))

 inputs = [ s ]

 s.listen(5)
 self.serverstatus = "1"
 while s:
     inputready, outputready, exceptready = select.select(inputs,[], [])
     if inputready == inputs and self.serverstatus == "1":
         c, addr = s.accept()
         ct = threading.Thread(target=self.client_thread, args=[c, addr])
         ct.start()
     if self.serverstatus == "0":
         break
 s.close()
 self.buttonswitch("0")

def client_thread(self, c, addr):
 print "Connection from: " + str(addr)
 while True:
     data = c.recv(1500)
     if not data:
         break
     c.send(data)
     if self.serverstatus == "0":
         break
 c.close()
```

Liite 5. Päätelaitteen koodi

Osa 1

```
for i in range(0,3):
  self.buttondic1[i].clicked.connect(partial(self.threadcontrol, '2', i))
  self.buttondic2[i].clicked.connect(partial(self.clientstop, '0', i))
```

Osa 2

```
def threadcontrol(self, threadtype, cn):
 if threadtype == "1":
      self.thread1 = threading.Thread(target = self.serverstart)
      self.thread1.setDaemon(True)
      self.thread1.start()
 elif threadtype == "2":
      self.thread2 = threading.Thread(target = self.clientstart, args=[cn])
      self.thread2.setDaemon(True)
      self.thread2.start()
```

Osa 3

```
def clientstart(self, cn):
 self.threadLocal.cn = cn

 self.threadLocal.sch = sched.scheduler(time.time, time.sleep)
 self.threadLocal.next_time = time.time()
 self.threadLocal.sch.enterabs(self.threadLocal.next_time, 0, self.oneround, ())
 self.reset.emit(cn)
```

```
time.sleep(0.03)
targetip = self.entrydic1[cn].text()
host = targetip
port = 5001
speed = int(self.entrydic2[cn].text())
self.threadLocal.delay = 1.0 / (speed*128)
n = bool(False)
err = 0

openfile = open('1024')
self.threadLocal.message = openfile.read()

if targetip:
    self.threadLocal.s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
    self.threadLocal.s.setblocking(0)
    try:
        self.threadLocal.s.connect((host,port))
        self.threadLocal.s.send(self.threadLocal.message)
        time.sleep(0.1)
        if len(self.threadLocal.s.recvfrom(1500)) > 1:
            n = (True)
    except IOError as (errno, strerror):
        if errno == -9:
            err = 1
            self.err1.emit(1, self.threadLocal.cn)
```

```
                if errno == 11:
                    err = 1
                    self.err1.emit(5, self.threadLocal.cn)
                if errno == -2:
                    err = 1
                    self.err1.emit(4, self.threadLocal.cn)
            self.clientstatus.update({self.threadLocal.cn:"1"})
            self.buttonswitch("2", cn)
            self.threadLocal.counter = 0
            if n == True:
                self.count = 0
                self.threadLocal.sch.run()
            elif err == 0:
                self.err1.emit(5, self.threadLocal.cn)

            self.threadLocal.s.close()
            self.buttonswitch("3", cn)
        else:
            self.err1.emit(3, self.threadLocal.cn)
```

Osa 4

```
    def oneround(self):
        if not self.stopclient.isSet():
            try:
                self.threadLocal.s.send(self.threadLocal.message)
```

```
            status = self.clientstatus[self.threadLocal.cn]
            self.err1.emit(2, self.threadLocal.cn)
            self.threadLocal.counter += 1
            if status == '1':
                self.threadLocal.next_time += self.threadLocal.delay
                self.threadLocal.sch.enterabs(self.threadLocal.next_tim$
            else:
                self.err1.emit(0, self.threadLocal.cn)
                self.threadLocal.s.close()
        except IOError as (errno, strerror):
            if errno == 111:
                self.err1.emit(6, self.threadLocal.cn)
                self.threadLocal.s.close()
```

Liite 6. Graafin koodi

Osa 1

```
    def setlandscape(self):
        self.setCanvasBackground(Qt.white)
        self.plotLayout().setAlignCanvasToScales(True)

        self.setAxisTitle(Qwt.QwtPlot.yLeft, 'Traffic (Mb)')
        self.setAxisTitle(Qwt.QwtPlot.xBottom, 'Time (seconds)')
        self.setAxisScale(Qwt.QwtPlot.yLeft, 0., 10.)
        self.setAxisScale(Qwt.QwtPlot.xBottom, 60., 0.)
```

```
self.legend = Qwt.QwtLegend()
self.legend.setFrameStyle(QFrame.Sunken)
self.legend.setItemMode(Qwt.QwtLegend.ClickableItem)
self.insertLegend(self.legend, Qwt.QwtPlot.BottomLegend)

self.grid = Qwt.QwtPlotGrid()
self.grid.enableXMin(True)
self.grid.setMajPen(QPen(QPen(Qt.gray)))
self.grid.setMinPen(QPen(QPen(Qt.lightGray)))
self.grid.attach(self)
```

Osa 2

```
def create_curves(self):
self.curve1 = Qwt.QwtPlotCurve('Outbound traffic')
self.curve1.setPen(QPen(Qt.red, 2))
self.curve1.setStyle(Qwt.QwtPlotCurve.Lines)
self.curve1.attach(self)
self.curve1.setRenderHint(Qwt.QwtPlotItem.RenderAntialiased)

self.curve2 = Qwt.QwtPlotCurve('Inbound traffic')
self.curve2.setPen(QPen(Qt.blue, 2))
self.curve2.setStyle(Qwt.QwtPlotCurve.Lines)
self.curve2.attach(self)
self.curve2.setRenderHint(Qwt.QwtPlotItem.RenderAntialiased)
```

```
self.curve3 = Qwt.QwtPlotCurve('Total traffic')
self.curve3.setPen(QPen(Qt.black, 2))
self.curve3.setStyle(Qwt.QwtPlotCurve.Lines)
self.curve3.attach(self)
self.curve3.setRenderHint(Qwt.QwtPlotItem.RenderAntialiased)
```

Osa 3

```
xlist = []
n = 0
for x in self.outlist:
    xlist.insert(n,n)
    n += 1
self.curve1.setData(xlist, self.outlist)
self.curve2.setData(xlist, self.inlist)
self.curve3.setData(xlist, self.totallist)
if max(self.totallist) > 10 and max(self.totallist) < 50:
    self.setAxisScale(Qwt.QwtPlot.yLeft, 0., 50.)
elif max(self.totallist) > 50:
    self.setAxisScale(Qwt.QwtPlot.yLeft, 0., 100.)
else:
    self.setAxisScale(Qwt.QwtPlot.yLeft, 0., 10.)
self.replot()
```

Osa 4

```
def netcalc(self):
 while True:
```

```
txdat = {}

rxdat = {}

n = 1

for i in ni.interfaces():

    data = self.get_bytes('tx', i)

    txdat.update({n:data})

    print txdat

    n += 1

n = 1

for i in ni.interfaces():

    data = self.get_bytes('rx', i)

    rxdat.update({n:data})

    print rxdat

    n += 1

n = 1

time.sleep(1)

for i in ni.interfaces():

    data = self.get_bytes('tx',i)

    totalsent = data - txdat[n]

    print totalsent

    n += 1

n = 1

for i in ni.interfaces():

    data = self.get_bytes('rx',i)

    totalrecv = data - rxdat[n]
```

```
            print totalrecv
            n += 1


    def get_bytes(self, t, iface):
     f = open('/sys/class/net/' + iface + '/statistics/' + t + '_bytes', 'r')
     data = f.read()
     return int(data)
```

Osa 5

```
    def netcalc(self):
     while True:
         sent = {}
         recv = {}
         n = 1
         for i in ni.interfaces():
             counter = psutil.net_io_counters(pernic=True)[i]
             sent.update({n:counter.bytes_sent})
             recv.update({n:counter.bytes_recv})
             n +=1
         time.sleep(1)
         n = 1
         for i in ni.interfaces():
             counter = psutil.net_io_counters(pernic=True)[i]
             totalsent = counter.bytes_sent - sent[n]
             totalrecv = counter.bytes_recv - recv[n]
             alldata = totalsent + totalrecv
```

```
            self.totalout = (totalsent/mb) * 8
            self.totalin = (totalrecv/mb) * 8
            self.totalinout = (alldata/mb) * 8
            inwards.insert((n-1), float("%.2f" % self.totalin))
            out.insert((n-1), float("%.2f" % self.totalout))
            total.insert((n-1), float("%.2f" % self.totalinout))
            n += 1
        self.inlist.insert(0, sum(inwards))
        self.outlist.insert(0, sum(out))
        self.totallist.insert(0, sum(total))
        if len(self.outlist) > 60:
            self.inlist.pop()
            self.outlist.pop()
            self.totallist.pop()
        self.plotchanged.emit(self.inlist[0], self.outlist[0], self. totallist[0])
        self.plot.emit()
```

Osa 6

```
    def traffic_presenter(self, inlist, outlist, totallist):


     self.traffic1.setText(str(outlist) + 'Mb/s')
     self.traffic2.setText(str(inlist) + 'Mb/s')
     self.traffic3.setText(str(totallist) + 'Mb/s')
```

Liite 7. Testausverkon konfiguraatiot

WG1-R1

```
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname WG1-R1
!
boot-start-marker
boot-end-marker
!
logging message-counter syslog
!
no aaa new-model
memory-size iomem 5
!
dot11 syslog
ip source-route
!
!
ip cef
!
!
ipv6 unicast-routing
ipv6 cef
!
multilink bundle-name authenticated
!
voice-card 0
!
archive
log config
 hidekeys
!
interface GigabitEthernet0/0
no ip address
shutdown
duplex auto
speed auto
!
interface GigabitEthernet0/1
no ip address
duplex auto
speed auto
ipv6 enable
no shut
!
interface GigabitEthernet0/1.30
encapsulation dot1Q 30
```

```
ipv6 address 2001:ABBA:30::1/64
!
interface GigabitEthernet0/1.40
encapsulation dot1Q 40
ipv6 address 2001:ABBA:40::1/64
!
interface GigabitEthernet0/1.90
encapsulation dot1Q 90
ipv6 address 2001:ABBA:90::1/64
!
interface Serial0/0/0
no ip address
shutdown
clock rate 2000000
!
interface Serial0/0/1
no ip address
shutdown
!
interface FastEthernet0/1/0
no ip address
shutdown
duplex auto
speed auto
!
interface FastEthernet0/1/1
no ip address
shutdown
duplex auto
speed auto
!
ip forward-protocol nd
no ip http server
no ip http secure-server
!
control-plane
!
line con 0
line aux 0
line vty 0 4
login
!
scheduler allocate 20000 1000
end
```

WG1-SW1

```
!
```

```
version 12.2
no service pad
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname WG1-SW1
!
!
no aaa new-model
ip subnet-zero
!
spanning-tree mode pvst
spanning-tree extend system-id
!
vlan internal allocation policy ascending
!
interface GigabitEthernet0/1
switchport trunk encapsulation dot1q
switchport mode trunk
!
interface GigabitEthernet0/2
switchport trunk encapsulation dot1q
switchport mode trunk
!
interface GigabitEthernet0/3
switchport mode dynamic desirable
!
interface GigabitEthernet0/4
switchport mode dynamic desirable
!
interface GigabitEthernet0/5
switchport mode dynamic desirable
!
interface GigabitEthernet0/6
switchport mode dynamic desirable
!
interface GigabitEthernet0/7
switchport mode dynamic desirable
!
interface GigabitEthernet0/8
switchport mode dynamic desirable
!
interface GigabitEthernet0/9
switchport mode dynamic desirable
!
interface GigabitEthernet0/10
switchport mode dynamic desirable
!
```

```
interface GigabitEthernet0/11
switchport mode dynamic desirable
!
interface GigabitEthernet0/12
switchport mode dynamic desirable
!
interface Vlan1
no ip address
shutdown
!
interface Vlan90
ipv6 address 2001:ABBA:90::2/64
!
ip classless
ip http server
ip http secure-server
!
ipv6 route ::/0 2001:ABBA:90::1
!
control-plane
!
!
line con 0
line vty 5 15
!
end
```

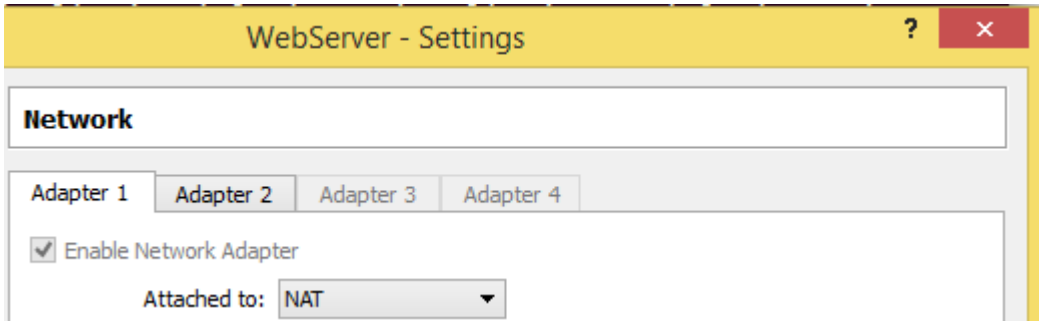
WG1-SW2

```
version 12.1
no service pad
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname WG1-SW2
!
!
ip subnet-zero
!
ip ssh time-out 120
ip ssh authentication-retries 3
!
spanning-tree mode pvst
no spanning-tree optimize bpdu transmission
spanning-tree extend system-id
!
interface FastEthernet0/1
```

```
switchport mode trunk
!
interface FastEthernet0/2
!
interface FastEthernet0/3
!
interface FastEthernet0/4
!
interface FastEthernet0/5
!
interface FastEthernet0/6
!
interface FastEthernet0/7
!
interface FastEthernet0/8
!
interface FastEthernet0/9
!
interface FastEthernet0/10
!
interface FastEthernet0/11
switchport access vlan 30
switchport mode access
!
interface FastEthernet0/12
switchport access vlan 40
switchport mode access
!
interface FastEthernet0/13
!
interface FastEthernet0/14
!
interface FastEthernet0/15
!
interface FastEthernet0/16
!
interface FastEthernet0/17
!
interface FastEthernet0/18
!
interface FastEthernet0/19
!
interface FastEthernet0/20
!
interface FastEthernet0/21
!
interface FastEthernet0/22
!
interface FastEthernet0/23
```

```
!
interface FastEthernet0/24
!
interface Vlan1
no ip address
no ip route-cache
shutdown
!
ip http server
!
line con 0
line vty 5 15
!
!
end
```

Liite 8. Laitteen olemassaolevat verkkorajapinnat



KUVIO 41. Virtuaalikoneen aktiiviset rajapinnat

```
root@nyoa-VirtualBox:/home/nyoa# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 08:00:27:fe:2a:8c
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fefe:2a8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6980 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2831 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6845856 (6.8 MB)  TX bytes:246947 (246.9 KB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:b1:12:19
          inet6 addr: 2001:abba:30::5/64 Scope:Global
          inet6 addr: fe80::a00:27ff:feb1:1219/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:20347 errors:0 dropped:0 overruns:0 frame:0
          TX packets:20483 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20766738 (20.7 MB)  TX bytes:20780393 (20.7 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:432 errors:0 dropped:0 overruns:0 frame:0
          TX packets:432 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:45963 (45.9 KB)  TX bytes:45963 (45.9 KB)

root@nyoa-VirtualBox:/home/nyoa#
```

KUVIO 42. Virtuaalikoneen rajapintojen todennus "ifconfig -a"

```
  GNU nano 2.2.6                File: /etc/hosts

127.0.0.1       localhost
127.0.1.1       nyoa-VirtualBox

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
2001:dead:40::5 serverbox
```

KUVIO 43. Virtuaalikoneen "/etc/hosts" tiedosto

Liite 9. Koko ohjelman lähdekoodi

```
#-*- coding: utf-8 -*-

#!/usr/bin/Python


import sys, os, socket, threading, time, select, signal, sched, argparse

import netifaces as ni

import psutil

from functools import partial

from PyQt4.QtGui import *

from PyQt4.QtCore import *

import PyQt4.Qwt5 as Qwt

from PyQt4.Qwt5.anynumpy import *



######### This section forms the graphical user interface #########


class Application(QMainWindow):


    threadLocal = threading.local()

    err1 = pyqtSignal(int, int)

    reset = pyqtSignal(int)


    inlistM = []

    outlistM = []
```

```
totallistM = []


def __init__(self, parent=None):
  super(Application, self).__init__()
  self.setGeometry(300, 300, 600, 600)
  self.setWindowTitle('Trafgen6')
  self.stoprequest = threading.Event()
  self.plot = PlotWidget()
  self.clientstatus = {}
  self.createwidgets()
  self.stopclient = threading.Event()
  self.stopserver = threading.Event()


def createwidgets(self):


  self.mainWidget = QWidget(self) #Dummy widget to contain the layout m$
  self.setCentralWidget(self.mainWidget)


  # Creating the main layout that contains vertical main layout and thr$


  self.mainLayout = QVBoxLayout(self.mainWidget)
  self.gLayout1 = QGridLayout()
  self.gLayout2 = QGridLayout()
  self.gLayout2.setColumnMinimumWidth(1, 15)
  self.gLayout2.setColumnMinimumWidth(3, 180)
```

```
self.gLayout3 = QGridLayout()
self.gLayout3.setColumnMinimumWidth(4, 10)
self.gLayout3.setColumnMinimumWidth(2, 200)

# Creating the boxes for Client and Server

self.serverBox = QGroupBox('Server')
self.clientBox = QGroupBox('Client')

# Creates

self.label2 = QLabel('Choose interface', self)

self.label3 = QLabel(' ', self)
self.label3.setTextInteractionFlags(Qt.TextSelectableByMouse)
self.label3.setAlignment(Qt.AlignLeft|Qt.AlignVCenter)

self.traffic1 = QLabel('0.0Mb/s', self)
self.traffic1.setAlignment(Qt.AlignRight)
self.traffic2 = QLabel('0.0Mb/s', self)
self.traffic2.setAlignment(Qt.AlignCenter)
self.traffic3 = QLabel('0.0Mb/s', self)
self.traffic3.setAlignment(Qt.AlignLeft)

self.button2 = QPushButton('start', self)
```

```
self.button2.clicked.connect(lambda: self.threadcontrol('1', 'null'))
self.button2.setEnabled(False)

self.button3 = QPushButton('stop', self)
self.button3.clicked.connect(lambda: self.serverstop('0'))
self.button3.setEnabled(False)

self.menubutton1 = QPushButton('None', self)
self.menu1 = QMenu(self)
self.menubox()
self.menubutton1.setMenu(self.menu1)

self.gLayout2.addWidget(self.label2, 1, 0)
self.gLayout2.addWidget(self.menubutton1, 1, 2)
self.gLayout2.addWidget(self.label3, 1, 3)
self.gLayout2.addWidget(self.button2, 1, 4)
self.gLayout2.addWidget(self.button3, 1, 5)

self.gLayout1.addWidget(self.traffic1, 1, 1)
self.gLayout1.addWidget(self.traffic2, 1, 3, 1 ,2)
self.gLayout1.addWidget(self.traffic3, 1, 5)

# Here are the dictionaries used in making GUI components in Client box

self.labeldic1 = {}
```

```
self.labeldic4 = {}

self.labeldic5 = {}

self.buttondic1 = {}

self.buttondic2 = {}

self.entrydic1 = {}

self.entrydic2 = {}

self.sliderdic1 ={}


# For loop adds all the client components to their respective dictionaries


for i in range (0,3):
  self.label1 = QLabel('Target IPv6 address')
  self.label5 = QLabel('Mb/s', self)


  self.label4 = QLabel('Connection failed', self)
  self.label4.setAlignment(Qt.AlignCenter)
  self.label4.hide()


  self.entry1 = QLineEdit(self)


  self.entry2 = QLineEdit(self)
  self.entry2.setMaxLength(2)
  self.entry2.setText("1")


  self.button1 = QPushButton('Start', self)
```

```
        self.button4 = QPushButton('stop', self)
        self.button4.setEnabled(False)

        self.slider1 = QSlider(self)
        self.slider1.setOrientation(Qt.Horizontal)
        self.slider1.setMinimum(1)
        self.slider1.setMaximum(10)
        self.slider1.setPageStep(1)
        self.slider1.setSingleStep(1)

        self.labeldic1.update({i:self.label1})
        self.labeldic4.update({i:self.label4})
        self.labeldic5.update({i:self.label5})
        self.entrydic1.update({i:self.entry1})
        self.entrydic2.update({i:self.entry2})
        self.buttondic1.update({i:self.button1})
        self.buttondic2.update({i:self.button4})
        self.sliderdic1.update({i:self.slider1})

    # Next for loop adds previously created components to the grid layout based on their position in their dictionaries

        rows = [1,3,5]
        x = 0
```

```
        for i in rows:
          self.gLayout3.addWidget(self.labeldic1[x],i,0)
          self.gLayout3.addWidget(self.labeldic5[x],i+1,3)
          self.gLayout3.addWidget(self.labeldic4[x],i+1,5,1,2)
          self.gLayout3.addWidget(self.entrydic1[x],i,1,1,4)
          self.gLayout3.addWidget(self.entrydic2[x],i+1,4)
          self.gLayout3.addWidget(self.buttondic1[x],i,5)
          self.gLayout3.addWidget(self.buttondic2[x],i,6)
          self.gLayout3.addWidget(self.sliderdic1[x],i+1,1,1,2)
          x += 1

        # This part builds the final layout by adding the plot and grid layouts to the main layout

        self.mainLayout.addWidget(self.plot)
        self.mainLayout.addLayout(self.gLayout1)
        self.serverBox.setLayout(self.gLayout2)
        self.mainLayout.addWidget(self.serverBox)
        self.clientBox.setLayout(self.gLayout3)
        self.mainLayout.addWidget(self.clientBox)

        # Next two dictionaries contain the code that links each clients slider to their respec-
tive entry

        funcdic1 = {}
        for x in range(0,3):
```

```
  def slidechanged(x=x):
     svalue = str(self.sliderdic1[x].value())
     self.entrydic2[x].setText(svalue)
  funcdic1.update({x:slidechanged})


funcdic2 = {}
for x in range(0,3):
  def textchanged(x=x):
    try:
     value = int(self.entrydic2[x].text())
     if int(value) > 0 and int(value) <= 10:
        self.sliderdic1[x].setValue(value)
     else:
        self.entrydic2[x].setText('10')
        self.sliderdic1[x].setValue(10)
    except:
        self.entrydic2[x].setText('1')
        self.sliderdic1[x].setValue(1)
  funcdic2.update({x:textchanged})


for i in range(0,3):
  self.buttondic1[i].clicked.connect(partial(self.threadcontrol, '2', i))
  self.buttondic2[i].clicked.connect(partial(self.clientstop, '0', i))


self.sliderdic1[0].valueChanged.connect(lambda: funcdic1[0]())
```

```
        self.sliderdic1[1].valueChanged.connect(lambda: funcdic1[1]())
        self.sliderdic1[2].valueChanged.connect(lambda: funcdic1[2]())
        self.entrydic2[0].editingFinished.connect(lambda: funcdic2[0]())
        self.entrydic2[1].editingFinished.connect(lambda: funcdic2[1]())
        self.entrydic2[2].editingFinished.connect(lambda: funcdic2[2]())

        self.err1.connect(self.errhandler)
        self.plot.plotchanged.connect(self.traffic_presenter)

    def traffic_presenter(self, inlist, outlist, totallist):

        self.traffic1.setText(str(outlist) + 'Mb/s')
        self.traffic2.setText(str(inlist) + 'Mb/s')
        self.traffic3.setText(str(totallist) + 'Mb/s')

    def menubox(self):

        self.menu1.addAction('None', lambda interface='None': self.callback(interface))
        for interface in ni.interfaces():
            if not interface.startswith('lo'):
                self.menu1.addAction(interface, lambda interface=interface:
self.callback(interface))

    def callback(self, interface):
```

```
        if not interface.startswith('lo'):


            try:
                self.intip = ni.ifaddresses(interface)[ni.AF_INET6][0]['addr']
                self.menubutton1.setText(interface)
            except:
                self.intip = 'None'


            if self.intip.startswith('fe80') or self.intip == 'None':#If the address is link local it is
ignored
              self.label3.setText('No IPv6 address found')
              self.button2.setEnabled(False)
            else:
              self.label3.setText(self.intip)
              self.button2.setEnabled(True)


         if interface =='None':
            self.menubutton1.setText('None')
            self.label3.setText(' ')
            self.button2.setEnabled(False)


        def buttonswitch(self, flip, cn):
         if flip == "1":
            self.button3.setEnabled(True)
            self.menubutton1.setEnabled(False)
```

```
            self.button2.setEnabled(False)
        elif flip == "0":
            self.button3.setEnabled(False)
            self.button2.setEnabled(True)
            self.menubutton1.setEnabled(True)
        elif flip == "2":
            self.buttondic2[cn].setEnabled(True)
            self.entrydic2[cn].setEnabled(False)
            self.sliderdic1[cn].setEnabled(False)
            self.buttondic1[cn].setEnabled(False)
        elif flip == "3":
            self.buttondic2[cn].setEnabled(False)
            self.entrydic2[cn].setEnabled(True)
            self.sliderdic1[cn].setEnabled(True)
            self.buttondic1[cn].setEnabled(True)

    def threadcontrol(self, threadtype, cn):
        if threadtype == "1":
            self.thread1 = threading.Thread(target = self.serverstart)
            self.thread1.start()
        elif threadtype == "2":
            self.thread2 = threading.Thread(target = self.clientstart, args=[cn])
            self.thread2.start()

    def update_label(self, status):
```

```
        self.label4.show()


def clientstop(self, status, cn):
  self.clientstatus.update({cn:"1"})
  if status =="0":
        self.clientstatus.update({cn:status})
  self.buttonswitch("3", cn)


def serverstop(self, status):
  self.stopserver.set()
  time.sleep(1.0 / 5)
  self.buttonswitch("0", 'null')


def serverstart(self):
  self.stopserver.clear()
  self.buttonswitch("1",'null')
  self.host = self.intip
  self.port = 5001
  s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
  s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  s.settimeout(0.01)
  s.bind((self.host, self.port))


  n = 1
  packets = bool(False)
```

```
 while not self.stopserver.isSet():
     try:
         message, address = s.recvfrom(1500)
         s.sendto(message, address)
     except:
         pass
 s.close()
 self.buttonswitch("0", 'null')

def clientstart(self, cn):
 self.threadLocal.cn = cn

 self.threadLocal.sch = sched.scheduler(time.time, time.sleep)
 self.threadLocal.next_time = time.time()
 self.threadLocal.sch.enterabs(self.threadLocal.next_time, 0, self.oneround, ())

 self.reset.emit(cn)
 time.sleep(0.03)
 targetip = self.entrydic1[cn].text()
 host = targetip
 port = 5001
 speed = int(self.entrydic2[cn].text())
 self.threadLocal.delay = 1.0 / (speed*128)
 n = bool(False)
 err = 0
```

```
openfile = open('1024')
self.threadLocal.message = openfile.read()


if targetip:
    self.threadLocal.s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
    self.threadLocal.s.setblocking(0)
    try:
        self.threadLocal.s.connect((host,port))
        self.threadLocal.s.send(self.threadLocal.message)
        time.sleep(0.1)
        if len(self.threadLocal.s.recvfrom(1500)) > 1:
            n = (True)
    except IOError as (errno, strerror):
        if errno == -9:
            err = 1
            self.err1.emit(1, self.threadLocal.cn)
        if errno == 11:
            err = 1
            self.err1.emit(5, self.threadLocal.cn)
        if errno == -2:
            err = 1
            self.err1.emit(4, self.threadLocal.cn)
    self.clientstatus.update({self.threadLocal.cn:"1"})
    self.buttonswitch("2", cn)
```

```
            self.threadLocal.counter = 0
            if n == True:
                self.count = 0
                self.threadLocal.sch.run()
            elif err == 0:
                self.err1.emit(5, self.threadLocal.cn)

            self.threadLocal.s.close()
            self.buttonswitch("3", cn)
        else:
            self.err1.emit(3, self.threadLocal.cn)

    def oneround(self):
        if not self.stopclient.isSet():
            try:
                self.threadLocal.s.send(self.threadLocal.message)
                status = self.clientstatus[self.threadLocal.cn]
                self.err1.emit(2, self.threadLocal.cn)
                self.threadLocal.counter += 1
                if status == '1':
                    self.threadLocal.next_time += self.threadLocal.delay
                    self.threadLocal.sch.enterabs(self.threadLocal.next_time, 0, self.oneround, ())
                else:
                    self.err1.emit(0, self.threadLocal.cn)
                    self.threadLocal.s.close()
```

```
    except IOError as (errno, strerror):
        if errno == 111:
            self.err1.emit(6, self.threadLocal.cn)
            self.threadLocal.s.close()


def errhandler(self, value, cn):
    if value == 1:
        self.labeldic4[cn].setText('Invalid address')
        self.labeldic4[cn].show()
    elif value == 0:
        self.labeldic4[cn].hide()
    elif value == 2:
        self.labeldic4[cn].setText('Sending packets')
        self.labeldic4[cn].show()
    elif value == 3:
        self.labeldic4[cn].setText('No IP address given')
        self.labeldic4[cn].show()
    elif value == 4:
        self.labeldic4[cn].setText('Name or service not known')
        self.labeldic4[cn].show()
    elif value == 6:
        self.labeldic4[cn].setText('Connection lost')
        self.labeldic4[cn].show()
    else:
        self.labeldic4[cn].setText('Connection failed')
```

```
            self.labeldic4[cn].show()


        def exit(self):
          self.stopserver.set()
          self.stopclient.set()
          if 'thread1' in locals():
              self.thread1.join()
          if 'thread2' in locals():
              self.thread2.join()


class PlotWidget(Qwt.QwtPlot):


        plot = pyqtSignal()
        plotchanged = pyqtSignal(float, float, float)


        def __init__(self, parent = None):
          Qwt.QwtPlot.__init__(self, parent)
          self.inlist = []
          self.outlist = []
          self.totallist = []


          self.setlandscape()
          self.create_curves()
          self.resize(550,200)
          self.move(10,20)
```

```
  self.networkthread = threading.Thread(target = self.netcalc)
  self.networkthread.setDaemon(True)
  self.networkthread.start()

  self.plot.connect(self.plot_data)

def setlandscape(self):
  self.setCanvasBackground(Qt.white)
  self.plotLayout().setAlignCanvasToScales(True)

  self.setAxisTitle(Qwt.QwtPlot.yLeft, 'Traffic (Mb)')
  self.setAxisTitle(Qwt.QwtPlot.xBottom, 'Time (seconds)')
  self.setAxisScale(Qwt.QwtPlot.yLeft, 0., 10.)
  self.setAxisScale(Qwt.QwtPlot.xBottom, 60., 0.)

  self.legend = Qwt.QwtLegend()
  self.legend.setFrameStyle(QFrame.Sunken)
  self.legend.setItemMode(Qwt.QwtLegend.ClickableItem)
  self.insertLegend(self.legend, Qwt.QwtPlot.BottomLegend)

  self.grid = Qwt.QwtPlotGrid()
  self.grid.enableXMin(True)
  self.grid.setMajPen(QPen(QPen(Qt.gray)))
  self.grid.setMinPen(QPen(QPen(Qt.lightGray)))
```

```
  self.grid.attach(self)


def create_curves(self):
  self.curve1 = Qwt.QwtPlotCurve('Outbound traffic')
  self.curve1.setPen(QPen(Qt.red, 2))
  self.curve1.setStyle(Qwt.QwtPlotCurve.Lines)
  self.curve1.attach(self)
  self.curve1.setRenderHint(Qwt.QwtPlotItem.RenderAntialiased)

  self.curve2 = Qwt.QwtPlotCurve('Inbound traffic')
  self.curve2.setPen(QPen(Qt.blue, 2))
  self.curve2.setStyle(Qwt.QwtPlotCurve.Lines)
  self.curve2.attach(self)
  self.curve2.setRenderHint(Qwt.QwtPlotItem.RenderAntialiased)

  self.curve3 = Qwt.QwtPlotCurve('Total traffic')
  self.curve3.setPen(QPen(Qt.black, 2))
  self.curve3.setStyle(Qwt.QwtPlotCurve.Lines)
  self.curve3.attach(self)
  self.curve3.setRenderHint(Qwt.QwtPlotItem.RenderAntialiased)

def plot_data(self):
  xlist = []
  n = 0
  for x in self.outlist:
```

```
        xlist.insert(n,n)
        n += 1
    self.curve1.setData(xlist, self.outlist)
    self.curve2.setData(xlist, self.inlist)
    self.curve3.setData(xlist, self.totallist)
    if max(self.totallist) > 10 and max(self.totallist) < 50:
        self.setAxisScale(Qwt.QwtPlot.yLeft, 0., 50.)
    elif max(self.totallist) > 50:
        self.setAxisScale(Qwt.QwtPlot.yLeft, 0., 100.)
    else:
        self.setAxisScale(Qwt.QwtPlot.yLeft, 0., 10.)
    self.replot()

def netcalc(self):
    mb = 1048576.00
    while True:
        sent = {}
        recv = {}
        inwards = []
        out = []
        total = []
        n = 1
        for i in ni.interfaces():
            counter = psutil.net_io_counters(pernic=True)[i]
            sent.update({n:counter.bytes_sent})
```

```
        recv.update({n:counter.bytes_recv})
        n +=1
time.sleep(1)
n = 1
for i in ni.interfaces():
        counter = psutil.net_io_counters(pernic=True)[i]
        totalsent = counter.bytes_sent - sent[n]
        totalrecv = counter.bytes_recv - recv[n]
        alldata = totalsent + totalrecv
        self.totalout = (totalsent/mb) * 8
        self.totalin = (totalrecv/mb) * 8
        self.totalinout = (alldata/mb) * 8
        inwards.insert((n-1), float("%.2f" % self.totalin))
        out.insert((n-1), float("%.2f" % self.totalout))
        total.insert((n-1), float("%.2f" % self.totalinout))
        n += 1
self.inlist.insert(0, sum(inwards))
self.outlist.insert(0, sum(out))
self.totallist.insert(0, sum(total))
if len(self.outlist) > 60:
        self.inlist.pop()
        self.outlist.pop()
        self.totallist.pop()
self.plotchanged.emit(self.inlist[0], self.outlist[0], self.totallist[0])
self.plot.emit()
```

```
###### This section is run by command line ########


class Server(threading.Thread):
      def __init__(self, host, port, parent = None):
        super(Server, self).__init__(parent)
        self.host = host
        self.port = port
        self.stoprequest = threading.Event()

      def run(self):

        host = ni.ifaddresses(self.host)[ni.AF_INET6][0]['addr']
        port = self.port
        self.stoprequest = threading.Event()

      def run(self):

        host = ni.ifaddresses(self.host)[ni.AF_INET6][0]['addr']
        port = self.port
        s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.settimeout(0.01)
```

```
        try:
            s.bind((host, port))
            bind = (True)
        except:
            print 'Could not bind to requested address: ' + host
            os.kill(os.getpid(), signal.SIGINT)
            bind = (False)

        if bind == True:
            print 'Server address:  ' + host + '\n' + 'Server port:  ' + str(port)

            while not self.stoprequest.isSet():
                try:
                    message, address = s.recvfrom(1500)
                    s.sendto(message, address)
                except:
                    pass
            s.close()

    def join(self, timeout=None):
        self.stoprequest.set()
        super(Server, self).join(timeout)

class Client(threading.Thread):
```

```
threadLocal = threading.local()


def __init__(self, host, port, rate, duration, parent = None):
  super(Client, self).__init__(parent)
  self.host = host
  self.port = port
  self.rate = rate
  self.duration = duration
  self.stoprequest = threading.Event()


def run(self):
  self.threadLocal.sch = sched.scheduler(time.time, time.sleep)
  self.threadLocal.next_time = time.time()
  self.threadLocal.sch.enterabs(self.threadLocal.next_time, 0, self.oneround, ())


  host = self.host
  port = self.port
  speed = self.rate
  duration = self.duration
  self.threadLocal.delay = 1.0 / (speed*128)
  n = bool(False)
  error = (False)
  openfile = open('1024')
  self.threadLocal.message = openfile.read()
```

```
self.threadLocal.s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
self.threadLocal.s.setblocking(0)
try:
    self.threadLocal.s.connect((host,port))
    self.threadLocal.s.send(self.threadLocal.message)
    time.sleep(0.1)
    if len(self.threadLocal.s.recvfrom(1500)) > 1:
        n = (True)
except IOError as (errno, strerror):
    if errno == -9:
        print 'Invalid address'
        error = (True)
        os.kill(os.getpid(), signal.SIGINT)
    if errno == 11:
        print 'Connection failed'
        error = (True)
        os.kill(os.getpid(), signal.SIGINT)
    if errno == -2:
        print 'Name or service unknown'
        error = (True)
        os.kill(os.getpid(), signal.SIGINT)

if n == True:
    self.timeout = time.time() + duration
    self.threadLocal.sch.run()
```

```
  elif error == False:
        print 'Connection failed'
        os.kill(os.getpid(), signal.SIGINT)

def oneround(self):
  if not self.stoprequest.isSet():
    try:
        self.threadLocal.s.send(self.threadLocal.message)
        self.threadLocal.next_time += self.threadLocal.delay
        if self.duration != 0 and time.time() > self.timeout:
              print '\n' + 'Client socket closed after ' + str(self.duration) + ' seconds'
              self.threadLocal.s.close()
              os.kill(os.getpid(), signal.SIGINT)
        else:
              self.threadLocal.sch.enterabs(self.threadLocal.next_time, 0, self.oneround, ())

    except IOError as (errno, strerror):
        if errno == 111:
              print '\n' + 'Connection lost'
              self.threadLocal.s.close()
              os.kill(os.getpid(), signal.SIGINT)

def join(self, timeout=None):
  self.stoprequest.set()
  super(Client, self).join(timeout)
```

```
class calculator(threading.Thread):


    def __init__(self, parent = None):
      super(calculator, self).__init__(parent)
      self.stoprequest = threading.Event()


    def run(self):
      mb = 1048576.00
      while not self.stoprequest.isSet():
          sent = {}
          recv = {}
          inwards = []
          out = []
          total = []
          n = 1
          for i in ni.interfaces():
              counter = psutil.net_io_counters(pernic=True)[i]
              sent.update({n:counter.bytes_sent})
              recv.update({n:counter.bytes_recv})
              n +=1
          time.sleep(1)
          n = 1
          for i in ni.interfaces():
              counter = psutil.net_io_counters(pernic=True)[i]
```

```
                totalsent = counter.bytes_sent - sent[n]
                totalrecv = counter.bytes_recv - recv[n]
                alldata = totalsent + totalrecv
                self.totalout = (totalsent/mb) * 8
                self.totalin = (totalrecv/mb) * 8
                self.totalinout = (alldata/mb) * 8
                inwards.insert((n-1), float("%.2f" % self.totalin))
                out.insert((n-1), float("%.2f" % self.totalout))
                total.insert((n-1), float("%.2f" % self.totalinout))
                n += 1
            inwards = sum(inwards)
            out = sum(out)
            total = sum(total)
            if not self.stoprequest.isSet():
                sys.stdout.write('\r' + 'Out ' + str(out) + 'Mb/s     ' + 'In ' + str(inwards) + 'Mb/s
'   'Total ' + str(tot$
                sys.stdout.flush()


        def join(self, timeout=None):
          self.stoprequest.set()
          super(calculator, self).join(timeout)


########## This section initiates the program ###########


if __name__ == '__main__':
```

```
f = open('1024','w')
for i in range(0, 962):
	f.write('@')
f.close()


def valid_interfaces():
	for i in ni.interfaces():
		try:
			ip = ni.ifaddresses(i)[ni.AF_INET6][0]['addr']
			if not ip.startswith('fe80') and not i.startswith('lo'):
				list.insert(0,i)
		except KeyError:
			print 'Interface \"' + i + '\" does not have an IP address'

parser = argparse.ArgumentParser(description='Send Ipv6 traffic through your network')
group1 = parser.add_mutually_exclusive_group()
group2 = parser.add_mutually_exclusive_group()
mode = 0
list = []
nohelp = (False)
valid_interfaces()

# This part creates the argument parser used to create command line user interface.
```

```
    group1.add_argument('-g', '--gui',
                        action='store_true', dest='gui',
                        help='Start graphical user interface. Do not use other arguments with this op-
tion')
    group1.add_argument('-s', '--server',
                        action='store_true', dest='server',
                        help='Starts the program in server mode')
    group1.add_argument('-c', '--client',
                        action='store_true', dest='client',
                        help='Starts the program in client mode')
    group2.add_argument('-i', '--interface',
                        action='store', dest='interface', choices=list,
                        help='Specifies the interface for server')
    group2.add_argument('-d', '--destip',
                        action='store', dest='targetip',
                        help='Sets the destination IPv6 address eg.2001:dead:beef::1')
    parser.add_argument('-p', '--port',
                        action='store', dest='port', type=int,
                        help='Specifies the port used by server or client | Default 5001', default=5001)
    parser.add_argument('-t', '--time',
                        action='store', dest='time', type=float,
                        help='Client only. Specifies in seconds how long traffic will be sent | Default (0)
is unlimited', default=0)
    parser.add_argument('-r', '--rate',
                        action='store', dest='rate', type=float,
```

```
                help='Client only. Sets the rate at which traffic will be sent (use plain numbers
such as 1 or 1.5) | Default 1Mb$


    # If there are no arguments the program will print help text and exit


    if len(sys.argv)==1:
        parser.print_help()
        sys.exit(1)
    args = parser.parse_args()


    # This part starts the GUI if the flag is set


    if args.gui:
        app = QApplication(sys.argv)
        app.setStyle('cleanlooks')
        mainwindow = Application()
        mainwindow.show()
        cr = app.exec_()
        mainwindow.exit()
        sys.exit(cr)


    # If the flag for server is set on and an interface is given the program starts in server
mode


    if args.server and args.interface:
```

```
        thread1 = Server(host=args.interface, port=args.port)
        thread1.start()
        mode = 1

    # If the flag for client is set on and a target ip address is given the program starts in client mode

    if args.client and args.targetip:
        thread1 = Client(args.targetip, args.port, args.rate, args.time)
        thread1.start()
        mode = 1

    # If the flag for client is set on and a target ip address is given the program starts in client mode

    if args.client and args.targetip:
        thread1 = Client(args.targetip, args.port, args.rate, args.time)
        thread1.start()
        mode = 2

    if mode != 0 and thread1.isAlive():
        thread2 = calculator()
        try:
            thread2.start()
        except:
```

```
            nohelp = (True)


if threading.active_count() > 2:
    while threading.active_count > 2:
        try:
            time.sleep(0.01)
        except KeyboardInterrupt:
            thread1.join()
            if mode == 1:
                print '\n' + 'Server stopped'
            elif mode == 2:
                print '\n' + 'Client stopped'
            thread2.join()
            sys.exit()
elif nohelp == False:
    parser.print_help()
    sys.exit(1)
```