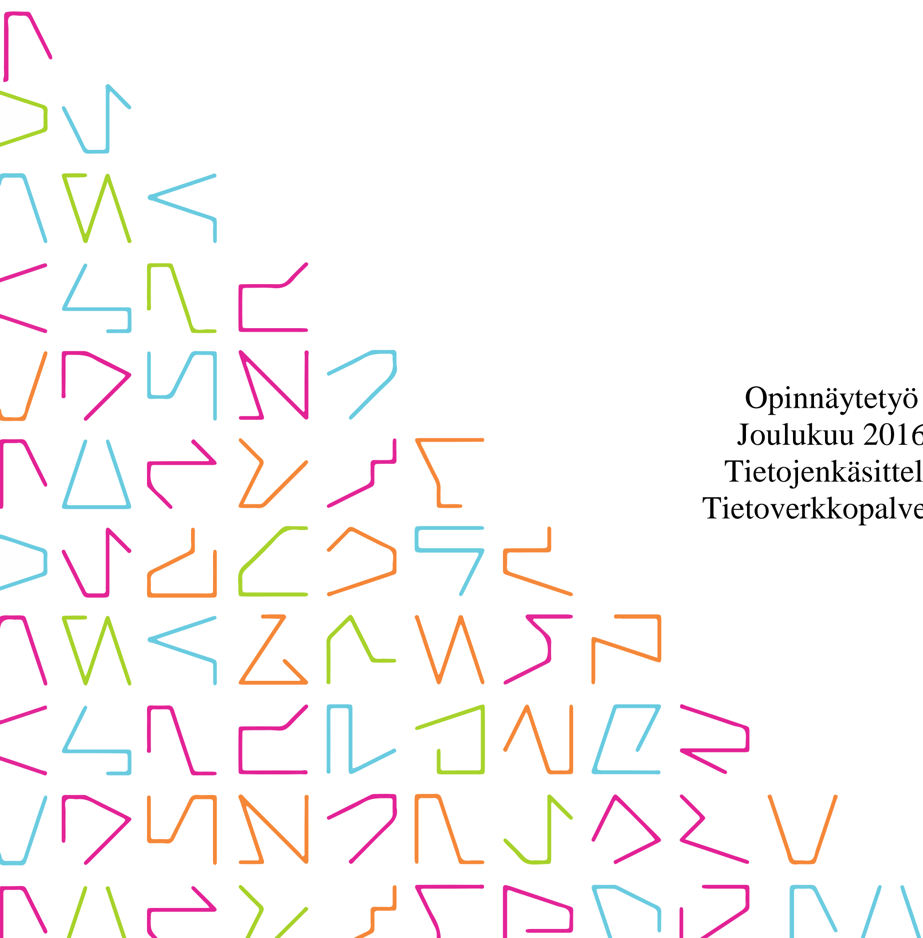
# ROBOT FRAMEWORK REITITYKSEN AUTOMAATTISEN TESTAAMISEN TOTEUTUKSESSA

Henri Karjalainen



Opinnäytetyö
Joulukuu 2016
Tietojenkäsittely
Tietoverkkopalvelut

**TIIVISTELMÄ**

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Tietoverkkopalvelut

KARJALAINEN, HENRI:
Robot Framework reitityksen automaattisen testaamisen toteutuksessa

Opinnäytetyö 39 sivua, joista liitteitä 3 sivua
Joulukuu 2016

---

Opinnäytetyön tavoitteena oli tutkia reitityksen automaattista testaamista Robot Framework -testausautomaatiokehyksen näkökulmasta ja selvittää, miten hyvin Robot Framework soveltuu reititystestaamisen automatisoinnin toteuttamiseen. Opinnäytetyön toimeksiantajana toimi Bittium Wireless Oy, ja koska yrityksen kehittämät teknologiat ovat salatuiksi luokiteltuja, opinnäytetyössä aihetta käsiteltiin vain yleisellä tasolla. Tarkoituksena oli tuottaa tietoa reitityksen automaattisesta testaamisesta ja Robot Frameworkin soveltuvuudesta reitityksen automaattisen testaamisen työvälineeksi.

Taustatyö ja käytännön tutkimus toteutettiin toimeksiantajan testiympäristössä ja kirjallisuuden avulla tutkimuksessa kerättyä tietoa pyrittiin esittämään opinnäytetyöraportissa yleispätevästi, jotta tietoa olisi mahdollista soveltaa mahdollisimman laaja-alaisesti. Lopputuotoksena syntynyt opinnäytetyöraportti ei ole varsinainen opas reititystestauksen toteuttamiseen Robot Frameworkilla vaan tietopaketti, jonka avulla on mahdollista tutustua testaamiseen, automaatioon, reititykseen ja Robot Frameworkiin yksittäisinä aihepiireinä sekä näiden kaikkien toteutukseen yhtenä kokonaisuutena.

Johtopäätöksinä opinnäytetyössä todetaan, että reitityksen testaaminen voi käsin toteutettuna olla työlästä ja aikaa vievää. Automaation avulla testaamista voidaan merkittävästi tehostaa ja tuloksien luotettavuutta parantaa. Robot Frameworkin todetaan ohjelmistoriippumattomuutensa, yleispätevän luonteensa ja laajennettavuutensa puolesta olevan soveltuva työväline reitityksen automaattisen testaamisen toteuttamiseen.

---

Asiasanat: reititys, tietoverkot, testaaminen, automaatio, robot framework

**ABSTRACT**

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Network Services

KARJALAINEN, HENRI:
Robot Framework in Implementation of Automated Network Routing Testing

Bachelor's thesis 39 pages, appendices 3 pages
December 2016

---

The objective of this thesis was to examine automated testing of network routing from the viewpoint of Robot Framework test automation framework. The suitability of Robot Framework for implementation of automated network routing testing was also investigated. The thesis was commissioned by Bittium Wireless Oy and, because their technologies are classified information, the subject of the thesis was covered only in general. The purpose of the thesis was to produce information about automated testing of network routing and suitability of Robot Framework to act as a tool of implementation in automated network routing testing.

Background work and practical research for this thesis were done in a test environment provided by commissioner. Key theoretical points were drawn from related literature in order to introduce the subject and to represent the gathered information in a general way. As a result of the thesis, a collection of information was produced with which it is possible to familiarize with testing, automation, network routing and Robot Framework.

In conclusion, it was found that testing of network routing can be troublesome and time consuming when it is done manually. With automation, the testing process can be greatly improved thus increasing the quality. The findings also indicate that Robot Framework is suitable for automated testing of network routing due to its platform-independent, generic nature and extensibility.

---

Key words: routing, networks, testing, automation, robot framework

# SISÄLLYS


1 JOHDANTO ... 7
2 TESTAAMINEN JA AUTOMAATIO ... 8
  2.1 Testaamisen määrittely ... 8
  2.2 Testaaminen tuotekehitysprosessissa ... 9
  2.3 Testaamisen automatisointi ... 10
  2.4 Automaation hyödyt ... 11
3 REITITYS TIETOLIIKENTEESSÄ ... 12
  3.1 Reitittäminen yleisesti ... 12
  3.2 Staattinen reititys ... 14
  3.3 Dynaaminen reititys ... 14
  3.4 Ryhmälähetys ... 15
  3.5 Reititysprotokollat ... 15
    3.5.1 OSPF ... 16
    3.5.2 BGP ... 17
    3.5.3 RIP ... 18
    3.5.4 EIGRP ... 18
    3.5.5 IS-IS ... 19
    3.5.6 OLSR ... 19
    3.5.7 B.A.T.M.A.N ... 20
4 ROBOT FRAMEWORK ... 21
  4.1 Yleisesti ... 21
  4.2 Asentaminen ja toiminta ... 21
  4.3 Testien ohjelmointi ... 22
    4.3.1 Testitapaus ... 23
    4.3.2 Testikokoelma ... 23
  4.4 Testien suorittaminen ... 24
  4.5 Muut testaamisen työvälineet ... 25
    4.5.1 Iperf ... 25
    4.5.2 Paketin analysointi ... 26
    4.5.3 Ping ... 26
    4.5.4 Traceroute ... 27
5 REITITYKSEN AUTOMAATINEN TESTAAMINEN ... 28
  5.1 Testitapausten suunnittelu ... 28
  5.2 Testausympäristö ... 29
  5.3 Robot Framework testaamisen automatisoinnissa ... 30
  5.4 Kerättävä data ... 33

5.5 Tulosten analysointi ... 33
5.6 Tulosten merkitys kehitystyössä ... 34
6 POHDINTA ... 35
LÄHTEET ... 36
LIITTEET ... 37
Liite 1. Esimerkki Robot Frameworkin mukautetusta testikirjastosta. ... 37
Liite 2. Esimerkki Robot Frameworkin testikokoelmasta. ... 39

# LYHENTEET JA TERMIT

| | |
|---|---|
| Automaatio | Koneen suorittama itsenäinen toiminta ilman ihmisen väliintuloa |
| ICMP | Internet Control Message Protocol, verkon virhetilanteiden tarkkailuun käytettävä protokolla |
| IGMP | Internet Group Management Protocol, ryhmälähetysryhmien hallintaan käytettävä protokolla |
| IP | Internet Protocol, tiedonsiirtoprotokolla |
| MANET-verkko | Mobile ad-hoc Network. Langaton verkko, missä reitityssolmut ovat yhteydessä toisiinsa ilman tukiasemaa |
| OSI-malli | Tiedonsiirtoa seitsemässä kerroksessa kuvaava viitemalli |
| Python | Ohjelmointikieli |
| Reititin | IP-paketteja verkosta toiseen välittävä verkon aktiivilaite |
| Reititys | Reitittimen suorittama toimenpide, jolla IP-paketti välitetään lähdeosoitteesta kohdeosoitteeseen |
| Reitityssprotokolla | Protokolla, jolla reitittimet vaihtavat tietoa verkon topologiasta |
| reStructuredText | Pääsääntöisesti Python-ohjelmoinnissa käytetty selkokielinen dokumentointisyntaksi |
| Robot Framework | Python-pohjainen avoimen lähdekoodin testausautomaatiokehys |
| TCP | Transmission Control Procotol, yhteydellinen tiedonsiirtoprotokolla |
| TCP/IP | Internetin tiedonsiirron arkkitehtuuria kuvaava viitemalli |
| Testaaminen | Kehitystyön tukitoimi, jolla pidetään yllä tuotteen laatua |
| UDP | User Datagram Protocol, yhteydetön tiedonsiirtoprotokolla |

# 1 JOHDANTO

Tietoyhteiskunta kehittyy jatkuvasti ja yhä useampi laite kytkeytyy Internettiin tai jonkinlaiseen tietoverkkoon. Tästä syystä myös reitityksellä on yhä suurempi merkitys joka päiväisessä elämässämme; selailimme sitten uutisia tietokoneella, katsoimme elokuvia älytelevisiolla tai osallistuimme ryhmäkeskusteluun puhelimen viestintäsovelluksella. Kaikki lähettämämme ja vastaanottamamme tietoliikenne reitittyy usean eri verkon kautta ennen päätymistä vastaanottajalle. Lisääntyvän tiedon jakamisen myötä reitityksen merkitys tietoliikenteessä korostuu entisestään ja uusia teknologioita kehitetään, jotta pysytään mukana jatkuvasti kasvavan digitaalisen viestinnän muutoksessa.

Kuten kaikessa kehitystyössä, myös uusia reititysratkaisuja on tärkeää testata ennen niiden käyttöönottoa. Testaaminen voi pahimmillaan olla hyvin itseään toistavaa, koska samaa asiaa täytyy testata useasti kehitystyön eri vaiheissa. Tästä syystä testaamista on kannattavaa pyrkiä automatisoimaan. Tämän opinnäytetyön tavoitteena on tutkia reitityksen automaattista testaamista Robot Framework -testausautomaatiokehyksen näkökulmasta. Tavoitteena on myös pohtia, miten ja mitä reitityksestä olisi hyödyllistä testata. Opinnäytetyön tarkoituksena on tuottaa tietoa reitityksen automaattisesta testaamisesta ja Robot Frameworkin soveltuvuudesta testaamisen sekä automaation toteutukseen.

Opinnäytetyön toimeksiantajana toimii Bittium Wireless Oy, joka viestintä- ja liitettävyysratkaisuja kehittävänä yrityksenä on kiinnostunut reitityksen automaattisen testaamisen toteuttamisesta. Bittium tarjoaa asiakkailleen luotettavia ja turvallisia ratkaisuja ja palveluita. Testaaminen on luotettavuuden ja turvallisuuden saavuttamiseksi tärkeä osa kehitystyötä. Testauksen automatisointi vähentää manuaalisia rutiinitoimenpiteitä ja lisää toistettavuutta sekä testauksen määrää ja luotettavuutta erilaisten järjestelmien toiminnan varmistamiseksi.

Opinnäytetyössä käsitellään aluksi testaamisen ja automaation merkitystä tuotekehitystyössä yleisesti. Tämän jälkeen perehdytään reititykseen tietoliikenteessä ja testausautomaatiokehys Robot Frameworkin toiminnallisuuksiin. Lopuksi asioita tarkastellaan kokonaisuutena ja pohditaan, miten reititystä automatisoidusti testataan, ja miten Robot Framework soveltuu kyseiseen tarkoitukseen.

# 2 TESTAAMINEN JA AUTOMAATIO

## 2.1 Testaamisen määrittely

Testaaminen on tärkeä osa jokaista tuotekehitysprosessia. Se on tuotannon tukitoimi ja laadunvalvonnan väline, joka tulisi ottaa huomioon jo kehitysprojektin suunnitteluvaiheessa. Testaamista olisi hyvä jatkaa läpi kehitystyön aina tuotteen tai palvelun käyttöönottoon ja käytöstä poistamiseen saakka. (Mette Jonassen Hass 2008, 1; Hutcheson 2003, 25.) Testaamisella pyritään varmistamaan, että tuote täyttää ne vaatimukset, jotka sille on asetettu, ja että se toimii niin kuin sen odotetaan toimivan. Testaamisen tarkoituksena on parantaa tuotannon laatua ja nopeuttaa kehitystyötä. (Menon 2013, 6.)

Ohjelmistokehityksessä testaamisella pyritään paljastamaan ohjelman tai järjestelmän sisältämät virheet (*bug*), jotka ovat haitaksi sen toiminnalle tai jotka sallivat ohjelman tehdä jotain sellaista, mitä sen ei pitäisi tehdä. Testaaminen aloitetaan oletuksesta, että ohjelma sisältää virheitä. (Myers, Sandler & Badgett 2011, 2, 6.) Tästä syystä testaamista olisi hyvä suorittaa sitä varten koulutetuilla henkilöillä, sillä kehittäjät eivät välttämättä osaa tarkastella kehittämäänsä ohjelmaa ulkopuolisen näkökulmasta, jolloin kaikkia virheitä ei välttämättä havaita. Kehittäjien oletuksena lähtökohtaisesti on, että ohjelma toimii, kun taas testaajien lähtökohtana on, että ohjelma sisältää virheitä. (Loveland, Shannon & Miller 2004, 6, 9-10.)

Testaaminen jaetaan useampaan vaiheeseen, joita voivat olla suunnittelu, testien luonnostelu, toteutus ja suoritus sekä tulosten analysointi ja raportointi. Testausprosessi tulisi suunnitella jokaiselle projektille erikseen. Suunnitteluvaiheessa on tärkeää määritellä testaamisstrategia, jossa kerrotaan, miten testaaminen kokonaisuudessaan toteutetaan ja mikä on testattavien asioiden tärkeysjärjestys. Yksittäisten testien suunnittelussa on tärkeää päättää, mitä testataan, miten testataan, kauanko testaamiseen kuluu aikaa ja mitä tietoa testaamisesta halutaan kerätä. Testien luonnosteluvaiheessa on hyvä tietää testattavien asioiden tarkka määrittely, jotta testitapaukset voidaan muodostaa. Määrittelyistä voidaan käydä keskustelua kehittäjien kanssa, tai jos ohjelmiston ominaisuuksista ovat tarkat dokumentaatiot, voidaan niitä käyttää hyödyksi. (Spillner, Linz & Schafer 2014.)

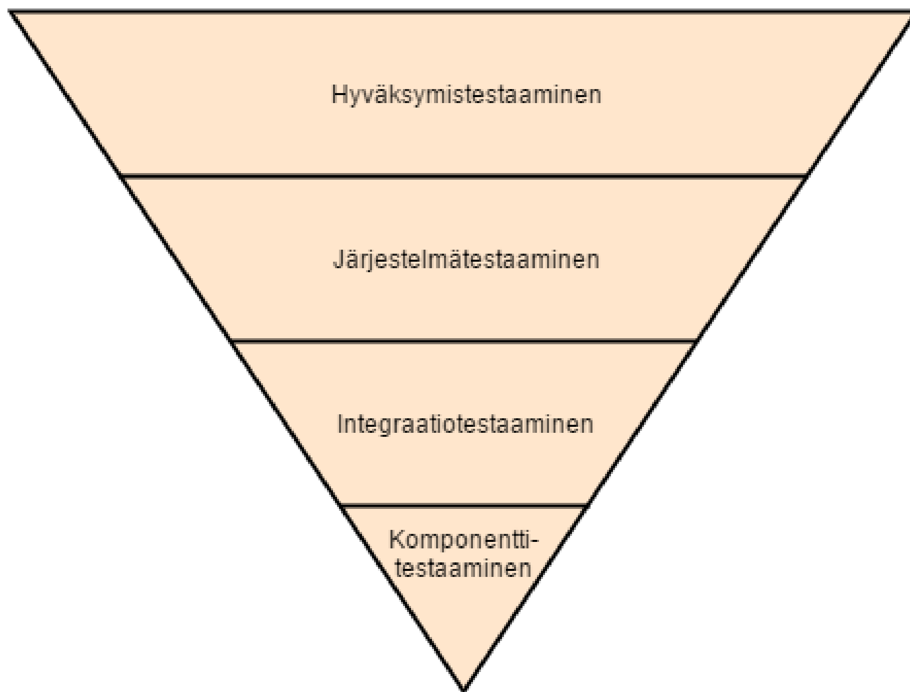
Toteutus- ja suoritusvaiheessa muodostetut testitapaukset ajetaan testattavalle järjestelmälle. Testitapauksista voidaan muodostaa testikokoelmia, joiden avulla testejä on helpompi hallita. Tulosten analysointi- ja raportointivaiheessa tuloksia verrataan ennen testejä asetettuihin odotuksiin toimivuudesta ja niitä verrataan myös aiempiin testeihin, minkä jälkeen tuloksista raportoidaan kehittäjille. (Spillner ym. 2014.)

## 2.2 Testaaminen tuotekehitysprosessissa

Kun tuotetta, kuten ohjelmistoa, kehitetään, on mahdollista, että virheitä tapahtuu niin suunnittelussa kuin varsinaisessa kehitysvaiheessa. Mitä myöhemmässä vaiheessa tuotantoprosessia virhe tapahtuu, sitä hankalampi se on korjata. Kun testaamista tehdään kehitystyön rinnalla alusta lähtien, pystytään mahdolliset virheet havaitsemaan ja korjaamaan ajoissa. Testaamiselle on olemassa joitakin yleisiä periaatteita. Sillä pyritään osoittamaan virheiden olemassaolo, ei niiden poissaoloa. Täysin tyhjentävä testaaminen on mahdotonta, sillä pienimmätkin ohjelmistot voivat sisältää satoja tai jopa tuhansia eri toiminnallisuuksia puhumattakaan valtavista, useista ohjelmistoista koostuvista järjestelmistä. Testaamisen pitäisi alkaa niin aikaisessa vaiheessa kehitystyötä kuin se vain on mahdollista. Viat ja virheet esiintyvät usein ryhmänä eikä saman testin jatkuva uudelleen suorittaminen välttämättä paljasta uusia vikoja. Tästä syystä testitapauksia voi joutua muokkaamaan. Se, että testitapaukset onnistuvat ei välttämättä tarkoita, että testattava ohjelmisto toimii juuri niin kuin sen halutaan toimivan, joten tuloksia ja testien suoritusta tuleekin arvioida tarkkaan. (Spillner ym. 2014.)

Testaamista on useaa eri tyyppiä (kuva 1) ja niitä hyödynnetään ohjelmistotuotantoprosessin eri vaiheissa. **Komponenttitestaaminen** on ensimmäinen testaamisen muoto, joka ohjelmistolle tai järjestelmälle yleensä tehdään. Siinä testataan ohjelmiston yksittäisen osan toimivuutta ja se perustuu komponentin vaatimuksiin. Seuraava muoto on **integraatiotestaaminen**, jossa komponentit on liitetty yhteen ohjelmakokonaisuudeksi ja niiden yhteentoimivuus pyritään varmistamaan. **Järjestelmätestaamisella** testataan ohjelmistoa, järjestelmää tai palvelua niin, että kaikki sen toiminnalliset osat ovat mukana testissä. Aiempien testausten vaatimukset perustuvat teknisiin vaatimuksiin, mutta järjestelmätestaamisen vaatimukset perustuvat asiakkaan asettamiin tuotteen toiminnallisiin vaatimuksiin. (Spillner ym. 2014.)

Kaikkein ylimmän tason testaamisen muoto on **hyväksymistestaaminen**. Se tehdään asiakkaan tai tuotteen loppukäyttäjän näkökulmasta ja joissakin tapauksissa asiakas tekee tämän itse. Hyväksymistestaamista voidaan tehdä myös yksittäisille komponenteille ennen järjestelmätestaamista. (Spillner ym. 2014.) Kehitystyön kannalta tärkeänä testaamisen muotona pidetään myös **regressiotestaamista**. Regressiotestaamista tehdään yleensä aina, kun uusi ohjelmistoversio valmistuu ja sillä testataan, etteivät tehdyt muutokset vaikuta ohjelmiston toimintaan odottamattomasti. Regressiotestaaminen toteutetaan usein automaation avulla. (Gregory & Crispin 2014.)




***Kuva 1.*** *Testaamisen tasot.*

## 2.3 Testaamisen automatisointi

Automaatiolla tarkoitetaan jonkin prosessin itsenäistä toimimista ilman ihmisen vuorovaikutusta. Testaamisen automatisoinnilla pyritään siirtämään koneille sellaisia tehtäviä, jotka ovat hankalia tai aikaa vieviä toteuttaa käsin. Automaatiota tulisi toteuttaa silloin, kun sille on selkeä tarve. Jokaisen testin automatisointi ei välttämättä ole kannattavaa vaan dynaamiset ja monimutkaiset testitapaukset olisi hyvä suorittaa manuaalisesti. Testausprosessista voidaan automatisoida testitapausten lisäksi myös esimerkiksi testausympäristön konfigurointi ja tulosten sekä lokitiedostojen tallentaminen. (Black & Mitchell 2015.)

Automaatio tulisi suunnitella huolella jo testaamisprosessin suunnittelun alkuvaiheessa. Suunnittelussa ja toteutuksessa on tärkeä ajatella pitkäaikaisia vaikutuksia, kuten saavutetaanko automaatiolla merkittävää hyötyä pitkällä aikavälillä ja kannattaako sitä näin ollen lähteä toteuttamaan. Automaation kehys olisi hyvä suunnitella huolella alusta alkaen ja siinä tulisi ottaa huomioon laajennettavuus sekä ylläpidettävyys. (Black & Mitchell 2015.)

## 2.4 Automaation hyödyt

Automaatiolla on mahdollista säästää tehokkaasti aikaa, pienentää testaamiseen käytettävää työtaakkaa ja parantaa testien kattavuutta, jolloin myös tuotannon laatu paranee. Automaatio mahdollistaa testaamisen heti, kun tarvetta ilmenee ja sillä voidaankin toteuttaa testausta esimerkiksi aina, kun uusi ohjelmistoversio valmistuu (Menon 2013, 6). Näin regressiotestaaminen nopeutuu huomattavasti. Myös suorituskyky- ja luotettavuustestausta voi olla hankala toteuttaa ilman automaatiota. (Black & Mitchell 2015.)

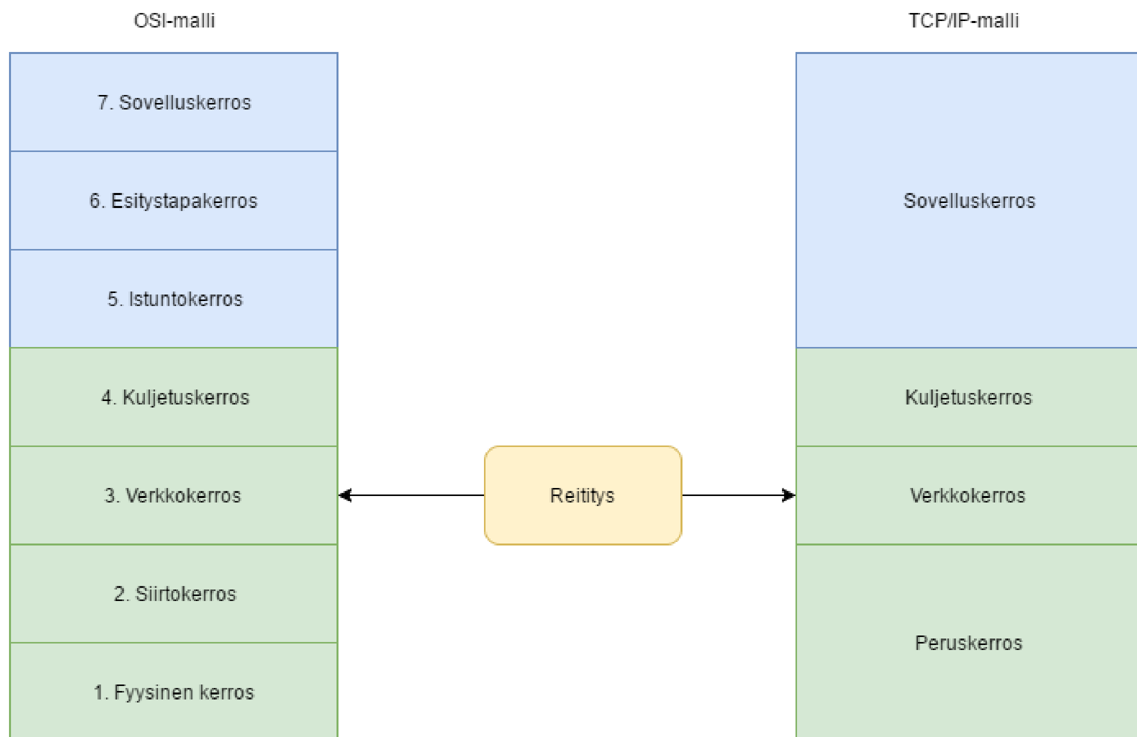
# 3 REITITYS TIETOLIIKENTEESSÄ

## 3.1 Reitittäminen yleisesti

Tietoliikenteessä reitittämisellä tarkoitetaan reitittimen suorittamaa toimenpidettä, missä IP-paketti siirretään IP-verkosta toiseen kunnes paketti on saavuttanut määränpäänsä. Reititystä tapahtuu silloin, kun paketin määränpää sijaitsee sen lähdeverkon ulkopuolella. Paketti ohjataan reitittimelle, joka hoitaa tarvittavan prosessoinnin ja välittää paketin eteenpäin kohti määränpäätä. Reitittimeksi kutsutaan verkon aktiivilaitetta, joka yhdistää IP-verkkoja toisiinsa. Reitittimellä on tieto muista verkoista sen lähistöllä, minkä avulla se voi tehdä päätökset paketin reitittämisestä. (Macfarlane 2006, 70-83.)

Reitittimet käyttävät reititystaulua reittien ja verkkojen hallintaan. Reititystaulu sisältää listan reitittimen omista sekä muilta reitittimiltä opituista verkoista ja reiteistä, joiden kautta eri verkot ovat saavutettavissa. Reititystaulu sisältää tiedon verkoista ja seuraavasta hypystä (*next-hop*), jonka kautta verkko on saavutettavissa. Reitityksessä hypyllä tarkoitetaan tapahtumaa, jossa paketti saapuu reitittimelle, josta se välitetään toiselle reitittimelle. Paketti kulkee eli ”hyppää” yhden reitittimen yli. Reitittimistä puhuttaessa käytetään usein termiä solmu (*node*), sillä reitittimet ovat ikään kuin verkossa olevia solmukohtia, joiden kautta tietoliikenne voi jatkaa useampaan eri suuntaan. (Macfarlane 2006, 70-83.)

Tietoliikenteen ja siihen liittyvien protokollien hahmottamiseen käytetään usein kahta erilaista mallia: OSI-mallia ja TCP/IP-mallia. Näistä TCP/IP on malli, jota käytetään oikeissa toteutuksissa, kuten Internetissä. OSI-malli puolestaan on täysin teoreettinen. Silti OSI-malli yleensä toimii pohjana ja vertailukohtana lähes kaikille käytössä oleville protokollakokoelmille. TCP/IP-mallissa ei määritellä, mitä verkkokerroksen alapuolella tarkalleen on. Kuvasta 2 voidaan havaita TCP/IP- ja OSI-mallin rakenne. (Kabelová & Dostálek 2006, 7-23.) Reititys tapahtuu OSI-mallin kolmannella kerroksella (Blank 2006, 26).




***Kuva 2.*** *OSI- ja TCP/IP-malli. (Kabelová & Dostálek 2006, 7.)*

Reititykseen kuuluu kappaleessa 3.5 käsiteltävien reititysprotokollien lisäksi myös muita protokollia, joiden tarkoitus on kuljettaa tietoa määränpäähän reititysprotokollia hyväksi käyttäen. Tällaisia protokollia ovat esimerkiksi IP, TCP ja UDP. IP (*Internet Protocol*) on verkkokerroksella toimiva protokolla, joka käyttää IP-osoitetta paketin määränpään määrittämiseen. Paketit voivat kulkea verkossa eri reittejä ja saapua kohteeseen missä järjestyksessä tahansa, jossa ne järjestetään uudelleen alkuperäiseen järjestykseen datan esitystä varten. TCP (*Transmission Control Protocol*) ja UDP (*User Datagram Protocol*) ovat kuljetuskerroksella toimivia protokollia. TCP on yhteydellinen protokolla, mikä tarkoittaa, että protokolla varmistaa paketin perille pääsyn, ja jos paketti ei päässyt perille, lähetetään se uudelleen. Näin ollen TCP on luotettava tiedonsiirtoprotokolla. UDP-protokollaa käytettäessä, paketti lähetetään välittämättä siitä, pääseekö se koskaan perille. Tästä syystä UDP on epäluotettava tiedonsiirtoprotokolla. (Kabelová & Dostálek 2006, 14.) Edellä mainitut protokollat käytännössä pakkaavat lähetettävän tiedon, jotta ne voidaan reitityksen avulla saattaa perille määränpäähänsä.

## 3.2 Staattinen reititys

Staattisella reitityksellä tarkoitetaan reittien asettamista käsin eli reititin ei ole oppinut kyseisiä reittejä muilta reitittimiltä minkään reititysprotokollan välityksellä. Staattisten reittien asettaminen voi olla työlästä ja niitä voi olla hankala hallita, jos on kyse isosta verkosta. Staattisten reittien huonona puolena on myös se, että verkkotopologian muuttuessa ne lakkaavat toimimasta ja tällöin reitit täytyy määritellä uudelleen uuden topologian mukaisesti. Tästä syystä staattisia reittejä käytetäänkin lähinnä erityistapauksissa. (Macfarlane 2006, 89-100.)

Staattisia reittejä käytetään usein pienissä verkoissa, jotka pysyvät suhteellisen muuttumattomina. Toisin kuin dynaaminen reititys, staattisten reittien käyttäminen ei kuluta ylimääräistä kaistaa verkon muulta liikenteeltä, kun reittejä ei mainosteta toisille reitittimille minkään protokollan kautta. Staattiset reitit eivät myöskään kuluta resursseja, kuten muistia ja prosessorin käyttöaikaa reitittimeltä, koska niiden muuttumattomuudesta johtuen niitä ei tarvitse prosessoida kuten reititysprotokollien kautta opittuja dynaamisia reittejä. Staattista reititystä käytetään usein dynaamisen reitityksen rinnalla esimerkiksi määrittelemään oletusyhdyskäytävä. (Macfarlane 2006, 89-100.)

Staattiset reitit jaetaan usein eteenpäin muille reitittimille dynaamisen reititysprotokollan ollessa käytössä verkossa. Näin staattista reittiä ei tarvitse asettaa jokaiselle reitittimelle erikseen vaan riittää, että se on asetettu yhdelle laitteelle, joka kertoo sen eteenpäin muille laitteille. Kelluvaksi staattiseksi reitiksi sanotaan staattista reittiä, joka ei ole koko ajan käytössä vaan voi toimia esimerkiksi varareittinä, jolloin se otetaan automaattisesti käyttöön dynaamisesti opitun reitin kadotessa. (Macfarlane 2006, 89-100.)

## 3.3 Dynaaminen reititys

Kun reititin oppii reittejä automaattisesti muilta reitittimiltä ilman, että niitä tarvitsee asettaa käsin, on kyseessä dynaaminen reititys. Dynaamista reititystä käytetään eteenkin isoissa verkkoympäristöissä sen automaattisuuden ja joustavuuden vuoksi. Dynaaminen reititys sallii verkon topologian muuttumisen ilman, että verkkoliikenteen kulku häiriintyy merkittävästi. Verkkoa voidaan myös helposti laajentaa ilman suurempaa vaivaa reitityksen konfiguroinnissa. (Macfarlane 2006, 103-116.)

Dynaamisen reitityksen periaatteena on, että liikenne reititetään automaattisesti kulkemaan parasta mahdollista reittiä. Jos reitti katoaa esimerkiksi reitittimen hajoamisen tai yhteyden katkeamisen seurauksena, laskevat reitittimet automaattisesti uuden parhaan reitin. Kun edellinen yhteys palautuu, reititetään liikenne jälleen kulkemaan vanhaa reittiä olettaen, että se on edelleen paras mahdollinen. Dynaaminen reititys toteutetaan käyttämällä jotain reititysprotokollaa välittämään ja vastaanottamaan reititystietoa muilta reitittimiltä. (Macfarlane 2006 103-116.)

## 3.4 Ryhmälähetys

Yleensä tietoliikenne kulkee verkossa täsmälähetyksenä (*unicast*), jolloin paketti välitetään yhdestä lähdeosoitteesta yhteen kohdeosoitteeseen. Jotkin palvelut, kuten videolähetykset ja IP-puhelut, voivat vaatia, että liikennettä lähetetään useaan kohdeosoitteeseen, mikä hoidetaan käyttämällä täsmälähetyksen sijaan ryhmälähetystä (*multicast*). Ryhmälähetys on yleislähetyksen (*broadcast*) muoto, jossa paketteja ei kuitenkaan lähetetä kaikille verkon laitteille vaan ne reititetään sen mukaan, mikä laite on tilannut kyseistä liikennettä. Laiteet tilaavat liikennettä liittymällä johonkin ryhmälähetysryhmään käyttämällä IP-osoitetta väliltä 224.0.0.0–239.255.255.255. Ryhmälähetysliikennettä hallitaan IGMP-protokollalla (*Internet Group Management Protocol*). Ryhmälähetysliikennettä tilannut laite lähettää säännöllisesti IGMP-paketteja verkkoon, joiden avulla reitittimet tietävät, mihin ryhmiin kuuluvaa liikennettä verkkoon tulisi reitittää. (Kabelová & Dostálek 2006.)

## 3.5 Reititysprotokollat

Reititysprotokolla on protokolla, joka sallii reitittimien jakaa keskenään tietoa verkoista, joista ne tietävät. Reititysprotokollia on olemassa lukuisia eri käyttötarkoituksiin. Tässä opinnäytetyössä ei paneuduta kaikkiin olemassa oleviin reititysprotokolliin vaan esitellään niistä yleisimmin käytetyt. Yleisesti reititysprotokollat toimivat mainostamalla omia reittejään ja vastaanottamalla reititysmainostuksia muilta reitittimiltä ja asettamalla opitut reitit omaan reititystauluunsa. Reititystauluun lisätään ja siitä poistetaan reittejä

sen mukaan, miten verkko muuttuu ja miten reitittimet mainostavat tietämiään reittejä. (Macfarlane 2006, 103-116.)

Reititysprotokollat jaetaan sisäisiin (*IGP, Internal Gateway Protocol*) ja ulkoisiin (*EGP, Exterior Gateway Protocol*) reititysprotokolliin. Sisäisellä reititysprotokollalla tarkoitetaan autonomisen järjestelmän (*AS, autonomous system*) eli yhden toimijan, esimerkiksi yrityksen, verkkoympäristön sisäisen reitityksen hoitavaa protokollaa. Sitä ei ole tarkoitettu reitittämään liikennettä muiden toimijoiden verkkoihin vaan sen hoitaa ulkoinen reititysprotokolla. (Macfarlane 2006, 103-116.)

Reititysprotokollat luokitellaan myös yhteystila- (*link state*) ja etäisyysvektoriprotokolliin (*distance vector*). Etäisyysvektoriprotokollien lähettämä reititystieto sisältää vain tiedon muiden solmujen tietämistä verkoista, muttei tietoa aktiivisista reitittimistä verkossa. Yhteystilaprotokollat sen sijaan lähettävät niin sanottuja yhteystilamainostuksia (*LSA, Link-State Advertisement*) kaikille verkon reitittimille toisin kuin etäisyysvektoriprotokollissa, missä vain naapurisolmut vaihtavat tietoa keskenään. Yhteystilamainostukset sisältävät tietoa reitittimen suorien yhteyksien tilasta, kuten IP-osoitteen ja yhteyden nopeuden. Yhteystilaprotokollat mahdollistavat reitittimien tietää muiden reitittimien määrän verkossa, sekä mitä verkkoja reitittimiin yhdistyy. (Macfarlane 2006, 103-116.)

### 3.5.1 OSPF

OSPF (*Open Shortest Path First*) on yksi yleisimmin käytetyistä avoimen standardin sisäisistä reititysprotokollista. Suurin osa markkinoilla olevista kaupallisista reitittimistä sisältää tuen OSPF:lle. Se on laajennettava, hyvin skaalautuva ja sitä voidaankin käyttää suurissa verkkoympäristöissä. Siinä ei ole rajoitettua verkkoliikenteen hyppyjen enimmäismäärää reitityssolmujen yli ja sen reititysmainostukset eivät kuluta paljoa verkon kaistaa. OSPF on nopea saavuttamaan verkon tasapainoisen tilan. Se ei ole taipuvainen reitityssilmukoille. Reitityssilmukalla tarkoitetaan tilannetta, jossa verkkoliikenne jää kiertämään kehää joidenkin reitittimien välille pääsemättä koskaan määränpäähänsä. (Macfarlane 2006, 221-225.)

OSPF on kehitetty korvaamaan RIP-protokolla, joka käsitellään kappaleessa 3.5.3. OSPF:ää käytetään pääsääntöisesti yritysten sisäisissä verkoissa, mutta myös jotkin palveluntarjoajat käyttävät sitä runkoverkon reititykseen. Se on alusta alkaen suunniteltu Internet-reititysprotokollaksi ja se sisältääkin erityisiä laajennuksia BGP-protokollalta (käsitellään kappaleessa 3.5.2) opittujen reittien käsittelyyn. OSPF muodostaa naapuruuksia muihin solmuihin niin sanottujen *hello*-pakettien avulla. OSPF tukee hierarkkista reititysympäristöä, jossa verkko on mahdollista jakaa useampaan protokollan sisäiseen alueeseen, mikä vähentää reitityspäivitysten määrää verkossa. Hierarkia on kaksitasoinen ja sisältää runkoalueen sekä toissijaiset alueet, jotka liittyvät runkoalueeseen. OSPF tukee myös reitityspakettien valinnaista todentamista, mikä lisää reitityksen tietoturvaa. (Macfarlane 2006, 221-225.)

### 3.5.2 BGP

BGP (*Border Gateway Protocol*) on nykyisin ainoa käytössä oleva ulkoinen reititysprotokolla. Se on tarkoitettu välittämään reititystietoa autonomisten järjestelmien välillä ja sitä käyttävätkin pääasiassa palveluntarjoajat sekä jotkin yritykset muodostamaan reittejä useisiin palveluntarjoajiin. Käytännössä BGP on protokolla, joka muodostaa Internetin. BGP luokitellaan polkuvektoriprotokollaksi (*path vector protocol*) ja etäisyysvektoriprotokollista poiketen, se ei mainosta tietämiään verkkoja, vaan pitää listaa autonomisista järjestelmistä, joiden läpi tietoliikenteen on kuljettava päästäkseen määränpäähänsä. (Macfarlane 2006, 346-349.)

BGP asettaa erilaisia määritteitä oppimilleen reiteille, mikä mahdollistaa reitityskäytäntöjen asettamisen. Reitityskäytännöillä voidaan hallita, miten reitit jakautuvat palveluntarjoajien kesken sekä palveluntarjoajan ja asiakkaan välillä. Näin voidaan myös valita, mitkä reitit sallitaan tai hylätään, ja mitkä reitit ovat ensisijaisia tiettyihin verkkoihin. BGP on hierarkkinen protokolla, joka voi toimia kahdessa eri tilassa: ulkoisena (*EBGP, External BGP*) tai sisäisenä BGP:nä (*IBGP, Internal BGP*). Toimintatilojen erot ovat vähäiset, mutta esimerkiksi sisäisen BGP:n reitit ovat ensisijaisia samoihin ulkoiselta BGP:ltä opittuihin reitteihin nähden. BGP:n hierarkia ei ole yhtä selkeä, kuin OSPF:ssä, mutta autonomisten järjestelmien voidaan ajatella muodostavan hierarkian, missä järjestelmän sisällä käytetään sisäistä BGP:tä ja järjestelmien välillä ulkoista BGP:tä. (Macfarlane 2006, 346-349, 358-359.)

### 3.5.3 RIP

RIP (*Routing Information Protocol*) on yksi vanhimmista edelleen käytössä olevista sisäisistä reititysprotokollista. Se on ei-hierarkinen etäisyysvektoriprotokolla ja sitä voidaan käyttää vain pieniin verkkoympäristöihin, sillä sen hyppyjen enimmäismäärä on rajoitettu 15 reitityssolmuun. Se on myös hidas muodostamaan vakaan verkon ja taipuvainen reitityssilmukoille. RIP sopii verkkoihin, joissa ei ole useita rinnakkaisia yhteyksiä, ja joiden yhteysnopeudet ovat samat. RIP:n etuna pidetään sen helppoa konfiguroitavuutta. RIP toimii siten, että se lähettää tietyin väliajoin koko reititystaulunsa sisällön naapurisolmulle, joka lisää siitä ennestään tuntemattomat reitit omaan reititystauluunsa ja jakaa sitten oman reititystaulunsa sisällön eteenpäin omille naapurisolmuilleen. Reititys perustuu ainoastaan hyppyjen määrään eikä muita verkon ominaisuuksia, kuten yhteyksien luotettavuutta ja laatua oteta huomioon. (Macfarlane 2006, 137-142.)

### 3.5.4 EIGRP

EIGRP (*Enhanced Interior Gateway Routing Protocol*) on Ciscon omistama sisäinen reititysprotokolla. Se on luonteelta sekamuotoinen etäisyysvektoriprotokolla, sillä siinä on sekä yhteystila- että etäisyysvektoriprotokollan ominaisuuksia. OSPF:n lailla EIGRP:tä on hyvän skaalautuvuuden vuoksi mahdollista käyttää suurissa verkoissa. Siinä ei ole OSPF:n tapaan rajoitettua hyppyjen määrää reitityssolmujen yli, se on nopea muodostamaan tasapainoisen verkon eikä ole taipuvainen reitityssilmukoille. EIGRP ei ole hierarkkinen protokolla, joten OSPF:stä poiketen, sitä ei ole mahdollista jakaa alueisiin. (Macfarlane 2006, 185-192.)

EIGRP on rakennettu kuluttamaan vähän verkon kaistaa. Se käyttää viittä eri viestityyppiä reitityksen toteuttamiseen. Hello-pakettien avulla protokolla muodostaa naapuruussuhteita muihin reitittimiin, päivityspaketeilla jaetaan varsinaista reititystietoa reitittimien välillä, tiedustelupaketeilla toteutetaan kysely silloin, kun havaitaan puuttuva reitti, vastauspaketilla protokolla vastaa tiedusteluun ja vahvistuspaketilla lähetetään vastaus RTP-protokollalle (*Reliable Transport Protocol*), jota EIGRP käyttää pakettien luotettavaan toimittamiseen. EIGRP:ssä on mahdollista toteuttaa reitityspakettien tunnistaminen, reitityksen tietoturvan lisäämiseksi. (Macfarlane 2006, 185-192.)

### 3.5.5 IS-IS

IS-IS (*Intermediate System-to-Intermediate System*) on alun perin OSI-malliin pohjautuvan CNLP-protokollan (*Connectionless Network Protocol*) reitittämistä varten kehitetty sisäinen yhteystilaprotokolla. Se on hierarkkinen ja sillä voidaan reitittää myös IP-liikennettä. IS-IS jaetaan OSPF:n tapaan alueisiin ja se käyttää aluejaossa kaksitasoista hierarkiaa. Runkoalueen reitittimiä kutsutaan tason kaksi reitittimiksi (*L2 router*) ja alemman tason alueen reitittimiä tason yksi reitittimiksi (*L1 router*). Alemman tason alueella on pakko olla vähintään yksi reititin, joka on kytköksissä runkoalueeseen, mutta OSPF:stä poiketen reititin ei sijaitse alueiden rajalla vaan runkoalue on yhdistetty linkin kautta. OSPF:n lailla myös IS-IS käyttää hello-paketteja naapuruuksien ylläpitämiseen. (Medhi, Ramasay & Zupan 2010, 185.) IS-IS soveltuu erittäin laajojen verkkojen reititykseen ja sitä käyttävätkin pääasiassa suuret palveluntarjoajat sisäisessä reitityksessään. (Anand & Chakrabarty 2004, 299.)

### 3.5.6 OLSR

OLSR (*Optimized Link State Routing*) on liikkuvissa langattomissa ad-hoc -verkoissa (*MANET, Mobile ad-hoc Network*) käytetty reititysprotokolla. MANET-verkolla tarkoitetaan liikkuvaa langatonta verkkoa, jossa reityssolmut keskustelevat suoraan toistensa kanssa ilman erillistä tukiasemaa. Se on itsenäisesti muodostuva ja itseään korjaava verkko, missä topologia voi muuttua nopeasti ja odottamattomasti. (Johnson, Ntlatlapa & Aichele 2016.)

OLSR on luonteeltaan **ennakoiva yhteystilaprotokolla**. Ennakoinnilla tarkoitetaan sitä, että protokolla lähettää ajoittain hallintaviestejä verkkoon, joiden avulla reitityssolmut pitävät yllä tietoa verkon rakenteesta ja näin ollen tarvittavat reitit ovat heti käytössä, kun niitä tarvitaan. Protokolla pitää jatkuvasti yllä tietoa reiteistä kaikkiin kohdeverkkoihin. Ennakoivan protokollan vastakohta on **reaktiivinen protokolla**, joka etsii reittejä kohteeseen vain silloin, kun tarve vaatii, joten reitityksessä esiintyy viivettä. Reaktiiviseen protokollaan verrattuna, ennakoivan protokollan reititys tapahtuu lähes viiveettömästi, mutta hallintaviestien takia verkon kuormitus on hieman suurempi. OLSR käyttää paranneltua yhteystilaprotokollaa, minkä etuna on, että viestit ovat tavallista pie-

nempiä, kun kaikkien yhteyksien tietoja ei lähetetä yhdellä kertaa. Viestejä ei myöskään lähetetä koko verkkoon vaan niin kutsutuille *multipoint relay* -naapureille, jotka lähettävät viestejä edelleen omille *multipoint relay* -naapureilleen. Näin verkon kuormitusta pyritään pienentämään. (Jacquet ym. 2016, 1-2.)

OLSR soveltuu suuriin ja tiheisiin verkkoihin, joissa sen paranneltu yhteystila-algoritmi toimii sitä paremmin, mitä suurempi ja tiheämpi verkko on, verrattuna normaaliin yhteystila-algoritmiin. Se toimii täysin hajautetusti eikä vaadi luotettavaa tiedonsiirtoa. Tästä syystä OLSR soveltuu hyvin käytettäväksi radioyhteyksellisissä verkoissa, missä osa paketeista saattaa hävitä tiedonsiirron aikana. OLSR sallii solmujen liikkumisen ja muuttuvaa topologiaa voidaan seurata hallintaviestien avulla. (Jacquet ym. 2016, 1-2.)

### 3.5.7 B.A.T.M.A.N

B.A.T.M.A.N (*Better Approach to Mobile Ad-hoc Networking*) on OLSR:n lailla MANET-verkkoja varten kehitetty reititysprotokolla. Se on kehitetty paikkaamaan OLSR-protokollan puutteet. Merkittäviä puutteita OLSR:ssä on esimerkiksi reitityksen ajoittainen epäluotettavuus ja joissakin tapauksissa aiheutuvat reitityssilmukat, joiden takia suorituskyky laskee. Reitityssilmukoiden seurauksena reititystaulut voivat tyhjentyä, jolloin reittejä katoaa odottamattomasti. Vaikka OLSR on tarkoitettu suuriin verkkoihin, sen tehokkuus laskee, jos verkko muodostuu liian suureksi. Esimerkiksi yli 300 solmun kokoinen verkko voi aiheuttaa ongelmia. (Johnson ym. 2016.)

B.A.T.M.A.N.-protokolla toimii lähettämällä hello-paketteja yleislähetyksenä naapurisolmuilleen. Hello-paketti sisältää alkuperäisen lähettäjän osoitteen, paketin senhetkisen lähettäjän osoitteen sekä yksilöidyn järjestysnumeron. Jokainen paketin vastaanottanut naapurisolmu korvaa senhetkisen lähettäjän osoitteen omalla osoitteellaan ja lähettää paketin uudelleen yleislähetyksenä omille naapureilleen. Kun alkuperäinen paketin lähettäjä vastaanottaa oman pakettinsa takaisin, tarkastaa se, että yhteys, josta paketti vastaanotettiin, on mahdollista käyttää molempiin suuntiin. Protokollalla ei ole tietoa kokonaisista reiteistä kohdeosoitteisiin vaan se ylläpitää tietoa yhteyksistä naapurisolmuihin, joiden kautta löytyy paras reitti kohteeseen. B.A.T.M.A.N. ei tarkista yhteyksien laatua vaan pelkästään niiden olemassaolon. (Johnson ym. 2016.)

# 4 ROBOT FRAMEWORK

## 4.1 Yleisesti

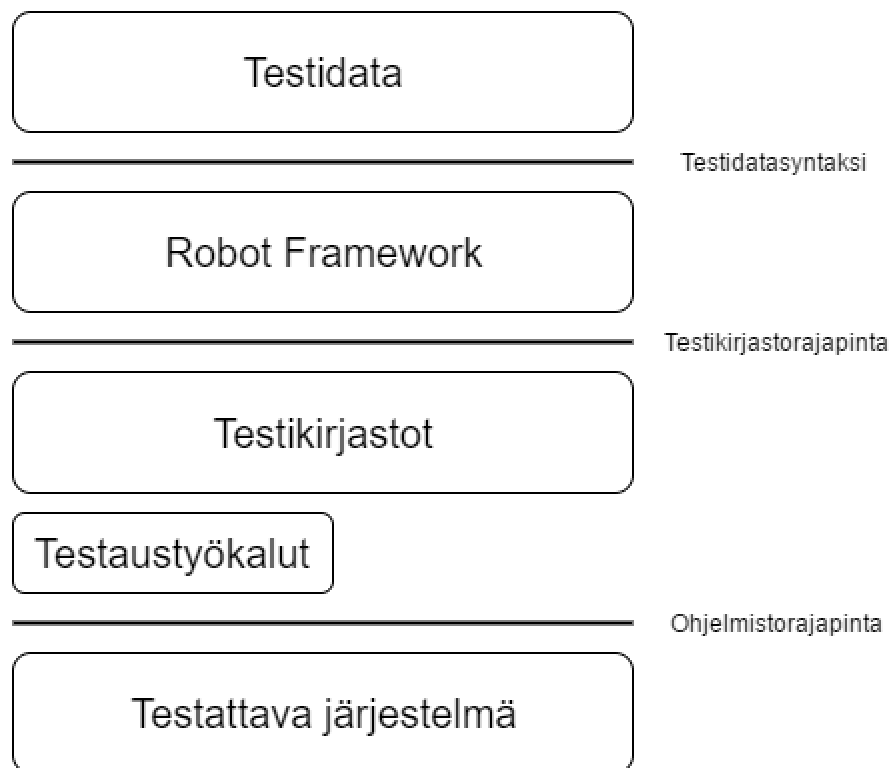
Robot Framework on yleispätevä, hyväksyntätestaamiseen kehitetty avainsanakäyttöinen testausautomaatiokehys. Se on avointa lähdekoodia ja pohjautuu Python-ohjelmointikieleen. Sitä voidaan käyttää myös Pythonin Javaan perustuvalla toteutuksella Jythonilla sekä C#-kieleen perustuvalla IronPythonilla. Robot Framework on helposti laajennettavissa lukuisilla valmiilla sekä mukautetuilla testikirjastoilla, eikä se ole käyttöjärjestelmä- tai ohjelmistoriippuvainen. Sitä voidaan käyttää useilla eri alustoilla, ja sillä voidaan testata monia erityyppisiä ohjelmistoja ja järjestelmiä. (Robot Framework User Guide 2016.)

Robot Frameworkin on kehittänyt suomalainen Pekka Klärck osana diplomityötään Nokia Siemens Networksilla vuonna 2005. Sen ensimmäinen virallinen julkaisu oli versio 2 Apache-lisenssin alla vuonna 2008. Ohjelmistolla on laaja ekosysteemi ja aktiivinen yhteisö, joka ylläpitää ja kehittää testikirjastoja sekä Robot Frameworkiin liittyviä työkaluja. Suurin osa testikirjastoista ja työkaluista ovat ydinkehyksen lailla avointa lähdekoodia. (Bisht 2013, 9-14; Robot Framework User Guide 2016.)

## 4.2 Asentaminen ja toiminta

Toimiakseen Robot Framework vaatii tietokoneelle joko Pythonin version 2 tai 3. Jythonia käytettäessä täytyy tietokoneelle asentaa Java-alusta. Jos Robot Frameworkia käytetään IronPythonilla, vaaditaan tietokoneelle .NET-alusta. Tässä opinnäytetyössä keskitytään vain Robot Frameworkin käyttöön Pythonilla, tarkemmin Pythonin versiolla 2.7.12. Robot Framework asennetaan lataamalla lähdekoodi ja ajamalla lähdekoodin hakemistossa komento ”*python setup.py install*” tai vaihtoehtoisesti ”*python install.py in*” (Bisht 2013, 9-14). Se voidaan asentaa myös Pythonin pip-paketinhallintasovelluksen kautta ajamalla komento ”*pip install robotframework*”. Kehittäjä suosittelee käyttämään Robot Frameworkin asentamiseen pip-paketinhallintaa. (Robot Framework User Guide 2016.)

Robot Frameworkin toiminta perustuu kuvassa 3 kuvattuun arkkitehtuuriin, missä ylimpänä ovat käyttäjä ja käyttäjän kirjoittamat testitapaukset. Kun Robot Frameworkilla ajetaan testausta, Robot Framework -kehys käsittelee ja suorittaa käyttäjän tekemät testitapaukset ja muodostaa testeistä lokit ja tulosraportit. Robot Framework -kehys itsessään ei ole tietoinen testattavasta kohteesta vaan se välittää komentoja ja vastaanottaa dataa testikirjastoilta. Testikirjastot voivat vaikuttaa testattavaan järjestelemään joko suoraan tai käyttämällä alemman tason testityökaluja testaamisen välineenä. Alimpana arkkitehtuurissa on testattava järjestelmä. (Robot Framework User Guide 2016.)




***Kuva 3.*** *Robot Frameworkin arkkitehtuuri. (Robot Framework User Guide 2016.)*

### 4.3 Testien ohjelmointi

Robot Frameworkin suoritettavia testikokonaisuuksia kutsutaan testidataksi. Testidata muodostuu tekstitiedostoista, joihin testitapauksia ohjelmoidaan. Testitapauksia voidaan ohjelmoida millä tahansa tekstieditorilla ja monille ohjelmointiympäristöille löytyy myös laajennusosa Robot Frameworkin syntaksia varten. Testitapaukset kirjoitetaan Robot Frameworkin omalla avainsanoihin perustuvalla kielellä. Avainsanoja voi luoda lisää yhdistelemällä olemassa olevia avainsanoja tai luomalla uusia avainsanoja käyttä-

mällä Python-ohjelmoituja mukautettuja kirjastoja. (Robot Framework User Guide 2016.)

### 4.3.1 Testitapaus

Testitapaus (*test case*) tarkoittaa yksittäistä asiaa tai ominaisuutta testaavaa testiä. Esimerkiksi naapurisolmun saavutettavuuden mittaaminen ping-työkalulla voidaan määritellä yhdeksi testitapaukseksi. Testitapaukset tulisi selkeyden vuoksi pitää lyhyinä ja yksinkertaisina:

```
*** Test Cases ***
Ping Test
    Ping To Neighbor
```

Monimutkainen toiminnallisuus olisi hyvä ohjelmoida mukautettuun testikirjastoon Pythonia käyttäen:

```
def ping_to_neighbor(self):
    self.con.send_command('ping 10.0.0.1')
    data = self.con.read_serial()
    if '!' in data:
        logger.console('Ping test successful')
    else:
        logger.console('Ping test unsuccessful')
```

Testitapauksia voidaan ohjelmoida joko käyttämällä tavallista pelkistettyä tekstiä, verkkosivujen tekemiseen käytettävää HTML-muotoa, taulukkolaskentasovelluksiin sopivaa TSV-muotoa tai dokumentoinnissa käytettävää reStructuredText-muotoa. Useimmissa tapauksissa testien kirjoittaminen tavalliseen tekstimuotoon on kannattavinta, sillä sitä on helppo lukea, hallita ja se soveltuu myös käytettäväksi esimerkiksi versionhallintaan. (Robot Framework User Guide 2016.)

### 4.3.2 Testikokoelma

Yksittäinen testitiedosto muodostaa testikokoelman (*test suite*), joka kokoaa yhteen samaa ominaisuutta testaavat testitapaukset yhdeksi kokonaisuudeksi. Testikokoelmassa

voidaan määritellä kaikille testitapauksille yleiset lähtö- sekä lopetustilanteet. Testikokoelmassa voidaan määritellä myös asetukset testitapauksille, kuten muuttuja- ja lähdetiedostot sekä testeissä käytettävät mukautetut kirjastot.

Kuvan 4 esimerkistä voidaan nähdä, kuinka testikokoelma voidaan jakaa neljään eri osaan. Alussa olevassa asetukset (*settings*) -osassa voidaan määritellä, mitä kirjastoja tai lähdetiedostoja testikokoelmassa käytetään. Siinä on myös mahdollista määritellä testikokoelmalle yleiset alku- (*setup*) ja lopetustoimet (*teardown*). Muutujat (*variables*) -osassa määritellään testeissä käytettävät muuttujat. Muuttujat voidaan määritellä myös erilliseen tiedostoon, mikä otetaan käyttöön asetusten määrittelyllä. Avainsanat (*keywords*) –osassa käyttäjä voi rakentaa uusia avainsanoja yhdistelemällä olemassa olevia avainsanoja toisiinsa. Viimeisenä osana ovat suoritettavat testitapaukset (*test cases*). Kuvasta voidaan havaita, että varsinaiset testitapaukset voidaan pitää selkeinä ja yksinkertaisina määrittelemällä korkeamman tason avainsanoja aiemassa Avainsanat-osiossa.

```
*** Settings ***
Library   MyCustomLibrary

*** Variables ***
${neighbor_address}      10.0.0.1
${file_name}             round-trip.txt

*** Keywords ***
Ping And Save Round Trip Time
    Ping To Neighbor                  ${neighbor_address}
    Save Round Trip Time To File      ${file_name}

*** Test Cases ***
Test Connectivity
    Ping And Save Round Trip Time
```

***Kuva 4.*** *Esimerkki testikokoelmatiedostosta.*

### 4.4 Testien suorittaminen

Testejä suoritetaan komentoriviltä käyttämällä komentoa *pybot [testikokoelma].txt*, jossa *[testikokoelma]* on suoritettavan tiedoston nimi. Suoritettavat tiedosto voivat olla joko ROBOT- tai TXT-tiedostomuodossa. Robot Frameworkin versiossa 3 käytetään *pybot*-komennon sijaan *robot*-komentoa. Myös testikokoelmia sisältävä hakemisto on mahdollista suorittaa kokonaisuudessaan sijoittamalla suoritettavan tiedoston nimen

tilalle suoritettavan hakemiston nimi. Testin suorituksen päätteeksi Robot Framework muodostaa testeistä HTML-muotoiset loki- ja raporttitiedostot, joita voidaan tarkastella verkkoselaimella.

## 4.5 Muut testaamisen työvälineet

Vaikka Robot Framework onkin testauskehys, se itsessään sisältää hyvin vähän ominaisuuksia verkkoympäristöjen testaamiseen. Sen kanssa onkin usein käytettävä muita verkon testaamiseen ja tarkkailuun kehitettyjä ohjelmia, joiden toimintaa automatisoidaan ja joiden lähettämää tietoa kerätään Robot Frameworkin kautta.

### 4.5.1 Iperf

Iperf (kuva 5) on verkon siirtonopeuksien mittaamista varten kehitetty testaussovellus. Se on saatavilla usealle eri alustalle, kuten Windowsille, Linuxille ja Applen macOS:lle. Iperf-ohjelmalla on mahdollista luoda TCP ja UDP liikennettä. Sillä on täsmälähetysliikenteen lisäksi mahdollista testata ryhmälähetysliikennettä ja testata myös useaa yhteyttä samaan aikaan. (Linux manual pages 2016.)

```
testuser@testpc:~$ iperf -u -c 10.0.2.2 -i 1
------------------------------------------------------------
Client connecting to 10.0.2.2, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.2.15 port 36518 connected with 10.0.2.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  64.6 KBytes   529 Kbits/sec
[  3]  1.0- 2.0 sec  64.6 KBytes   529 Kbits/sec
[  3]  2.0- 3.0 sec  63.1 KBytes   517 Kbits/sec
[  3]  3.0- 4.0 sec  64.6 KBytes   529 Kbits/sec
[  3]  4.0- 5.0 sec  63.1 KBytes   517 Kbits/sec
[  3]  5.0- 6.0 sec  64.6 KBytes   529 Kbits/sec
[  3]  6.0- 7.0 sec  64.6 KBytes   529 Kbits/sec
[  3]  7.0- 8.0 sec  63.1 KBytes   517 Kbits/sec
[  3]  8.0- 9.0 sec  64.6 KBytes   529 Kbits/sec
[  3]  9.0-10.0 sec  63.1 KBytes   517 Kbits/sec
[  3]  0.0-10.0 sec   641 KBytes   525 Kbits/sec
[  3] Sent 893 datagrams
```

***Kuva 5.*** *Esimerkki Iperf-ohjelman käytöstä.*

### 4.5.2 Paketin analysointi

Pakettien kaappamiseen ja analysoimiseen käytettäviä sovelluksia ovat esimerkiksi Wireshark ja tcpdump (kuva 6). Niillä on mahdollista seurata eri rajapintojen ja yhteyksien läpi kulkevaa liikennettä ja kerätä tietoa esimerkiksi siitä, minkä tyyppistä tietoliikennettä rajapinnan kautta kulkee. (Linux manual pages 2016.)

```
testuser@testpc:~$ sudo tcpdump -n -c 4 -i enp0s3
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
19:17:38.431804 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 4658, seq 109, len
gth 64
19:17:38.472437 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 4658, seq 109, lengt
h 64
19:17:39.433520 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 4658, seq 110, len
gth 64
19:17:39.471775 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 4658, seq 110, lengt
h 64
4 packets captured
4 packets received by filter
0 packets dropped by kernel
```

***Kuva 6.*** *Esimerkki tcpdump-työkalun käytöstä.*

### 4.5.3 Ping

Ping (kuva 7) on lähes jokaisesta tietokoneesta ja verkkolaitteesta löytyvä TCP/IP-protokollan työkalu, millä mitataan verkkolaitteiden saavutettavuutta. Se käyttää saavutettavuuden selvittämiseen ICMP-protokollaa (*Internet Control Message Protocol*), jolla se lähettää vastauspyynnön haluttuun IP-osoitteeseen, ja jos osoite on saavutettavissa, lähettää sen omistava laite vastausviesti takaisin. (Linux manual pages 2016.)

```
testuser@testpc:~$ ping -c 4 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=56 time=39.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=38.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=56 time=38.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=56 time=38.9 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 38.388/38.791/39.327/0.403 ms
testuser@testpc:~$
```

***Kuva 7.*** *Esimerkki Ping-työkalun käytöstä.*

#### 4.5.4 Traceroute

Traceroute (kuva 8) on työkalu, millä voidaan jäljittää tietoliikenteen kulkema reitti haluttuun IP-osoitteeseen. Se on reitityksen kannalta hyvin olennainen työkalu, koska sillä on mahdollista havaita muutokset reiteissä esimerkiksi yhteyden katketessa joidenkin solmujen välillä. (Linux manual pages 2016.)

```
testuser@testpc:~$ traceroute -n 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  10.0.2.2  0.322 ms  0.200 ms  0.133 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  *
```

***Kuva 8.*** *Esimerkki Traceroute-työkalun käytöstä.*

# 5 REITITYKSEN AUTOMAATINEN TESTAAMINEN

## 5.1 Testitapausten suunnittelu

Reititys on laaja konsepti, joten sen testaamisessa on paljon huomioon otettavia asioita. Vaikka reititysprotokollia on useita ja niiden ominaisuudet vaihtelevat toisistaan, on niillä kuitenkin kaikilla yhteinen tavoite: reitittää tietoliikenne lähdeosoitteesta oikeaan kohdeosoitteeseen. Testitapauksia suunniteltaessa kaikille protokollille yhteisiä ominaisuuksia kannattaa pyrkiä testaamaan yleiseksi kehitetyillä testitapauksilla, jolloin testejä tarvitsee kirjoittaa vähemmän ja niitä voidaan käyttää laajemmin.

Reitityksen kannalta on tärkeää testata reittien muutokset esimerkiksi yhteyden katketessa. Testattavia asioita voi esimerkiksi olla; kuinka paljon aikaa kuluu uudelleen reititykseen, mitä kautta liikenne reitittyy uudelleen ja palaako tietoliikenne kulkemaan vanhaa reittiä yhteyden palatessa. Tässä voidaan hyödyntää ping- ja traceroute-työkaluja. Myös yhteyksien suoritustehoa olisi hyvä testata, jolloin voidaan selvittää, miten paljon yhteyden tyyppi tai reitin pituus vaikuttaa suoritustehoon. Suoritustehotesti voidaan toteuttaa esimerkiksi iperf-ohjelmalla, jolla on mahdollista tuottaa suuri määrä verkkoliikennettä ja seurata, kuinka paljon verkon kaistaa liikenne käyttää. Iperf-ohjelmalla voidaan testata myös TCP- ja UDP-liikenteen täsmälähetystä.

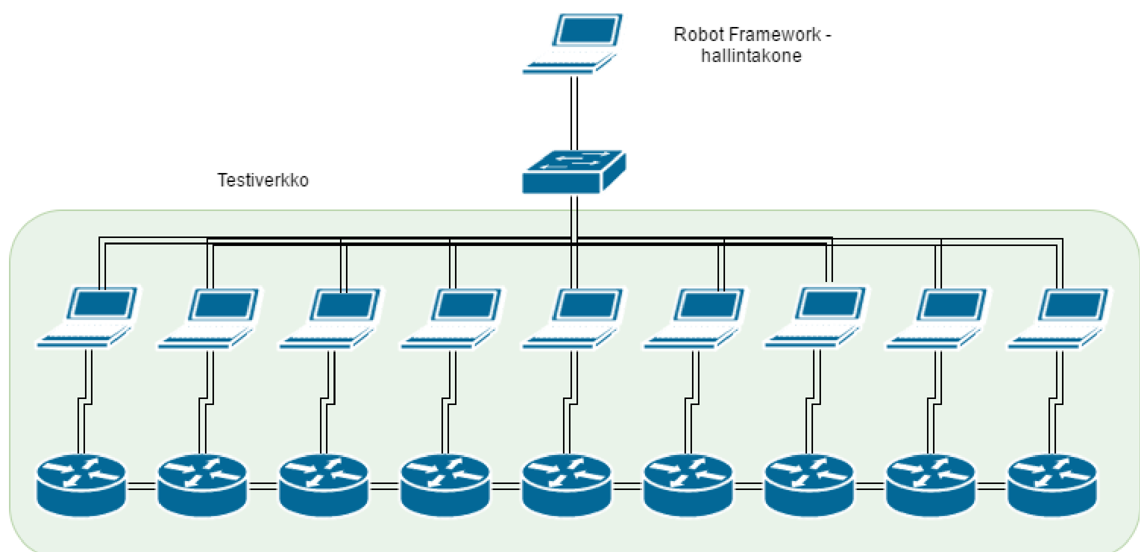
Ryhmälähetys on täsmälähetystä monimutkaisempi toteutus, joten siinä on myös suurempi mahdollisuus virheille. Ryhmälähetyksestä on kannattavaa testata, että viestit reitittyvät vain niille solmuille, jotka ovat liikennettä tilanneet, ja että ryhmälähetysliikenne pystyy hyppäämään solmujen yli. Myös tämä testi voidaan hoitaa hyödyntämällä iperf-ohjelmaa liittymällä yhdellä laitteella ryhmälähetysryhmään ja luomalla toisella laitteella ryhmälähetysliikennettä edellä mainittuun ryhmään.

Jos testattava reititysprotokolla mahdollistaa liikenteen todentamisen, myös sen toimivuutta tulisi testata. Reitityksen mainostusta voidaan testata esimerkiksi tarkkailemalla reititystaulua ja reititykseen osallistuvia rajapintoja. Rajapintoja voidaan tarkkailla jollakin paketin analysointityökalulla ja havainnoida, mitä liikennettä, kuten reitityspäivityksiä, rajapinnan kautta kulkee.

## 5.2 Testausympäristö

Kuten kappaleessa 3.1 aiemmin todettiin, on reititys pohjimmiltaan laitteiden välistä viestien vaihtamista ja tietoliikenteen ohjaamista oikeisiin verkkoihin. Tästä syystä reitityksen testausympäristössä on voitava muodostaa useita verkkoja, jotta reitityksen ominaisuuksia voidaan testata luotettavasti. Esimerkiksi monet yritysverkot voivat sisältää useita kymmeniä reitittimiä, mutta aivan näin isoa verkkoa harvemmin testaamiseen voidaan kuitenkaan toteuttaa. Reitittimiä olisi hyvä olla ainakin kymmenen, jolloin voidaan luotettavasti testata suurin osa reitityksen ominaisuuksista ja myös kriittisimmät viat todennäköisimmin ilmenevät jo tämän kokoluokan verkossa.

Testausympäristön (kuva 9) ytimenä toimii Robot Framework, jonka hallitseminen kannattaa hoitaa yhdeltä tietokoneelta, jottei testaamisen ja automaation ylläpidettävyys muutu liian monimutkaiseksi. Hallintatietokoneelta muodostetaan yhteys testikoneille, jotka voivat olla esimerkiksi kannettavia tietokoneita, Raspberry Pi:ta tai virtuaalikoneita. Testikoneet ovat yhteydessä testattaviin reitittimiin, jotka ovat yhteydessä toisiinsa muodostaen testiverkon. Testikoneille asennetaan reitityksen testaamiseen liittyvät työkalut, joita esitellään kappaleessa 4.5. Robot Framework käyttää näitä työkaluja testien suorittamiseen.




***Kuva 9.*** *Esimerkki testiverkon topologiasta.*

### 5.3 Robot Framework testaamisen automatisoinnissa

Koska Robot Framework on tehty yleispäteväksi, on sen avulla mahdollista automatisoida testaamista hyvin laaja-alaisesti. Parhaimmassa tapauksessa Robot Frameworkilla on mahdollista automatisoida koko testaamistilanne laitteiden konfiguroinnista testien ajamiseen ja lopulta lokitiedostojen keräämiseen sekä lopetustoimenpiteiden tekemiseen. Esimerkiksi Pythonin SSH-etähallintakirjastoa hyödyntäen voidaan Robot Frameworkilla automatisoida reitityksen konfiguraatio, suorittaa testejä reitittimen sisällä ja tallentaa reitittimen kertomaa dataa. Robot Frameworkille on kehitetty Rammbock-niminen kirjasto <https://github.com/robotframework/Rammbock/> verkkoprotokollien testaamista varten, jolla voidaan testata paketteja ja viestimalleja, mutta sitä ei tässä opinnäytetyössä käsitellä (Bisht 2013, 67).

Robot Frameworkilla toteutettu automatisoitu testitilanne voisi olla esimerkiksi seuraavanlainen: Robot Framework ottaa yhteyden reitittimeen, konfiguroi reitittimen rajapinnat sekä OSPF-prosessin ja testaa, onko yhteys naapurisolmuun toimiva ja toimiiko OSPF-prosessi niin kuin pitää. Kuvasta 10 nähdään Pythonilla ohjelmoitu mukautettu testikirjasto, jossa reitittimen rajapinnoille asetetaan IP-osoitteet ja määritellään rajapinnat aktiivisiksi. Kuvasta nähdään myös, kuinka OSPF-prosessi luodaan reitittimelle testikirjaston avulla ja OSPF asetetaan mainostamaan reitittimeen yhdistyviä verkkoja.

```
#Configure given interfaces with ip address and subnetmask.
def configure_interface(self, interfaces):
    for int_id, val in interfaces.iteritems():
        self.send_command(cmd='int ' + int_id)
        for net, mask in val.iteritems():
            self.send_command(cmd='ip address ' + str(net) + ' ' + str(mask))
            logger.console("\tConfigured interface " + int_id + " with ip address " +
                           str(net) + " " + str(mask))
        self.send_command(cmd='no shut')

#Configure OSPF process with given process id, router-id and advertised networks and areas.
def configure_ospf(self, process_id, router_id, network_list):
    self.send_command(cmd='router ospf ' + process_id)
    self.send_command(cmd='router-id ' + router_id)
    logger.console("\tConfigured OSPF process " + process_id  + ' with router-id ' + router_id)
    logger.console("Advertised networks: ")
    for area, val in network_list.iteritems():
        for net, mask in val.iteritems():
            self.send_command(cmd='network ' + str(net) + " " + str(mask) + ' ' + area)
            logger.console('\t' + str(net) + " in " + area)
```

***Kuva 10.*** *Esimerkki mukautetusta testikirjastosta konfiguroinnin automatisointiin.*

Konfiguroinnin lisäksi mukautettuun kirjastoon ohjelmoidaan myös testien toiminnallisuus. Kuvasta 11 havaitaan, kuinka yhteyttä naapurisolmuun testataan ping-työkalua hyväksi käyttäen ja reitittimen lähettämää dataa ping-työkalun tuloksista jäsennetään ja tulostetaan näytölle. Jos naapurisolmulta ei saada vastausta, määritetään testi Robot Frameworkin sisäänrakennetun *fail()*-funktion avulla epäonnistuneeksi. OSPF-prosessia testataan antamalla reitittimelle komento tulostaa reititystaulu ja lukemalla reititystaulusta, löytyykö sieltä OSPF-protokollan avulla opittuja reittejä muilta reitittimiltä. Jos reittejä ei löydy, määritellään testi epäonnistuneeksi.

```
#Ping to neighbor and check for reply.
def ping_to_neighbor(self, ping):
    self.send_command(cmd='ping ' + ping)
    time.sleep(3)
    data = self.read_serial()
    regex = re.compile(r'Success rate is \d+ percent \([1-5]\/5\)')
    match = regex.search(data)
    if match:
        logger.console('\nPing test successful. Neighbor is reachable')
    else:
        logger.console('\nPing test unsuccessful. Neighbor not reachable')
        BuiltIn().fail("Ping test failed.")

#Check routing table for OSPF entries.
def check_routing_table(self):
    self.send_command(cmd='show ip route')
    data = self.read_serial()
    regex = re.compile(r'O IA')
    match = regex.search(data)
    if match:
        logger.console('OSPF routes found from routing table.')
    else:
        logger.console('OSPF routes cannot be found from routing ' +
                       'table.')
        BuiltIn().fail("Routing table test failed")
```

***Kuva 11.*** *Mukautettu testikirjasto naapuriyhteyden ja OSPF-protokollan testaamiseen.*

Varsinainen automaattinen testaaminen toteutetaan Robot Frameworkilla. Mukautetuista kirjastoista muodostetaan Robot Frameworkin testikokoelmaan avainsanoja, joista muodostetaan testitapauksia. Kuvassa 12 on Robot Frameworkin testikokoelma, jonka avainsanat ja testitapaukset ovat muodostettu aiemmissa kuvissa 10 ja 11 esiteltyjen mukautettujen testikirjastojen funktioita hyväksi käyttäen. Robot Framework suorittaa testitapaukset, ja jos testi ei ensimmäisellä yrityksellä onnistu, odottaa Robot Framework ennalta määritellyn ajan ja yrittää uudelleen. Jos testi ei odottamisesta huolimatta mene läpi, merkitään testausraporttiin *FAIL*. Läpäisty testi merkitään merkinnällä *PASS*. Liitteissä 1 ja 2 esitellään testikirjasto ja testikokoelma kokonaisuudessaan.

```
#Configure as many interfaces as are defined in variable file.
#Sets ip addresses and subnetmasks and executes 'no shutdown' command.
Configure Interfaces
    Log To Console   \nConfiguring Interfaces...
    Run Keyword And Continue On Failure   Configure Interface
    ...   ${interfaces}

#Configure OSPF process. Areas and advertised networks are defined
#in variable file.
Configure Ospf Process
    Log To Console   \nConfiguring OSPF process...
    Run Keyword And Continue On Failure   Configure Ospf
    ...   ${process_id}   ${router_id}   ${network_list}

#Test connectivity with neighbors and check if OSPF process
#is working as intented.
Test Connectivity
    Return To Enable Mode
    Wait Until Keyword Succeeds   10s   5s   Ping To Neighbor   ${ospf_neighbor}
    Wait Until Keyword Succeeds   30s   10s   Check Routing Table
    Enter Configuration Mode

*** Test Cases ***
Cisco Configuration Automation
    Configure Interfaces
    Configure Ospf Process
Cisco Connectivity Test
    Test Connectivity
```

***Kuva 12.*** *Osa Robot Frameworkin testikokoelmasta.*

Robot Frameworkin arkkitehtuuri mahdollistaa suurien ympäristöjen testaamisen, sillä, kuten kappaleessa 4.2 todettiin, Robot Framework ei yksinään toteuta testaamista vaan toimii testikirjastojen ja -työkalujen avulla. Testityökalut voivat olla hajautettuna esimerkiksi kappaleessa 5.2 mainituilla testikoneilla, joilla testaamisen toiminnallisuus tapahtuu. Tämä mahdollistaa testiverkon koon muuttamisen ilman Robot Frameworkin ja testausympäristön uudelleen konfigurointia.

Robot Frameworkin automaatiolla on mahdollista tuottaa satunnaisuutta testaamiseen, jolloin voidaan esimerkiksi satunnaistaa testitapausten suoritusjärjestys. Pythonilla voidaan myös rakentaa testikirjasto, joka esimerkiksi katkaisee jonkin satunnaisen yhteyden reitittimien väliltä. Tällöin testitapaus muistuttaa enemmän tosielämän tilannetta ja uusia vikoja voi ilmetä, koska testiä ei suoriteta aina samalla kaavalla.

Toisinaan testaaminen saattaa vaatia useiden testien yhtäaikaista suorittamista varsinkin reitityksen monimutkaisesta luonteesta johtuen. Pabot on Robot Frameworkin laajennus, jolla testikokoelmien rinnakkainen ajaminen on mahdollista. Yhdellä testikokoel-

malla voitaisiin esimerkiksi muodostaa OSPF-verkko, toisella EIGRP-verkko ja kolmannella mitata, kauanko verkoilla menee aikaa yhdessä saavuttaa tasapainoinen tila.

## 5.4 Kerättävä data

Testauksesta olisi hyvä kerätä talteen tietoa tuloksista, jotta niitä voidaan verrata keskenään edellisten sekä tulevien testitulosten kanssa. Robot Frameworkin omien testauslokien lisäksi reitityksestä olisi hyvä tallentaa tietoa ainakin mitattavista numeerisista arvoista, kuten verkon muodostumiseen ja liikenteen kulkemiseen liittyvistä aikamääreistä. Esimerkiksi aika, mikä reitittimiltä kuluu liikenteen uudelleen reitittämiseen topologian muuttuessa, kannattaa tallentaa.

Laitteiden tallentamat yleiseen standardiin perustuvat syslog-lokitiedostot ovat hyödyllisiä esimerkiksi, kun selvitetään vikatilannetta, jota ei voi numeerisesta datasta havaita. Jos reititysprotokolla on jostain syystä lakannut toimimasta kokonaan, voidaan syslog-tiedostoista katsoa, milloin tämä on tapahtunut ja mitkä mahdolliset muut prosessit ovat voineet vaikuttaa virheen tapahtumiseen. Koska kaikelle kerättävälle datalle ei ole olemassa valmista lokitiedostojen keräysprosessia, voidaan Robot Frameworkilla tehdä muokattuja kirjastoja, joilla esimerkiksi jäsennetään laitteiden komentoriville tulostamasta datasta hyödyllistä tietoa omaksi lokitiedostoksi.

## 5.5 Tulosten analysointi

Testaamisen yhtenä tärkeänä vaiheena on tulosten analysointi. Robot Frameworkin tallentamista testauslokeista voidaan havainnoida, onko testi onnistunut vai ei. Jos testi on epäonnistunut, voidaan lokeista saada myös tietoa siitä, missä kohtaa testausta virhe on tapahtunut. Vaikka Robot Frameworkin testilokit näyttäisivätkin testin onnistuneen, voi silti olla, ettei testin lopputulos ole sitä, mitä odotettiin. Tällöin siirrytään analysoimaan muuta testin aikana kerättyä tietoa. Numeerisesti mitattavista testaustuloksista olisi hyvä piirtää kuvaaja, jotta tuloksia voidaan helpommin verrata edellisten testien vastaaviin tuloksiin ja havaita, onko ominaisuudessa tapahtunut muutosta suuntaan tai toiseen.

Testaamisesta voidaan havaita myös paljon tuloksia, joita ei voi suoraan mitata. Esimerkiksi jokin edellisellä kerralla toiminut ominaisuus on voinut lakata toimimasta, minkä vuoksi voidaan joutua turvautumaan esimerkiksi reitittimen tallentamaan syslog-tietoon. Syslog-tiedostoon kerätään tietoa laitteen toiminnasta sekä virheilmoituksista, joita laite lähettää virhetilanteen sattuessa. Laitteella voi olla käytössä myös muita toimintaan liittyviä lokeja, mitkä helpottavat virhetilanteen havainnointia.

## 5.6 Tulosten merkitys kehitystyössä

Koska testaaminen on kehitystyön tukitoimi, ei tulosten keräämisellä tai koko testaamisella, luonnollisesti, olisi mitään merkitystä, ellei sen tuottamaa tietoa voitaisi hyödyntää varsinaisessa kehitystyössä. Tuloksia analysoimalla kehittäjillä on mahdollisuus jäljittää virheet ohjelmiston toiminnassa hyvinkin tarkasti ja nopeasti. Tuloksilla varmistetaan myös, että ohjelmiston aiempi toiminnallisuus ei ole taantunut, joten voidaan alkaa suunnittelemaan myös uusia ominaisuuksia. Tuloksista täytyy aina keskustella kehittäjien ja mahdollisesti myös johtoryhmän kanssa, jotta voidaan päättää jatkotoimenpiteistä.

## 6 POHDINTA

Tämän opinnäytetyön tavoitteena oli tutkia reitityksen automaattista testaamista Robot Frameworkin avulla. Toimeksiantajana toiminut Bittium Wireless Oy mahdollisti tutkimustyön tekemisen tiloissaan antamalla käyttööni tarvittavat resurssit. Tarkoituksena oli tutkimuksen avulla tuottaa tietoa reitityksen automaattisesta testaamisesta Robot Frameworkin näkökulmasta. Reititys ja automaattinen testaaminen ovat molemmat todella laajoja aihealueita ja pelkästään Robot Frameworkistakin olisi ollut mahdollista kirjoittaa opinnäytetyö yksistään, joten on selvää, että opinnäytetyö on suhteellisen tiivis ja aihealueita vain pintapuolisesti käsittelevä kokonaisuus.

Mielestäni opinnäytetyössä kuitenkin onnistuneesti yhdistettiin edellä mainitut aihealueet yhdeksi selkeäksi kokonaisuudeksi, johon perehtymällä on mahdollista saada kohtuullisen kattava kuva reitityksestä, testaamisesta, automaatiosta ja kaikista näistä kokonaisuutena. Tiedon yleisestä luonteesta johtuen sitä on mahdollista hyödyntää kaikenlaisen reitityksen testaamisessa.

Reitityksen automaattisen testaamisen työvälineeksi Robot Framework soveltuu hyvin sen yleiskäyttöisyyden ja ohjelmistoriippumattomuuden ansioista. Koska Robot Framework on Pythonin avulla hyvin laajennettavissa, voidaan sillä käytännössä testata mitä vain, mihin Pythonilla on mahdollista ohjelmoida mukautettu kirjasto. Vaikka Robot Framework on alun perin hyväksymistestaamiseen kehitetty, soveltuu se myös regressio- ja integraatiotestaamiseen.

Reitityksen testaamista voidaan laajentaa yhdistämällä siihen esimerkiksi tietoturvaan liittyviä ominaisuuksia kuten liikenteen salauksen testaaminen, palomuurin testaaminen tai reitityksen testaaminen tunneleiden läpi. Opinnäytetyön rajatusta aiheesta johtuen näitä ei suuresta mielenkiinnosta huolimatta pystytty tässä työssä käsittelemään.

Opinnäytetyötä tehdessä opin ennen kaikkea suunnittelun tärkeydestä automaation ja testaamisen toteuttamisessa. Kun testitapaukset suunnitellaan alusta asti huolella, on niitä helppo laajentaa ja ylläpitää. Sama pätee automaatioon. Opin myös, että ilman automaatiota reitityksen testaaminen voi olla todella hankalaa ja aikaa vievää.

**LÄHTEET**

Anand, V. & Chakrabarty, K. 2004. Cisco IP Routing Protocols: Troubleshooting Techniques. Charles River Media / Cengage Learning.

Bisht, S. 2013. Robot framework test automation. Packt Publishing.

Black, R. & Mitchell, J. 2015. Advanced Software Testing - Vol. 3, 2nd Edition, 2nd Edition. Rocky Nook.

Blank, A. 2006. TCP/IP JumpStart : Internet Protocol Basics. Sybex.

Gregory, J. & Crispin, L. 2014. More Agile Testing: Learning Journeys for the Whole Team. Addison-Wesley Professional.

Hutcheson, M. 2003. Software Testing Fundamentals: Methods and Metrics (1). Wiley.

Jacquet, P., Mühlethaler, P., Clausen, T., Laouiti, A., Qayyum, A. & Viennot, L. Optimized Link State Routing Protocol for Ad Hoc Networks. Luettu 26.10.2016. https://www.cs.jhu.edu/~dholmer/600.647/papers/OLSR.pdf

Johnson, D., Ntlatlapa, N. & Aichele, C. A simple pragmatic approach to mesh routing using BATMAN. Luettu 26.10.2016. http://cs.ucsb.edu/~davidj/Files/Batman_ifip.pdf

Kabelová, A. & Dostálek, L. 2006. Understanding TCP/IP : A Clear and Comprehensive Guide (1). Packt Publishing.

Linux manual pages. Luettu 20.10.2016. https://linux.die.net/man/

Loveland, S., Shannon, M. & Miller, G. 2004. Software Testing Techniques: Finding the Defects That Matter. Charles River Media / Cengage Learning.

Macfarlane, J. 2006. Network Routing Basics: Understanding IP Routing in Cisco Systems. Indianapolis, IN: Wiley.

Medhi, D., Ramasamy, K. & Zupan, J. 2010. The Morgan Kaufmann Series in Networking: Network Routing: Algorithms, Protocols, and Architectures. Morgan Kaufmann.

Menon, V. 2013. TestNG Beginner's Guide (1). Packt Publishing.

Mette Jonassen Hass, A. 2008. Guide to Advanced Software Testing. Artech House Books.

Myers, G., Sandler, C. & Badgett, T. 2011. The Art of Software Testing (3). Wiley.

Robot Framework User Guide version 3.0. Luettu 19.9.2016. http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

Spillner, A., Linz, T. & Schaefer, H. 2014. Software Testing Foundations, 4th Edition. Rocky Nook.

# LIITTEET

Liite 1. Esimerkki Robot Frameworkin mukautetusta testikirjastosta.

```
import serial
import time
import sys
import Queue
import re

from robot.api import logger
from robot.libraries.BuiltIn import BuiltIn

class CiscoConfigurationManager:

    ROBOT_LIBRARY_SCOPE = 'GLOBAL'

    def __init__(self):
        self.console = ''
        self.queue = ''
        self.buffer = ''

    #Read data from serial connection.
    def read_serial(self):
        while True:
            self.queue = Queue.Queue(0)
            self.buffer += self.console.read(self.console.inWaiting())
            self.queue.put(self.buffer)
            data = self.queue.get(False)
            return data

    #Check if user is already logged in to router
    def check_if_logged_in(self):
        self.console.write("\r\n\r\n")
        time.sleep(0.5)
        prompt = self.read_serial()
        if '>' in prompt or '#' in prompt:
            return True
        else:
            return False



    #Check login status and login to router.
    def login_to_router(self, username, password):
        login_status = self.check_if_logged_in()
        if login_status:
            logger.console("Already logged in")
            return None

        #Checks if router is asking username and/or
        #password.
        while True:
            self.console.write("\r\n")
            time.sleep(0.5)
            input_data = self.read_serial()
            if not 'Username' in input_data:
                continue
            self.console.write(username + "\r\n")
            time.sleep(0.5)
            print input_data
            input_data = self.read_serial()
            if not 'Password' in input_data:
                continue
            self.console.write(password + "\r\n")
            time.sleep(0.5)
            login_status = self.check_if_logged_in()
            if login_status:
                logger.console("Logged in")
                break

    #Logout from router.
    def logout_from_router(self):
        if self.check_if_logged_in():
            self.console.write("exit\r\n")

    #Send data to router.
    def send_command(self, cmd=''):
        self.console.write(cmd + '\r\n')
        time.sleep(1)
        return self.read_serial()
```

```
#Disable paging on router. Purpose is to avoid
#errors while reading data.
def disable_paging(self):
    self.send_command(cmd='terminal length 0')

#Disable ip domain look-up. Purpose is to avoid
#errors executing commands in router.
def disable_ip_domain_lookup(self):
    self.enter_configuration_mode()
    self.send_command(cmd='no ip domain-lookup')
    self.return_to_enable_mode()

#Enter enable mode.
def enter_enable_mode(self):
    self.send_command(cmd='en')

#Enter terminal configuration mode.
def enter_configuration_mode(self):
    self.send_command(cmd='configure t')

#Return to enable mode regardless of the current mode.
def return_to_enable_mode(self):
    self.send_command(cmd='end')

#Configure given interfaces with ip address and subnetmask.
def configure_interface(self, interfaces):
    for int_id, val in interfaces.iteritems():
        self.send_command(cmd='int ' + int_id)
        for net, mask in val.iteritems():
            self.send_command(cmd='ip address ' + str(net) + ' ' + str(mask))
            logger.console("\tConfigured interface " + int_id + " with ip address " +
                           str(net) + " " + str(mask))
        self.send_command(cmd='no shut')

#Configure OSPF process with given process id, router-id and advertised networks and areas.
def configure_ospf(self, process_id, router_id, network_list):
    self.send_command(cmd='router ospf ' + process_id)
    self.send_command(cmd='router-id ' + router_id)
    logger.console("\tConfigured OSPF process " + process_id  + ' with router-id ' + router_id)
    logger.console("Advertised networks: ")
    for area, val in network_list.iteritems():
        for net, mask in val.iteritems():
            self.send_command(cmd='network ' + str(net) + " " + str(mask) + ' ' + area)
            logger.console('\t' + str(net) + " in " + area)
```

```
#Initialize and open serial connection.
def setup_serial_connection(self):
    self.console = serial.Serial(
        port='/dev/ttyUSB0',
        baudrate=9600,
        parity='N',
        stopbits=1,
        bytesize=8)

    if not self.console.isOpen():
        sys.exit()

    return self.console

#Gracefully close serial connection.
def close_serial_connection(self):
    self.console.close()

#Ping to neighbor and check for reply.
def ping_to_neighbor(self, ping):
    self.send_command(cmd='ping ' + ping)
    time.sleep(3)
    data = self.read_serial()
    regex = re.compile(r'Success rate is \d+ percent \([1-5]\/5\)')
    match = regex.search(data)
    if match:
        logger.console('\nPing test successful. Neighbor is reachable')
    else:
        logger.console('\nPing test unsuccessful. Neighbor not reachable')
        BuiltIn().fail("Ping test failed.")

#Check routing table for OSPF entries.
def check_routing_table(self):
    self.send_command(cmd='show ip route')
    data = self.read_serial()
    regex = re.compile(r'O IA')
    match = regex.search(data)
    if match:
        logger.console('OSPF routes found from routing table.')
    else:
        logger.console('OSPF routes cannot be found from routing ' +
                       'table.')
        BuiltIn().fail("Routing table test failed")
```

Liite 2. Esimerkki Robot Frameworkin testikokoelmasta.

```
*** Settings ***
Variables    cisco_variables.py

Suite Setup  Suite Setup
Suite Teardown  Suite Teardown

Test Setup  Test Setup
Test Teardown  Test Teardown

*** Variables ***
${process_id}  1
${router_id}  1.2.3.4

*** Keywords ***
Suite Setup
    Import Library  CiscoConfigurationManager

    #Initialize and open the serial connection to Cisco router
    #console port and execute login keyword.
    Log To Console  Initializing serial connection...
    Wait Until Keyword Succeeds  30s  5s  Setup Serial Connection
    Wait Until Keyword Succeeds  5s  2s  Login

Suite Teardown
    #Execute logout keyword and gracefully close the serial connection
    Wait Until Keyword Succeeds  5s  2s  Logout
    Log To Console  Closing Serial Connection...
    Wait Until Keyword Succeeds  30s  5s  Close Serial Connection
    Log To Console  Connection Closed

Test Setup
    Enter Configuration Mode

Test Teardown
    Return To Enable Mode

#Login to Cisco router and disable paging and ip domain-lookup.
Login
    Log To Console  Logging into router...
    Wait Until Keyword Succeeds  30s  5s  Login To Router
    ...  ${username}  ${password}
    Enter Enable Mode
    Disable Paging
    Disable Ip Domain Lookup

#Configure as many interfaces as are defined in variable file.
#Sets ip addresses and subnetmasks and executes 'no shutdown' command.
Configure Interfaces
    Log To Console  \nConfiguring Interfaces...
    Run Keyword And Continue On Failure  Configure Interface
    ...  ${interfaces}

#Configure OSPF process. Areas and advertised networks are defined
#in variable file.
Configure Ospf Process
    Log To Console  \nConfiguring OSPF process...
    Run Keyword And Continue On Failure  Configure Ospf
    ...  ${process_id}  ${router_id}  ${network_list}

#Logout from the router.
Logout
    Log To Console  Logging out from router...
    Wait Until Keyword Succeeds  30s  5s  Logout From Router

#Test connectivity with neighbors and check if OSPF process
#is working as intented.
Test Connectivity
    Return To Enable Mode
    Wait Until Keyword Succeeds  10s  5s  Ping To Neighbor  ${ospf_neighbor}
    Wait Until Keyword Succeeds  30s  10s  Check Routing Table
    Enter Configuration Mode

*** Test Cases ***
Cisco Configuration Automation
    Configure Interfaces
    Configure Ospf Process
Cisco Connectivity Test
    Test Connectivity
```