Tampereen yliopisto

Valtteri Valkonen

VERKONHALLINTAPROTOKOLLIEN HYÖDYNTÄMINEN VERKKOJEN KYVYKKYYDEN TESTAUKSESSA

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
Lokakuu 2020

TIIVISTELMÄ

Valtteri Valkonen: Verkonhallintaprotokollien hyödyntäminen verkkojen kyvykkyyden testauksessa
Kandidaatintyö
Tampereen yliopisto
Tieto- ja sähkötekniikan tutkinto-ohjelma
Lokakuu 2020

Verkkotestaus on tärkeää moderneissa tuotantoverkoissa niiden laajuuden ja monimutkaisuuden vuoksi. Huomaamatta jääneet virhetilanteet voivat vaikuttaa merkittävästi verkkojen suorituskykyyn, mikä heijastuu negatiivisesti palveluntarjoajalle. Tämän kandidaatintyön tavoitteena on esitellä verkonhallintaprotokollia, tuotantoverkkojen testauksen menetelmiä ja teknologioita, joita voidaan hyödyntää tietoverkkojen testauksessa. Lisäksi työssä esitellään verkkotestauslaitteen toteutus, joka hyödyntää käsiteltyjä protokollia ja teknologioita. Lähdeaineistoina on käytetty määrittelydokumentteja sekä tietoverkoista ja testauksesta kirjoitettuja artikkeleita ja konferenssijulkaisuja.

Verkonhallintaprotokollista esitellään muutamia yleisesti käytössä olevia protokollia eri alustoilta. Esitellyt verkonhallintaprotokollat ovat Internet Control Message Protocol, Simple Network Management Protocol, Windows Management Instrumentation ja Network Configuration Protocol. Esittelyissä kuvataan protokollien kehysrakenteet ja toiminta. Näistä protokollista Network Configuration Protocol todettiin sopivimmaksi toteutetulle testauslaitteelle monipuolisen toiminnallisuutensa vuoksi.

Tuotantoverkkojen testausta monimutkaistavat laajat protokollatukivaatimukset, palvelunlaatuvaatimukset ja sisäiset riippuvuudet. Testauksen onnistuminen vaatii tuekseen näkökulmia ja menetelmiä, joista esitellään muutamia yleisesti käytössä olevia. Testaustavoista alhaalta ylöspäin -lähestymistapa todetaan paremmaksi tietoverkoille kuin ylhäältä alaspäin -lähestymistapa. Yhdistelmä laitevirtualisointia, emulointia ja liikennesimulaatiota todetaan tehokkaaksi tavaksi laskea testauskustannuksia ja varmistaa, että testausverkko vastaa toiminnaltaan tuotantoverkkoa.

Työssä esitellään toteutus mobiilista verkkotestauslaitteesta, joka kykenee tarkastamaan siihen kytketyn verkon tuen eri protokollille ja testaamaan sen odotettuja parametreja vasten. Testaaja pystyy laitteen avulla varmistamaan Open Shortest Path First -reititysprotokollan toiminnan ja tarkastamaan IPsec-valmiuden laitteissa, joilta sitä odotetaan. Laboratoriossa tehdyn testiajon perusteella laite helpottaa verkkotestausta, sillä se kykenee testaamaan useita protokollia yhdellä laitteella ja ohjelmistolla. Verkkotestauslaitteen toiminnallisuutta voidaan laajentaa tulevaisuudessa kattamaan useampia protokollia ja järjestelmäratkaisuja.

Avainsanat: Verkonhallintaprotokollat, testaus, testauslaitteisto, testausmenetelmät, testausnäkökulmat, virtualisointi

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

HTTP	Hypertext Transfer Protcol. Protokolla, jota käytetään verkkosivujen siirtoon.
OSPF	Open Shortest Path First. Reititysprotokolla, joka hakee reittejä verkosta dynaamisesti.
IETF	Internet Engineering Task Force on vuonna 1986 perustettu protokollien standardoinnista vastaava organisaatio.
vSRX	Juniper Networks:in kehittämä virtuaalinen palomuurireititin.
LAN	Local Area Network. Pienellä maantieteellisellä alueella toimiva verkko.
WAN	Wide Area Network. Siirtoverkko joka kattaa suuren maantieteelliseen alueen.
VRF	Virtual Routing and Forwarding.

1. JOHDANTO

Tietoverkkojen määrän ja monimutkaisuuden kasvu on lisännyt tarvetta asetushallinnan parannuksille; etenkin suuret verkot alkavat olla niin monimutkaisia, että niiden perinteinen hallinta käsin konsolin tai konsoliyhteyden kautta alkaa olla hankalaa ja virhealtista työtä. Sherryn et al. vuonna 2012 suorittamassa kyselytutkimuksessa erikokoisten yritysten verkkojenhallintahenkilöstö arvioi virheellisten konfiguraatioiden aiheuttavan suurimman osan verkkojen virhetiloista [1]. Ylläpitotyötä helpottamaan on kehitetty verkkoautomaatiota ja verkonhallintaprotokollia, joiden tarkoituksena on vähentää kustannuksia ja tehdä konfigurointiprosessista yksinkertaisempaa.

Samalla, kun verkot monimutkaistuvat, monimutkaistuu myös niiden toiminnallisuuden testaaminen ja todentaminen. Testauksen, virheiden havaitsemisen ja korjaamisen tueksi voisikin olla hyödyllistä kehittää automaattisia työkaluja, joilla verkkojen toiminnallisuutta voisi tehokkaammin tarkastella. Tässä työssä osoitetaan, että samat protokollat, joita käytetään verkon konfiguraatiohallintaan ja tarkkailuun, soveltuvat myös testausvälineiksi, joilla voi yhdellä toteutuksella asettaa verkkokonfiguraatiot ja sitten toisella toteutuksella todentaa niiden toimivuuden. Työssä myös määritellään ja esitellään esimerkkitoteutus tällaisesta verkkotestauslaitteesta, jota tässä työssä kutsutaan verkkotesteriksi.

Verkkotesteri on toteutettu tiimiprojektina, jossa on useita tekijöitä. Tämä työ on osa laajempaa projektia, mutta työn sisältö keskittyy omiin vastuualueisiini, jotka ovat verkot, verkkoprotokollat ja Juniperin vSRX virtuaalinen palomuurireititin. Työssä käsitellään myös verkkotesterin järjestelmäarkkitehtuurisia valintoja, koska ne liittyvät oleellisesti laitteen sisäisten verkkojen suunnitteluun ja toteutukseen.

Toisessa luvussa perehdytään verkonhallintaprotokolliin käsitteenä ja esitellään muutamia yleisiä, käytössä olevia verkonhallintaprotokollia. Tämän jälkeen, luvussa kolme, puhutaan verkkojen testaamisesta ja verkonhallintaprotokollien hyödyntämisestä testaamisen automatisoinnissa. Luvussa 4 esitellään verkkotesterin esimerkkitoteutuksen arkkitehtuuria ja toimintaa. Viidennessä luvussa suoritetaan testiajo verkkotesterillä laboratorioverkossa ja analysoidaan saatuja esimerkkituloksia. Lopuksi vedetään yhteen tutkielmassa käsitellyt protokollat ja menetelmät, sekä verkkotesterin toiminta ja tulevaisuuden kehitysmahdollisuudet.

2. VERKONHALLINTAPROTOKOLLAT

Verkonhallintaprotokollat ovat verkkoprotokollaperhe, joka mahdollistaa verkon tilan tarkkailun ja asetusten etähallinnan. Tämä mahdollistaa helpomman, keskitetyn tavan hallita suuria verkkoja suhteessa käsinhallintaan komentorivin välityksellä. Muista sovelluskerroksen protokollista, kuten Hypertext Transfer Protocol (HTTP), poiketen verkonhallintaprotokollat eivät kuljeta käyttäjien hyötykuormaa, vaan ainoastaan dataa verkon ja sen laitteiden tilasta.

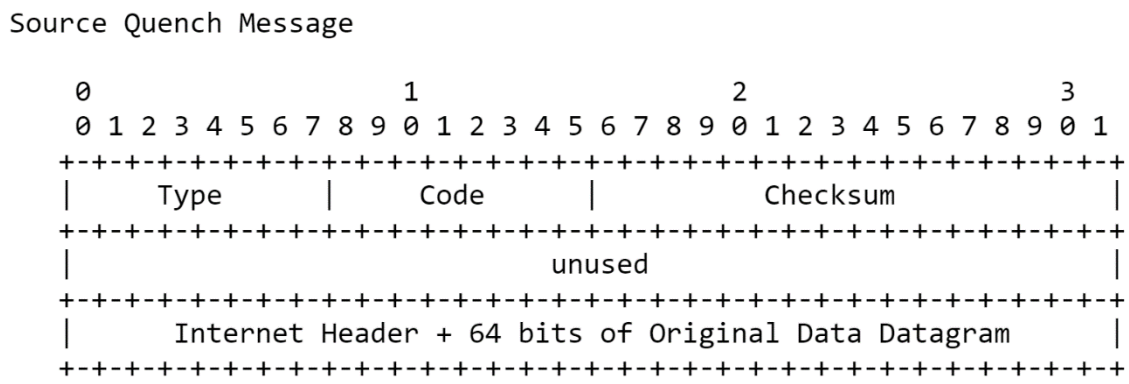
Verkonhallintaprotokollilla on kolme erilaista näkökulmaa verkon tarkkailuun: virheet, suorituskyky ja toiminta. Virhenäkökulmassa protokolla havaitsee ja raportoi verkon virheitä. Suorituskykynäkökulmassa protokolla kerää tietoa erilaisista suorituskykyparametreista, esimerkiksi läpäisykyvystä tai pudotetuista paketeista. Tämän tyyppinen verkon tarkkailu voi antaa vihjeitä esimerkiksi verkon pullonkauloista. Toimintanäkökulmassa protokolla tarjoaa työkaluja erilaisten toimintojen suorittamiseen hallittaville olioille. Oliot ovat tässä tapauksessa reitittimiä ja niiden tietoaineistoja, kuten asetuksia ja tilatietoja. Toteutuksesta riippuen protokolla voi tarjota työkaluja usean eri näkökulman toteuttamiseen. [2]

2.1 Internet Control Message Protocol

Internet Control Message Protocol (ICMP) on RFC 792 -dokumentissa vuonna 1981 määritelty kontrolliprotokolla, joka tukee TCP/IP-protokollapinoa kertomalla päästä päähän -yhteyksissä tapahtuvista IP-pakettien käsittelyvirheistä. ICMP ei ole varsinaisesti verkonhallintaprotokolla ja se toimii paljon matalammalla kerroksella kuin verkonhallintaprotokollat yleensä. ICMP:n tehtävä ei myöskään ole tarkkailla verkkoa ja sen laitteita laajemmin, vaan se tarjoaa vain virheraportointia yksittäisten yhteyksien tasolla. Sen funktio verkossa on kuitenkin samantapainen kuin verkonhallintaprotokollilla; ICMP tarjoaa tietoa yhteyksissä tapahtuvista virheistä automaattisen ja manuaalisen virheidenkäsittelyn tueksi. ICMP on tärkeä osa TCP/IP-protokollapinoa, ja jokaisen IP-moduulin on sitä tuettava [3].

ICMP kehitettiin, koska Internet Protocol (IP) ei ole täysin luotettava ja tarvittiin jokin tapa, jolla verkko tai yhteyden päässä oleva päätelaite voi ilmoittaa tapahtuneista virheistä. ICMP:llä ei ole omaa otsikkoa, vaan ICMP-viestit lähetetään käyttäen IP-otsikkoa ja protokollakentän arvoa 1. Kuvassa 1 näkyvät ICMP-viestin hyötykuormana olevat ICMP-

kentät, jotka määrittävät, minkälaisesta virheestä on kyse ja viimeisenä virheen aiheuttaneen IP-paketin IP-otsikko ja 64 ensimmäistä bittiä.



Kuva 1. *ICMP-kentät Source Quench –viestissä [3].*

Ensimmäiset 32 bittiä ovat kaikissa ICMP-viesteissä samat. Loput kentät antavat lisätietoja viestin syystä ja vaihtelevat siksi hieman ICMP-viestin mukaan. Ensimmäisenä hyötykuormassa on tyyppikenttä, joka kertoo ICMP-viestin tyypin. Tyyppikentän arvo voi olla välillä 0–255, ja näistä on standardoitu tai varattu joitain kymmeniä. Eräitä yleisimmin käytettyjä tyyppejä ovat

- 8 – Echo Request
- 0 – Echo Reply
- 3 – Destination Unreachable
- 11 – Time Exceeded.

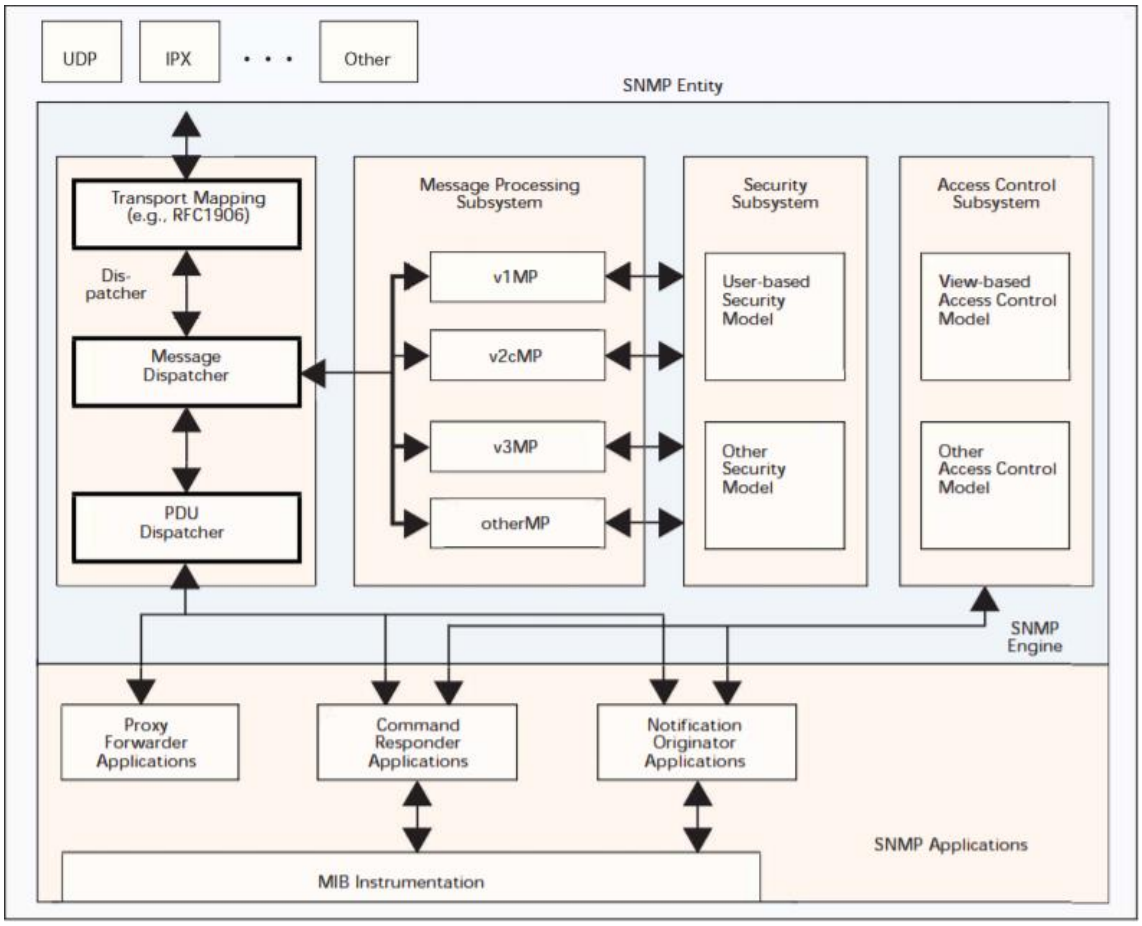
Tyypit 8 (Echo Request) ja 0 (Echo Reply) muodostavat yhdessä ping-työkalun, jolla testataan osoitteiden välistä saavutettavuutta verkossa. Destination Unreachable -tyypin viestiä käytetään kertomaan paketin reititysongelmista. Time Exceeded -tyypin viesti kertoo esimerkiksi Time To Live -kentän ehtymisestä IP-otsikossa.

Seuraava ICMP-kenttä on koodi, jolla voidaan tarkentaa ICMP-viestin aiheuttanutta virhettä. Esimerkiksi Destination Unreachable -tyypin viesti käyttää koodeja 0–5 kertomaan, mikä yhteyden osuus on saavuttamattomissa. Koodit liittyvät aina ICMP:n tyyppiin, ja niiden määrä sekä merkitys vaihtelevat tyypin mukaan. Kaikki ICMP-tyypit eivät käytä koodeja, jolloin koodikenttä on aina nolla.

2.2 Simple Network Management Protocol

Simple Network Management Protocol (SNMP) on Internet Engineering Task Force:n (IETF) vuonna 1988 standardoima verkonhallintaprotokolla. SNMP:stä on olemassa kolme versiota: SNMPv1, SNMPv2 ja SNMPv3, joiden määrittelyt löytyvät useista eri

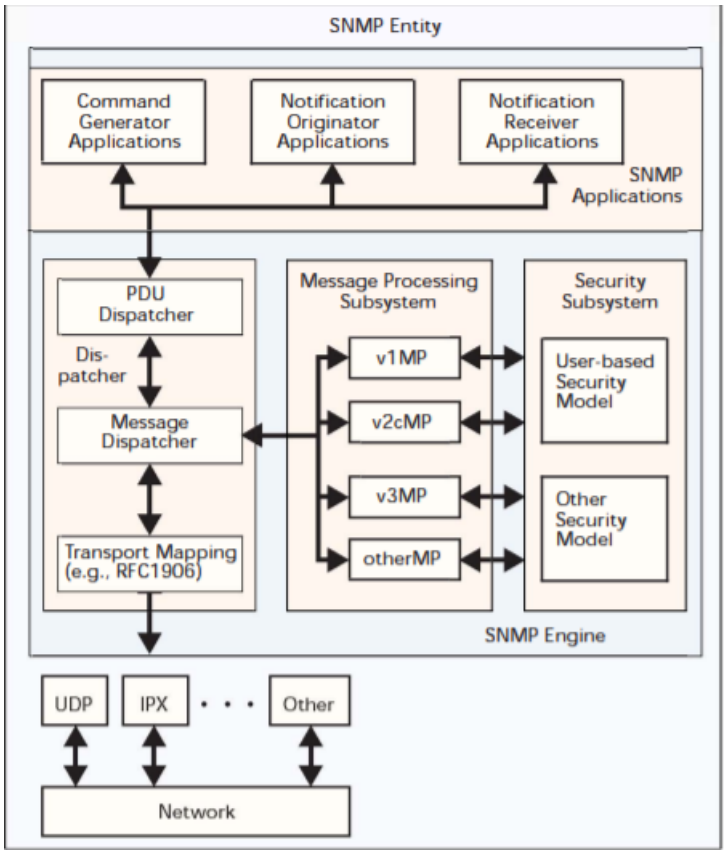
RFC-dokumentoinneista. Näitä dokumentteja on myös päivitetty myöhemmin ja kaikkien SNMP versioiden määrittelyt kattavatkin kokonaisuudessaan parisen kymmentä RFC-dokumenttia. SNMPv1 on alun perin määritelty RFC:issä 1065-1067 ja se on vanhuudestaan huolimatta edelleen laajasti käytössä. Uudempiakin versioita on olemassa, mutta koska uudempaan versioon siirtyminen on hankalaa uusien ominaisuuksien takia, eri SNMP versioita käytetään paljon rinnakkain [4]. Esimerkiksi paranneltu SNMPv2-versio SNMPv2c ei ole yhteensopiva SNMPv1 toteutuksien kanssa erilaisen viestiformaatin ja operaatiokutsujen takia. Tämä on johtanut uudempien versioiden rajalliseen levinneisyyteen. Lisäksi vasta SNMPv3:ssa on toteutettu kunnollista tietoturvaa tukevat ohjelmamoduulit; aiemmat versiot käyttävät salaukseen vain heikkoa yhteisömerkkijonoa SNMP-otsikossa.



Kuva 2. *SNMP-agentin moduulirakenne © 1998, IEEE [5].*

SNMP:n toiminta perustuu kahden SNMP-olion, SNMP-agentin ja -managerin, toiminnalle ja vuorovaikutukselle. SNMP-järjestelmässä on useita SNMP-agentteja, vähintään yksi SNMP manageri ja protokolla, jolla nämä elementit keskustelevat keskenään [6]. SNMP-oliot koostuvat ohjelmamoduuleista, jotka määrittävät olion toiminnan viestien käsittelyssä ja tietovarastojen hallinnassa [6].

Kuvan 2 SNMP-agentti toimii paikallisesti verkkolaitteella ja sen tehtävä on kerätä ja ylläpitää dataa verkkolaitteensa tarkkailtavista elementeistä. Verkkolaitteen elementit tunnistetaan lyhyellä kirjallisella kuvauksella (Object Descriptor) ja objektitunnisteella (OID, engl. object identifier) ja niistä kerätty tieto tallennetaan Management Information Base (MIB) –tietokantaan [7]. Verkkoelementeistä kerätyt tiedot voivat olla esimerkiksi anturidataa tai verkkosovittimen asetuksia ja tietoa laitteen verkkoliikennevirroista. MIB:n sisällä tieto jakautuu konteksteihin, jotka ovat esimerkiksi laitteen verkkosovittimia. Hallintatieto voi sijaita useassa eri kontekstissa, mutta sillä on aina yksikäsitteinen tunniste kunkin kontekstin sisällä [7].



Kuva 3. *SNMP-managerin moduulirakenne © 1998, IEEE [5].*

SNMP-agentteja hallinnoi yksi tai useampi SNMP-manageri, jonka rakenne on esitetty kuvassa 3. Managerin tehtävä on kerätä tietoa hallittavilta agenteilta ja tarjota käyttöliittymä, jolla SNMP-järjestelmää käytetään ja johon kerättyä dataa voidaan koostaa käyttäjän luettavaksi [5]. Manageri viestii agenttien ja muiden managerien kanssa käyttäen ASN.1-koodattuja SNMP-viestejä:

- **GetRequest**-viestillä manageri pyytää agentilta tiedot tietyllä ID:llä olevasta elementistä.

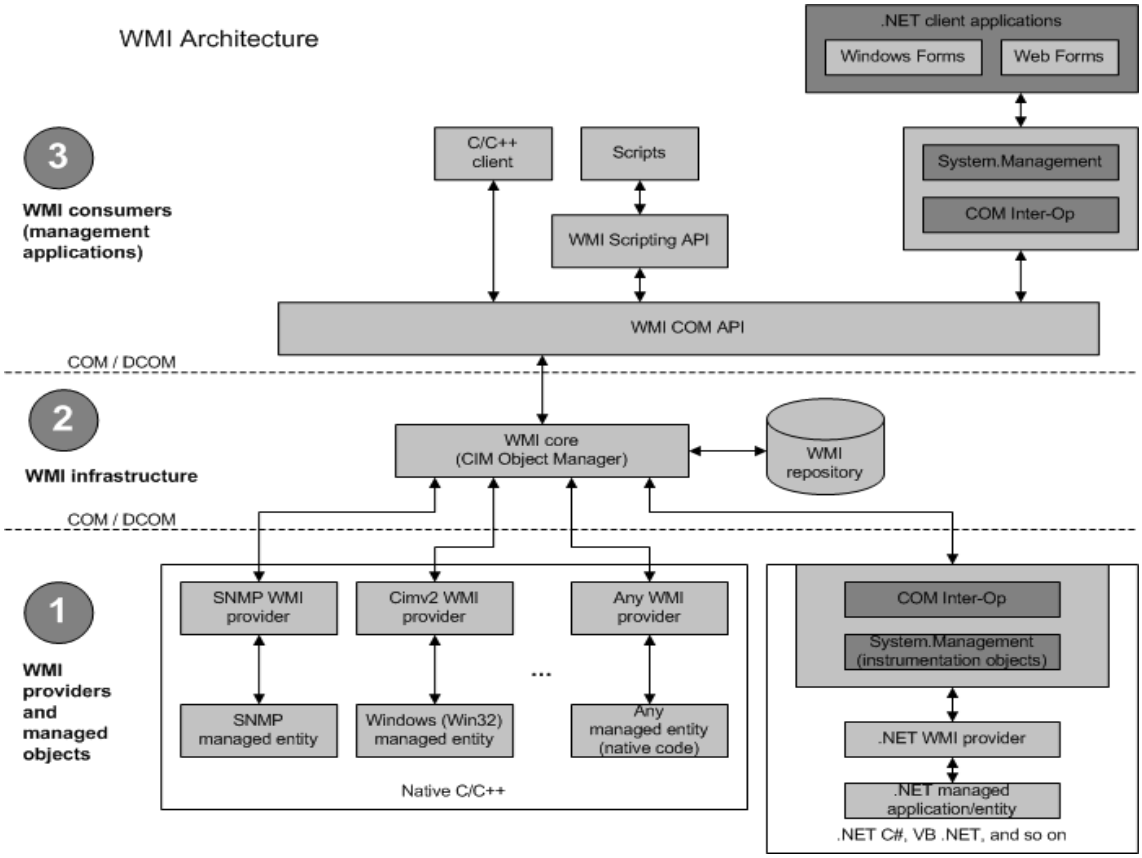
- **GetNextRequest**-viestillä manageri voi pyytää agentilta kysytystä elementistä seuraavaa elementtiä. Tätä viestityyppiä käytetään käymään MIB:tä läpi silmukkarakenteen omaisesti.
- **GetBulkRequest**-viesti on SNMPv2:ssa toteutettu, paranneltu GetNextRequest, jolla manageri voi pyytää agentilta useamman elementin kerralla.
- **Response**-viesti on vastaus GetRequest-, GetNextRequest- ja SetRequest-viesteihin. Viestin lähettää SNMP-agentti. Jos vastaus on lähetetty liittyen Get-viestiin, se sisältää tiedot managerin pyytämästä yhdestä tai useammastaelementistä. Viestissä on lisäksi ErrorStatus- ja ErrorIndex-kentät, joilla agentti ilmoittaa, onko viestin rakentamisessa tapahtunut virhe, ja mikä virhe on kyseessä. Tällainen virhe voisi esimerkiksi olla pyyntö elementistä, jota ei löydy agentin MIB:stä. SetRequest viestiin vastattaessa agentti vastaa viestillä, jossa on kopio vastaanotetusta SetRequest-viestistä ja virhekentät kertomaan, onnistuiko set-operaatio.
- **Trap**-viesti on agentin lähettämä viesti managerille, jossa se ilmoittaa jostain paikallisesta tapahtumastaan. Trap-viestit lähetetään niitä varten varattuun porttiin managerilla, mutta koska verkkoprotokollana on UDP, viestien vastaanottoa managerilla ei varmisteta.
- **Inform**-viesti on kuin trap-viesti, mutta SNMP-agentti toistaa sitä tietyin väliajoin, kunnes manageri kuittaa sen saaduksi. Inform-viesti on toteutettu SMNPv2:ssa.
- **Report Protocol Data Unit (PDU)** on SNMP-viesti, jolla SNMP-moottorit (engl. SNMP engine) viestivät toisilleen SNMP-viestien käsittelyssä ilmenevistä ongelmista, esimerkiksi koodaus- ja kryptausvirheistä. SNMP-moottori on SNMP-olion osa, joka on vastuussa viestien käsittelystä.
- **SetRequest**-viestillä SNMP-manageri pyytää agenttia muokkaamaan tietyllä ID:llä olevaa elementtiä viestin VariableBindings-kenttien mukaan. [8]

SNMP:llä on heikkouksia, joiden vuoksi sitä ei juuri käytetä varsinaisessa konfiguraatiohallinnassa. Ensinnäkin, standardoidut MIB:t tarjoavat vain vähän kirjoitusoperaatiota tukevia moduuleja ja uusien moduulien toteuttaminen on monimutkaista ja työlästä. Datan kerääminen on lisäksi nopeaa vain pienillä määrillä. Jos operaatio vaatii isomman tietomäärän hakua, kuten esimerkiksi suuren reititystaulun hakua, protokollan toiminta hidastuu merkittävästi. Tämä johtuu siitä, että MIB:lle tehtävät loogiset operaatiot voivat vaatia useamman atomisen operaation, mikä tarkoittaa, että toteutuksen on pidettävä tilakirjan-

pitoa, kunnes lopullinen operaatio joko onnistuu tai epäonnistuu. Asetuksia on myös vaikea saada talteen ja palauttaa virhetilanteissa ja SNMP:n datan kategorisointi vaikeuttaa kirjoitettavan ja luettavan datan, ja konfiguraatiodatan ja tiladatan erottelua. [9]

2.3 Windows Management Instrumentation

Windows Management Instrumentation (WMI) on Microsoftin kehittämä hallintajärjestelmä Windows-pohjaisille ympäristöille. Se kuvaa Distributed Management Task Forcen (DMTF) määrittämän Common Information Model (CIM) -mallin avulla Windows-järjestelmien hallintainformaation WMI-luokkien avulla. WMI-luokkia hallitsee WMI-tarjoaja, joka on ohjelma tai komentosarja, joka tuottaa tieto hallittavista olioista WMI-kuluttajille. Se tarjoaa myös ohjelmointirajapinnan tarjoajasovelluksia tai -komentosarjoja varten, sekä protokollan, jolla tietoa voidaan siirtää verkon yli. WMI on ollut käytettävissä Windows 95 -julkaisusta lähtien. [10][11]



Kuva 4. *WMI:n arkkitehtuuri [11].*

Kuvassa 4 nähdään tarkempi malli WMI:n arkkitehtuurista, joka eroaa kohtalaisesti SNMP:n arkkitehtuurista, vaikka päällisin puolin molemmat protokollat toimivat samankaltaisesti. WMI:n arkkutehtuuri koostuu kolmesta palasesta, jotka ovat tarjoajat ja hallittavat oliot, infrastruktuuri ja kuluttajat. WMI-tarjoajat keräävät tietoa hallittavista ele-

menteistään SMNP-agentin tavoin. Ne koostuvat Dynamic Link Library (DLL) -tiedostosta ja Managed Object Format (MOF) –tiedostosta, joka määrittää, mistä järjestelmän elementeistä tarjoaja kerää tietoa [11]. Hallittavat elementit voivat olla fyysisiä tai loogisia laitteita tai järjestelmiä, kuten muistia tai palveluja. Jokaiselle elementille on WMI-luokka, joka kertoo elementin ominaisuuksista ja toiminnoista, jos se niitä tukee. Luokkia on neljää eri tyyppiä:

- **WMI System Class** kattaa WMI:n perustoiminnallisuudesta vastaavat luokat ja mm. Windows-käyttöjärjestelmän tarjoajaluokan.
- **MSFT Class** sisältää muut Microsoftin WMI-luokat, kuten erilaiset käyttöjärjestelmälle tarkoitetut, täydentävät luokat.
- **CIM Class** CIM-kaavion pohjalta periytetty WMI-luokka. Tähän kategoriaan kuuluvat esimerkiksi käyttäjien itse tekemät luokat.
- **Standard Consumer Class** on kuluttajaluokka, joka reagoi ennalta asetettuihin tapahtumiin jollain toiminnolla. [12]

Seuraava osio WMI-arkkitehtuurista kuvassa 4 on infrastruktuuri. WMI-infrastruktuuri koostuu kahdesta palasta, ytimestä ja kuvauskannasta, ja se on osa Windows-käyttöjärjestelmää [13]. WMI-ydin on ohjelma, joka hoitaa WMI-viestien käsittelyn ja WMI-kuvauskanta sisältää hierarkkisesti kaikki hallittavat elementit. Kuvauskannassa on erilaisia nimiavaruuksia, joiden sisälle WMI-luokat on jaoteltu. Kun applikaatio tekee kyselyn nimiavaruuteen, WMI-ydin hakee tiedon kuvauskannasta, jos kysely tehdään staattiselle datalle tai ohjaa kyselyn tarjoajaluokalle, kun kysellään dynaamista dataa [13]. Staattinen data on esimerkiksi luokkien nimiä ja dynaamisella datalla tarkoitetaan jatkuvasti muuttuvaa dataa, kuten suorittimen kuormitustasoa tai komponenttien lämpötiloja.

Yleisesti ottaen WMI on helppo ottaa käyttöön verkkojen tarkkailua varten. WMI tulee valmiiksi Windows-käyttöjärjestelmään asennettuna, eikä verkon eri koneille ja reitittimille täten tarvitse erikseen asentaa agenttina toimivia sovelluksia, eikä verkossa tarvitse olla erillisiä hallintaolioita; mikä tahansa laite tukee asiakas- tai palvelinroolia tarpeen mukaan. WMI tukee myös etähallintaa verkon yli ja tarjoaa turvallisen yhteyden ja käyttäjätunnistuksen. WMI:n käytön suhteen suurin ongelma on kuitenkin se, että sitä tukee vain Windows-käyttöjärjestelmä, joka on harvinainen palvelinmaailmassa suhteessa kevyempään Linuxiin tai reititinvalmistajien omiin alustoihin kuten Cisco IOS:ään tai Juniperin Junos OS:ään.

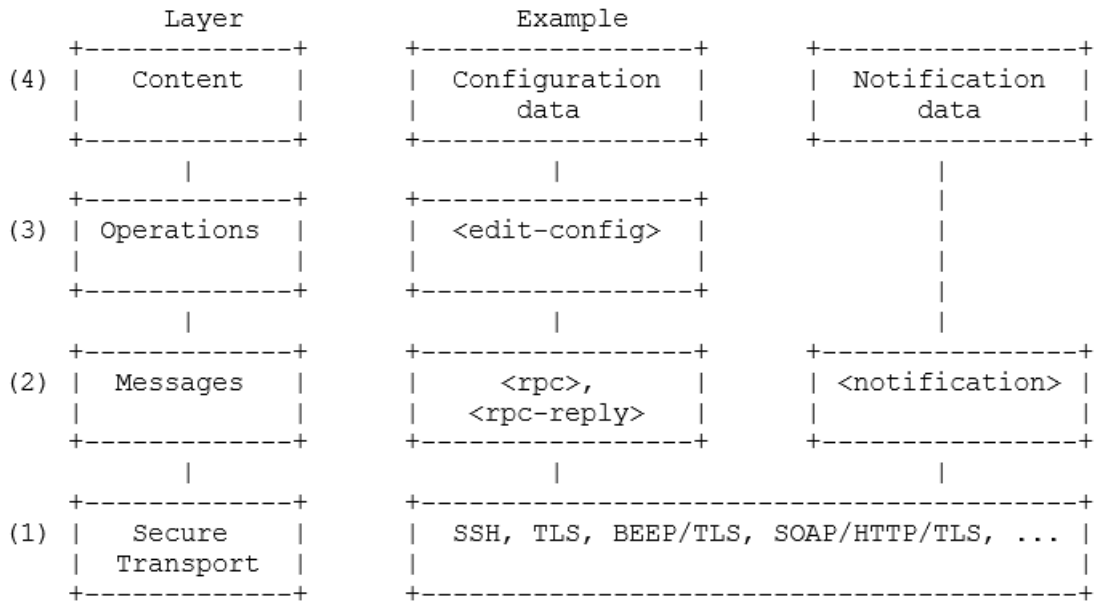
2.4 Network Configuration Protocol

Network Configuration Protocol (NETCONF) on IETF:n RFC 4741 -dokumentissa vuonna 2006 standardoima verkonhallintaprotokolla, joka mahdollistaa verkkolaitteiden konfiguraatioiden muokkaamisen ja tarkastelun. Protokollamäärittelyä on myöhemmin paranneltu RFC 6241 -dokumentissa. Sen kehittämisen päätavoite oli luoda laitevalmistajista riippumaton tapa hallinnoida verkon asetuksia [14].

NETCONF:n perustana on koko verkon laajuiset transaktiot, joilla laajojen verkkojen asetushallintaa ja palveluaktivointia saadaan yksinkertaisemmaksi verkkojen ylläpitäjille. Tässä yksinkertaistuksessa isossa roolissa on ollut myös operaatiosekvensoinnin ja virheidenkäsittelyn poistaminen ylläpitäjiltä niin sanotulla all-or-nothing-periaatteella, jossa kaikki asetusmuutokset tapahtuvat samanaikaisesti ja asetussetin tarkka toteuttaminen laitteella kuuluu laitteelle itselleen. Tähän periaatteeseen kuuluu myös, että laite validoi konfiguraation, testaa sen ja mikäli sen toiminnassa havaitaan virheitä, koko konfiguraatio hylätään. Tämä auttaa välttämään esimerkiksi SNMP:ssä koettuja ongelmia, joissa väärässä järjestyksessä annetut komennot rikkovat konfiguraation. [15]

NETCONF tarjoaa ohjelmointirajapinnan (API), jonka läpi verkkolaitteet voivat vastaanottaa asetussettejä asiakaslaitteilta NETCONF-istunnon kautta käyttäen NETCONF:n Hello-viestejä [16]. Internet Assigned Numbers Authority (IANA) on varannut NETCONF-istunnoille portin 830. Protokolla tarjoaa rajapinnan laitteen komentamiseen, mikä pienentää toteutuskustannuksia ja helpottaa laitetoiminnallisuuden päivittämistä ja käyttöönottoa [16]. Tämä mahdollistaa myös palvelimen tukemien lisätoimintojen oppimisen dynaamisen, koska NETCONF tarjoaa standardoidun tavan ilmoittaa niistä Yet Another Next Generation (YANG) -mallien avulla [16]. YANG on datan mallinnuskieli, joka on kehitetty NETCONF-protokollaa varten ja sillä kuvataan helposti luettavassa muodossa laitteen sisäinen asetushierarkia ja tuetut ominaisuudet tarkkaan määriteltyjen (engl. well-defined) Extensible Markup Language (XML) -dokumenttien avulla [17]. Tätä hierarkiaa vasten voidaan vertailla potentiaalisia konfiguraatioita ja asettaa syntaktisesti oikeita konfiguraatioita laitteille, sekä suorittaa ylläpidollisia operaatioita laitteella. YANG on määritelty RFC-dokumenteissa 6020, 6021, 6087, 6110, 6244 ja 6643. YANGin tarkempi läpikäynti ei kuulu tämän työn piiriin.

NETCONF:n protokollakehys jakautuu neljään kerrokseen kuvassa 5 esitetyllä tavalla. Ne ovat turvallinen kuljetuskerros, viestikerros, operaatiokerros ja sisältökerros [16]. Kerrokset jakavat yhteyden loogisiin osiin toiminnallisuuden suhteen.



Kuva 5. *NETCONF:n protokollakehyksen kerrokset [16].*

Ensimmäinen kerros on turvallinen kuljetuskerros. Turvallinen kuljetuskerros kuvaa varsinaisen kuljetusprotokollan, jolla NETCONF-yhteys on toteutettu, esimerkiksi Secure Shell (SSH) tai Transport Layer Security (TLS). Kuljetusprotokollan tulee olla yhteyspohjainen ja tarjota käyttäjän työkalut käyttäjän tunnistamiseen. Väärin tunnistettu käyttäjä voi aiheuttaa suurta vahinkoa verkolle esimerkiksi tahallisilla, viallisilla asetuksilla. NETCONF-istunto itsessään vaatii luotettavan tiedonsiirron ja pakettien vastaanoton oikeassa järjestyksessä, joten kuljetuskerroksen on mahdollistettava se muun muassa pakettien sekvenssinumeroiden ja tarkistussumman seurannalla. Toinen kerros kuvaa Remote Procedure call (RPC) -yhteyttä, johon sisältyy RPC-kehysympäristöön liittyvät vaiheet, kuten RPC-viestien vaihto. Kolmannessa kerroksessa siirrytään NETCONF-viestin sisältöön, josta ensimmäisenä on operaatiokerros ja viimeisenä sisältökerros. Operaatiokerroksella kuvataan mitä operaatiota halutaan käyttää ja sisältökerros tarjoaa muun muassa parametrit valitun operaation suorittamiseen. Kuvassa 4 on esimerkki get-config-viestistä, josta voi hahmottaa kolme näistä kerroksista (ei itse TCP-yhteyttä). [16]

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
  </get-config>
</rpc>
```

Kuva 6. *Esimerkki RPC-viestistä: get-config [16].*

NETCONF-istunto noudattaa asiakas-palvelinmallia, jossa asiakaslaite muodostaa yhteyden palvelimeen TCP-yhteyden läpi. Istunto noudattaa RPC-paradigmaa, jossa kaksi

eri osoiteavaruuksissa olevaa oliota voivat antaa toisilleen korkean tason käyttöjärjestelmäkomentoja [18]. RPC-viestit on toteutettu tarkkaan määritellyillä XML-dokumenteilla, jotka muodostetaan laitteen tarjoamien YANG-mallien ja NETCONF:n määrittelyjen pohjalta.

NETCONF erittelee datan kahteen eri tyyppiin: konfiguraatio- ja tilatietoon. Konfiguraatiodata sisältää tiedon laitteen asetuksista ja tiladata käsittää laitteen operatiivisen statuksen ja protokollastatistiikan. Tällaisella erittelyllä vältetään muun muassa SNMP:ssä olevat ongelmat datan erottelussa [9]. Lisäksi tämän erittelyn ansiosta vältetään seuraavat ongelmat:

- Konfiguraatioiden vertailu olisi hankalaa, jos dataseteissä on mukana vertailulle epäoleellista tilatietoa ja statistiikkaa.
- Datasetit olisivat hyvin suuria, jos kaikki tieto välitettäisiin jokaisella pyynnöllä.
- Operaatiot voisivat sisältää pyyntöjä ylikirjoittaa vain luettavissa olevaa dataa.
- Arkistoidun datan palauttaminen laitteelle (rollback) olisi hankalaa, jos seassa voisi olla vain luettavissa olevaa dataa. [16]

Tiedon erittelyn ansiosta saadaan asetuspyyntöoperaatioista suodatettua pois paljon epäoleellista dataa, koska laitteen tilatiedot eivät tule siinä mukana.

NETCONF määrittää kolme erilaista tietovarastoa, joita asiakaslaite käsittelee NETCONF-istunnossa operaatioilla. Nämä tietovarastot ovat running, candidate ja startup, joista running sisältää laitteen ajossa olevan konfiguraation, candidate sisältää mahdollisen, tulevaisuudessa käytettävän konfiguraation ja startup sisältää laitteen käynnistyessä käytettävän konfiguraation. Erilaisia operaatioita, joita voidaan suorittaa tietovarastoille ovat esimerkiksi get, get-config, edit-config, copy-config ja delete-config. Get-operaatio hakee kaiken datan ja get-config -operaatio vain konfiguraatiodatan halutusta tietovarastosta. Edit-config -operaatio muokkaa haluttua tietovarastoa, tavallisesti candidate-tietovarastoa. Copy-config -operaatio kopioi koko tietovaraston toiseen tietovarastoon ylikirjoittaen vanhan tietovaraston tai luoden uuden, jos sitä ei vielä ollut olemassa. Tätä operaatiota käytetään tavallisesti kandidaattikonfiguraation tallentamiseen ajokonfiguraatioksi, kun laite on ensin tarkastanut kandidaattikonfiguraation kelvollisuuden. Muita operaatioita, jotka on määritelty NETCONF:n pohja RFC:ssä 4741, ovat lock, unlock, close-session ja kill-session, joita käytetään istunnon hallintaan ja tietovarastojen lukitsemiseen niiden käsittelyn ajaksi. NETCONFin kautta voi myös ajaa laitteilla muita ylläpidollisia operaatioita, mikäli ne on määritelty laitteen YANG-malleissa. [19]

3. VERKKOJEN TESTAUS

Verkkoteknologioiden kehittyessä, kasvaessa ja yleistyessä tietoverkot monimutkaistuvat ja siirtyvät lähemmin osaksi yritysten toimintaa, kriittistä infrastruktuuria ja tavallisten ihmisten jokapäiväistä elämää. Yritysten tietojärjestelmät kytkeytyvät yhä tiiviimmin niiden liiketoimintaan. Kannettavien teknologioiden kuten kannettavien tietokoneiden ja älypuhelinten yleistyminen on johtanut mobiliteettivaatimusten kasvuun asioiden internetistä (IoT) puhumattakaan. Kaikki tämä asettaa alati kasvavia suorituskyky- ja palvelunlaatuvaatimuksia (Quality of Service) verkkoprotokollille ja järjestelmille, sillä huonot järjestelmät ja hitaat palvelut maksavat yrityksille rahaa ja asiakkaita.

Avain hyvään toteutukseen on onnistunut testaus. Verkkotestausta voidaan tehdä erilaisista näkökulmista ja erilaisilla laitteilla sekä menetelmillä. Näiden sopivan yhdistelmän valintaan vaikuttaa testausta suorittava taho, testausasetelma ja testattavan verkon tai elementin käyttötarkoitus. Esimerkiksi laajojen yritysverkkojen ja yksityisten kotiverkkojen testaus eroaa toisistaan muun muassa tuettavien verkkoteknologioiden vuoksi. Usean protokollan tai järjestelmän päällekkäisyys ja yhteistoiminta aiheuttavat useampia mahdollisia hajoamiskohtia ja pullonkauloja suhteessa yksinkertaisempaan verkkoon. Tällainen protokollien tai järjestelmien vuorovaikutus monimutkaistaa nopeasti testausvaatimuksia.

3.1 Testausnäkökulmia

Testausta voidaan suorittaa järjestelmän eri kerroksilla pienempinä tai suurempina kokonaisuuksina. Verkkotestauksessa tämä tarkoittaa IP-protokollapinon eri osia, kuten laite-elementtejä ja antenneja. OSI-malli koostuu monesta kerroksesta, joten on luontaista jakaa testausta niiden mukaan. Esimerkiksi fyysisellä kerroksella voidaan tarkastella muun muassa signaalin häiriölähteitä, bittivirhesuhdetta ja kaapelien ylikuulumista. Ylempien kerrosten testauksessa voidaan keskittyä esimerkiksi tarkastelemaan protokollatoteutuksien RFC:n mukaisuutta ja virhetilanteita. Kun testaus keskittyy tiettyyn protokollakerrokseen, puhutaan yksikerroksisesta lähestymistavasta (engl. single layer approach), joka voi auttaa jakamaan monimutkaisen verkon testausta pienempiin palasiin. [20]

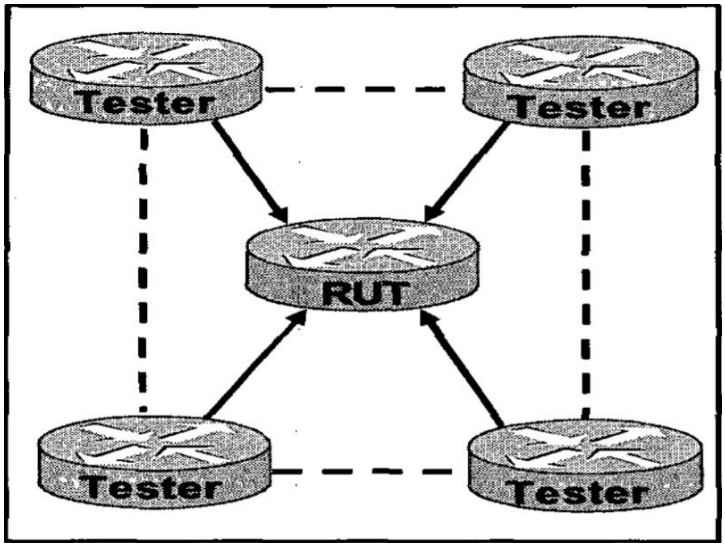
Tietojärjestelmien yksittäisten osien ja kerrosten testaus erillisesti ei kuitenkaan ole riittävää koko järjestelmän toiminnan takaamiseksi. Tämä johtuu esimerkiksi siitä, että järjestelmän osien toteutukset eivät välttämättä ole yhteensopivia. Eri osien yhtäaikainen

toiminta voi myös esimerkiksi aiheuttaa viiveitä järjestelmän eri osissa, kuten käyttöjärjestelmän ytimessä (engl. kernel), joihin ei olla osattu varautua suunnitteluvaiheessa. Tätä varten tarvitaan monikerroksista lähestymistapaa (engl. cross layer approach), jossa tarkastellaan suurempia kokonaisuuksia, jotka koostuvat usealla kerroksella toimivista järjestelmän osista [20].

Testausta voi tehdä ylhäältä alaspäin tai alhaalta ylöspäin. Molemmat lähtökohdat ovat tavallisia esimerkiksi ohjelmistopuolella, jossa tilanteesta tai ohjelmoijasta riippuen voi olla perusteltua käyttää kumpaa tahansa lähestymistapaa. Tietoverkkojen tapauksessa IP-protokollapinon alapäässä tapahtuvat virheet voivat siirtyä ylemmille kerroksille erikoisilla tavoilla, mikä voi vaikeuttaa virhelähteen löytämistä. Ylempiä kerroksia testattaessa joudutaan olettamaan, että alemmat kerrokset toimivat oikein. Jos alempien kerrosten toimintaan ei ole ensin testattu tai niihin ei voi luottaa, ylemmiltä kerroksilta saadut testitulokset eivät ole luotettavasti liitettävissä testattuun ylempään kerrokseen. Tästä syystä on suositeltavaa aloittaa verkkotestaus mahdollisimman alhaalta protokollapinosta. [20]

3.2 Testausmenetelmiä

Nykyiset tietoverkot ovat monimutkaisia ja niiden toiminnan ennustaminen on vaikeaa etenkin solmukohtien lukumäärän kasvaessa tai uusia protokollia käyttöönotettaessa. Protokollien toteutusten tarkastaminen on myös tärkeää, jotta voidaan varmistua siitä, että niiden toiminta on standardien mukaista [21]. Ei ole järkevää tehdä laajoja toteutuksia suoraan, sillä kustannukset ovat suuria eikä kaikkea ole välttämättä pystytty huomioimaan suunnittelutilanteessa [22]. Siksi tarvitaan menetelmiä, joissa testaus voidaan suorittaa irrallisesti tuotantoverkoista.



Kuva 7. *Esimerkkitopologia reitittimen testaamiseen liikennegeneraattoreilla © 2004, IEEE [21].*

S. Kalidindi et al. käsittelee vuonna 2004 julkaistussa konferenssipaperissa "'Real Life' System Testing of Networking Equipment" tapaa testata korkean suorituskyvyn verkkolaitteita käyttämällä hyväksi liikennegeneraattoreita, jotka pystyvät jäljittelemään todellisten verkkojen liikennekuormia. Paperissa esitelty liikennegeneraattori on fyysinen laite, joka on suunniteltu emuloimaan verkkoliikennettä erilaisilla protokollilla ja liikennemäärillä. Liikennegeneraattori pystyy myös tarkastelemaan testattavan laitteen ulostuloa tärkeiksi valituilla parametreilla, kuten viiveellä tai pakettien katoamisella. Liikennegeneraattoreita voidaan kytkeä yhteen testattavan laitteiston kanssa muodostamaan erilaisia topologioita, kuten yllä kuvassa 6, joiden avulla testattavaa laitteistoa voidaan kuormittaa; muuttamalla verkkoja, liikennemääriä ja protokollia dynaamisesti testauksen aikana liikennegeneraattori voi rasittaa reitittimen ohjaustasoa (engl. control plane) hyvin monipuolisesti. Dynaamiset verkkojen muutokset ja reittien uudelleenlaskenta ovat hyvin tavallisia tuotantoverkoissa, joten reitittimen on tärkeää selviytyä niistä kuormituksen alaisena ilman merkittävää suorituskyvyn menetystä. Testauksen aikana liikennegeneraattori varmistaa suorituskykyparametreja mittaamalla, ettei reitittimen suorituskyky heikkene. [21]

Laboratorio-oloissa pystytään harvoin rakentamaan suuria verkkoja fyysisistä verkkolaitteista kustannus- ja tilarajoitteiden takia. Erityisesti korkean suorituskyvyn laitteisto voi olla hyvin kallista, jolloin nimenomaan suuriin verkkoihin tarkoitettujen laitteiden hankinta riittävässä mittakaavassa testausta varten on kannattamatonta tai jopa mahdotonta. Pienten verkkojen ongelma taas on, että ne harvoin vastaavat toiminnallisuudeltaan ja liikennemääriltään suuria tuotantoverkkoja, jolloin testiverkon käyttäytyminen ei vastaa tuotantoverkon käyttäytymistä.

Laitteiston aiheuttamia kustannuksia voidaan vähentää erilaisilla virtualisointiteknologioilla, kuten laitevirtualisoinnilla ja virtualisointipalvelimilla (engl. hypervisor). Laitevirtualisoinnilla on mahdollista jakaa yksittäinen fyysinen laite useaan virtuaaliseen laitteeseen, jolloin pienemmällä laitemäärällä voidaan rakentaa loogisesti suuria verkkoja. Virtualisointipalvelin taas on ohjelma, joka tarjoaa virtuaalilaitteille vieraskäyttöjärjestelmän ja hallinnoi niiden resursseja [22]. Testauksen näkökulmasta virtuaaliset liikennegeneraattorit ovat kuitenkin yleensä yksinkertaisempia kuin fyysiset liikennegeneraattorit, eikä niillä ole välttämättä käytettävissään fyysisten liikennegeneraattorien erikoistunutta laitteistoa, koska ne hyödyntävät suoraan isäntänsä resursseja. Tästä johtuen testauslogiikkaa joudutaan siirtämään pois liikennegeneraattoreilta muille tahoille, kuten muille virtuaalilaitteille.

Virtualisointiteknologioita käyttämällä voidaan emuloida ja simuloida suuria verkkoja huomattavasti pienemmällä laitemäärällä. Simulointi tarkoittaa matemaattista mallintamista, jossa matkitaan oikean järjestelmän toimintaa [22]. Verkkosimulaatioissa voidaan esimerkiksi jäljitellä tuotantoverkkojen liikennettä, joka voidaan sitten antaa testattavalle laitteistolle sisäänmenona. Näin voidaan tehdä esimerkiksi käyttämällä virtuaalisia liikennegeneraattoreita jotka simuloivat tuotantoverkon liikennettä testausverkkoon päin.

Simulaatioilla ei saada oikeaa kuvaa verkon toiminnasta, koska simulaatio on vain yhtä tarkka kuin sen matemaattinen malli, eikä se ota kantaa siihen, kuinka liikenne varsinaisesti syntyy verkkosolmuissa. Tarkemman kuvan saamiseksi verkko täytyy emuloida, jolloin solmuina toimivien reitittimien sisäistä logiikkaa voidaan matkia esimerkiksi virtuaalireitittimellä. Tällöin virtuaalireititin toimii samalla tavalla ohjelmallisesti kuin oikeakin reititin, mutta sitä ajetaan eri laitteistolla eikä fyysistä reititintä tarvita. Näin voidaan testata suuriakin verkkoja täysin virtuaalisesti ja yhä varmistua siitä, että ne käyttäytyvät samoin kuin tuotannossa. [22]

3.3 Testauslaitteita

Erilaiset instrumentit antavat mahdollisuuden tarkkailla verkon tapahtumia. Tämä on oleellista sekä laboratoriotestauksessa että tuotantoverkoissa virheiden etsimiseksi ja palvelun laadun takaamiseksi.

Instrumentit voidaan luokitella taktisiin ja strategisiin instrumentteihin. Strategiset instrumentit on integroitu osaksi varsinaista verkkoa ja niiden tehtävä on tukea verkon toimintaa havaitsemalla ja hälyttämällä tapahtumista, jotka vaativat ylläpidollista huomiota. Näihin järjestelmän osiin kuuluvat esimerkiksi linjaprotokollat, verkonhallintaprotokollat ja instrumentit joilla jatkuvasti tarkkaillaan verkon toimivuutta. Taktiset instrumentit taas ovat liikuteltavia laitteita, jotka ovat tarvittaessa kytkettävissä verkkoon vikatilanteiden selvittämiseksi. Taktisia instrumentteja ovat esimerkiksi oskilloskoopit, jotka voidaan kytkeä verkkoon kiinni signaalin häiriöiden selvittämiseksi. [20]

Testauslaitteet toimivat aina jollain kerroksella. Ne voidaan jakaa karkeasti fyysisellä kerroksella toimiviin laitteisiin ja protokolla-analysaattoreihin, jotka toimivat ylemmillä kerroksilla. Fyysisellä kerroksella toimivat laitteet analysoivat varsinaista tietoliikennesignaalia. Niitä on monia, mutta muutamia yleisiä ovat:

- oskilloskooppi
- spektrianalysaattori
- kaapelitesteri

Oskilloskoopilla saadaan tietoa signaalin ominaisuuksista, kuten jännitteestä, taajuudesta ja kohinasta. Spektrianalysaattorilla saadaan tietoa signaalista tietyllä taajuusalueella. Spektrianalysaattori on erityisen tärkeä elektromagneettisten häiriöiden selvittämiseksi, koska sillä voidaan selvittää häiriön taajuusalue. Kaapelitesterillä taas voidaan testata kaapelin ominaisuuksia, kuten resistanssia ja ylikuulumista. [20]

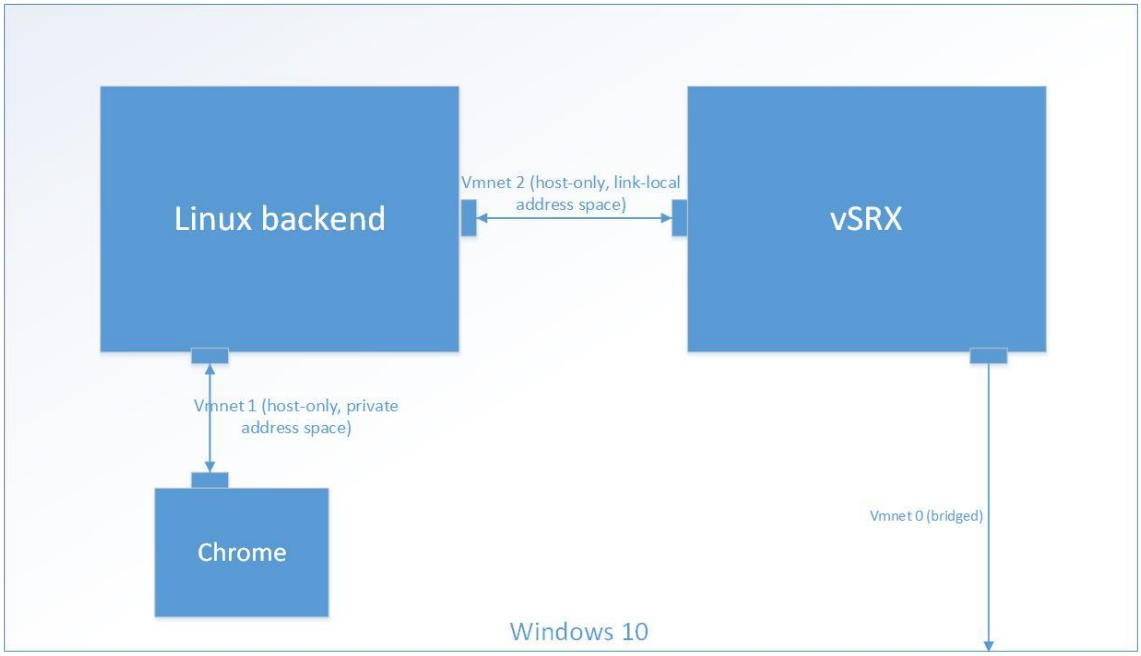
Protokolla-analysaattorit toimivat linkkikerrokselta aina sovelluskerrokselle asti. Niillä voidaan tarkastella protokollakehysrakennetta ja kehyksiä, ja siten varmistaa, että protokolla toimii odotetulla tavalla. Protokolla-analysaattorit toimivat sillä kerroksella, jolla niiden tarkastelema protokollakin toimii. Protokolla-analysaattorit voivat olla ohjelmisto, erikoislaitteisto tai yhdistelmä molempia. [20]

4. CASE: VERKKOTESTERI

Työn ohessa on toteutettu verkkotesteri, joka pystyy analysoimaan toiminnallisuutta verkossa, johon se on kytketty. Näin saadaan käsitys testattavan verkon toimintakyvystä.

4.1 Arkkitehtuuri

Verkkotesteri on toteutettu Windows-päätelaitteena, jossa ajetaan kahta virtuaalikonetta virtualisointipalvelimella. Toinen virtuaalikoneista on Juniper vSRX -virtuaalipalomuuri ja toinen on Linux-palvelin. Linux-palvelin tarjoaa selaimen välityksellä käyttäjälle käyttöliittymän, jonka kautta voidaan suorittaa testejä verkkotesteriin kiinnitetylle verkolle.



Kuva 8. *Verkkotesterin järjestelmäarkkitehtuuri.*

Kuvassa 8 havainnollistetaan verkkotesterin järjestelmäarkkitehtuuria. Kuvan VMnet1 kuvaa yksityistä osoiteavaruutta, jonka kautta selaimella saa yhteyden Linux-palvelimeen. Linux-palvelimella on yhteys vSRX-palomuurireitittimeen VMnet2:n kautta NETCONF-istunnon avulla, jonka välityksellä palvelin komentaa vSRX:ää testejä varten. Sekä VMnet1 että VMnet2 ovat virtualisointipalvelimen luomia virtuaaliverkkoja, jotka on eristetty palomuurilla vSRX:n virtuaalisovittimesta Vmnet0, joka on kytketty testattavaan verkkoon. Palomuurieristys tarvitaan, jotta estetään yhteydet testattavasta verkosta verkkotesteriin, sillä avoimet portit verkkotesterissä olisivat tietoturvariski. Linux palvelimen ja vSRX:n välinen verkko on asetettu link local -osoiteavaruuteen (169.254.0.0/16),

jotta verkkojen törmäys runkoverkon suuntaan estyy. VMnet0 on lisäksi sillattu testattavaan verkkoon, mikä estää sovittimen näkyvyyden päätelaitteen käyttöjärjestelmälle. Siltauksella estetään suora yhteydenotto päätelaitteelta virtuaalipalomuuriin ilman next hop -reititintä tai pääkäyttäjän salasanaa. vSRX:llä käytetään osoitteenmuunnosta (engl. Source Network Address Translation, SNAT) testattavan verkon suuntaan lähteville yhteyksille, mikä piilottaa Linux-palvelimen IP-osoitteen ja estää virtualisointipalvelimen sisäisten verkkojen vuotamisen testattavaan verkkoon.

4.2 Verkkoanalyysi

Verkkotesterillä tehtävä analyysi pystyy todentamaan verkkolaitteiden saavutettavuuden, sekä verkkotesteriin kytketyn verkon tuen erilaisille protokollille. Verkkoanalyysi suoritetaan Linux-palvelimen ja vSRX virtuaalipalomuurin yhteistyönä käyttäen hyväksi NETCONF-protokollaa. NETCONF-protokolla valittiin käyttöön laajan toiminnallisuutensa, vSRX:n tuen ja Linux-alustan perusteella. Tämän työn tekohetkellä verkkotesterin analyysi tukee testejä seuraaville protokollille:

- OSPF-reitit ja -naapuruus
- Käyttäjän asettamien verkkolaitteiden saavutettavuus
- PIM-naapuruus ja BSR-reitittimet
- NTP-palvelimen ja -ajan haku

Jotta tiettyjen protokollien toimivuus voidaan todentaa, verkkotesteri tarvitsee verkosta joitain esitietoja analyysiä varten. Nämä tiedot ovat OSPF Hello interval ja Dead Interval, jotta verkkotesteri osaa muodostaa OSPF-naapuruuden testattavan verkon reitittimen kanssa. Jos linkissä käytetään autentikointia, vaaditaan naapuruuden muodostamiseen lisäksi autentikoinnin käyttämä hajautusalgoritmi, avain-ID ja salasana. Muut protokollatestit eivät vaadi esitietoja, mutta verkkotesteri vaatii IP-osoitteen testattavan verkon reitittimen verkosta.

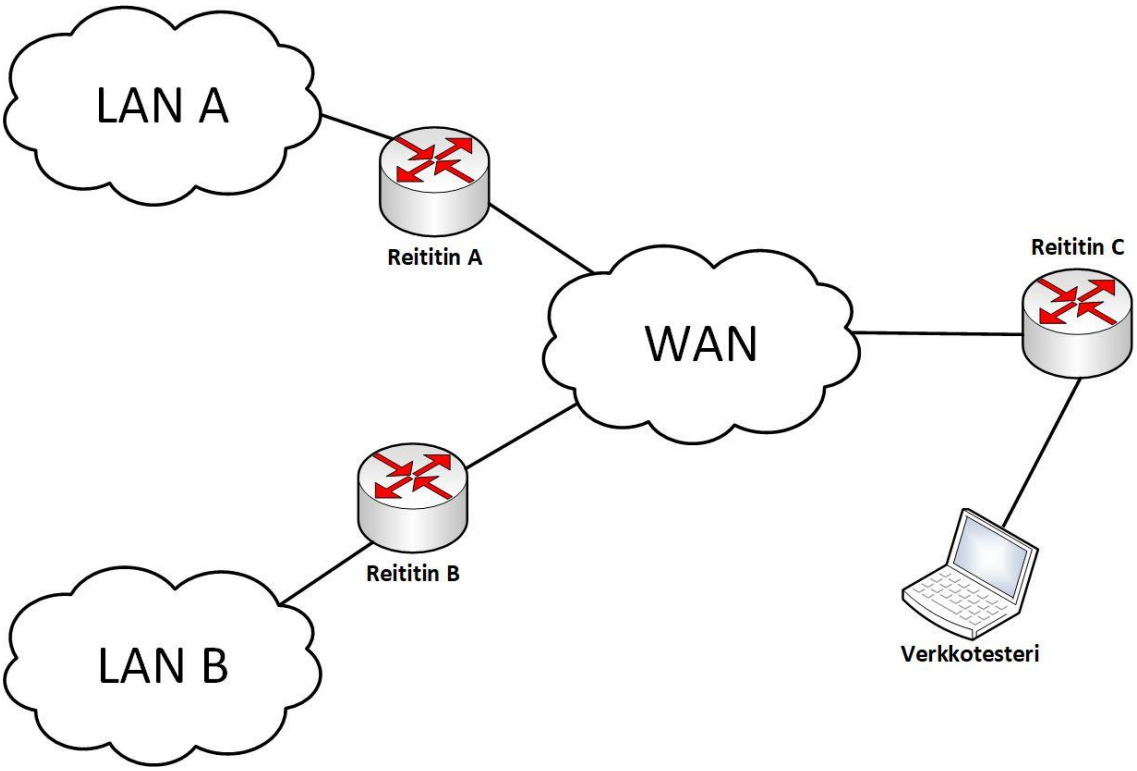
Verkon analyysi tapahtuu kahdessa vaiheessa, joista ensimmäisen suorittaa vSRX-reititin ja toisen suorittaa Linux-palvelin. Ensimmäisessä vaiheessa reititin muodostaa OSPF- ja PIM-naapuruudet testattavan verkon reitittimen kanssa. PIM-naapuruudella voidaan todentaa verkon mahdollinen multicast-reitityskyvykkyys ja saadaan tiedot bootstrap-reitittimestä (BSR-reitittimestä). OSPF-naapuruudella todennetaan naapuruusparametrit, joihin kuuluvat Hello Interval, Dead Interval, ja verkkomaski. Lisäksi tarkistetaan autentikointiin liittyvät parametrit avain-ID, hajautusalgoritmi ja salasana, jos linkissä käytetään salasanaa.

Ensimmäisen vaiheen OSPF-naapuruuden kautta saadaan reitit analyysin toista vaihetta varten. Tämä tarkoittaa sitä, että jos naapuruutta ei jostain syystä saada muodostettua, esimerkiksi virheellisten parametrien vuoksi, verkkoanalyysin toista vaihetta ei voida suorittaa.

Kun ensimmäinen vaihe on suoritettu ja jos OSPF-naapuruuden muodostus onnistui, verkkotesteri suorittaa analyysin toisen vaiheen. Toisessa vaiheessa Linux-palvelin todentaa käyttäjän esiasettamien verkkolaitteiden saavutettavuuden. Näihin kuuluvat NTP-palvelin, sekä yksittäiset muut IP-osoitteet, joiden saavutettavuus verkossa halutaan todentaa. NTP todetaan toimivaksi, mikäli NTP-palvelin löydetään ja siltä saadaan NTP-aika. Muiden IP-osoitteiden saavutettavuus varmistetaan saavutettavuustestillä.

Kun saavutettavuustestit on suoritettu, Linux-palvelin kerää ja esittää tulokset käyttöliittymässä. Analyysin ensimmäisen vaiheen data tallentuu vSRX:lle reitittimen tietoaineistoiksi, jotka Linux-palvelin hakee vSRX:ltä NETCONF-istunnon läpi. Toisen vaiheen tulokset ovat Linux-palvelimella itsellään, koska se itse suoritti kyseisen vaiheen. Tulokset kootaan yhteen käyttöliittymässä, jossa kerrotaan onnistuivatko testit ja saatiinko kaikki esiasetetut IP-osoitteet saavutettua.

5. VERKKOTESTERIN AJO LABORATORIOVERKOSSA



Kuva 9. *Laboratorion verkkotopologia.*

Tässä luvussa käydään läpi esimerkkiajo verkkotesterillä laboratorioverkossa. Verkko koostuu kuvan 9 mukaisesta kolmesta reitittimestä, WAN-verkosta, kahdesta LAN-verkosta ja verkkotesteristä. Verkossa on kaksi palvelua, yksi liikenteelle ja toinen hallintayhteyksille. Reititykseen käytetään salasanalla suojattua OSPF-reititystä, jossa käytetään numeroimattomia sovittimia (engl. unnumbered interface).

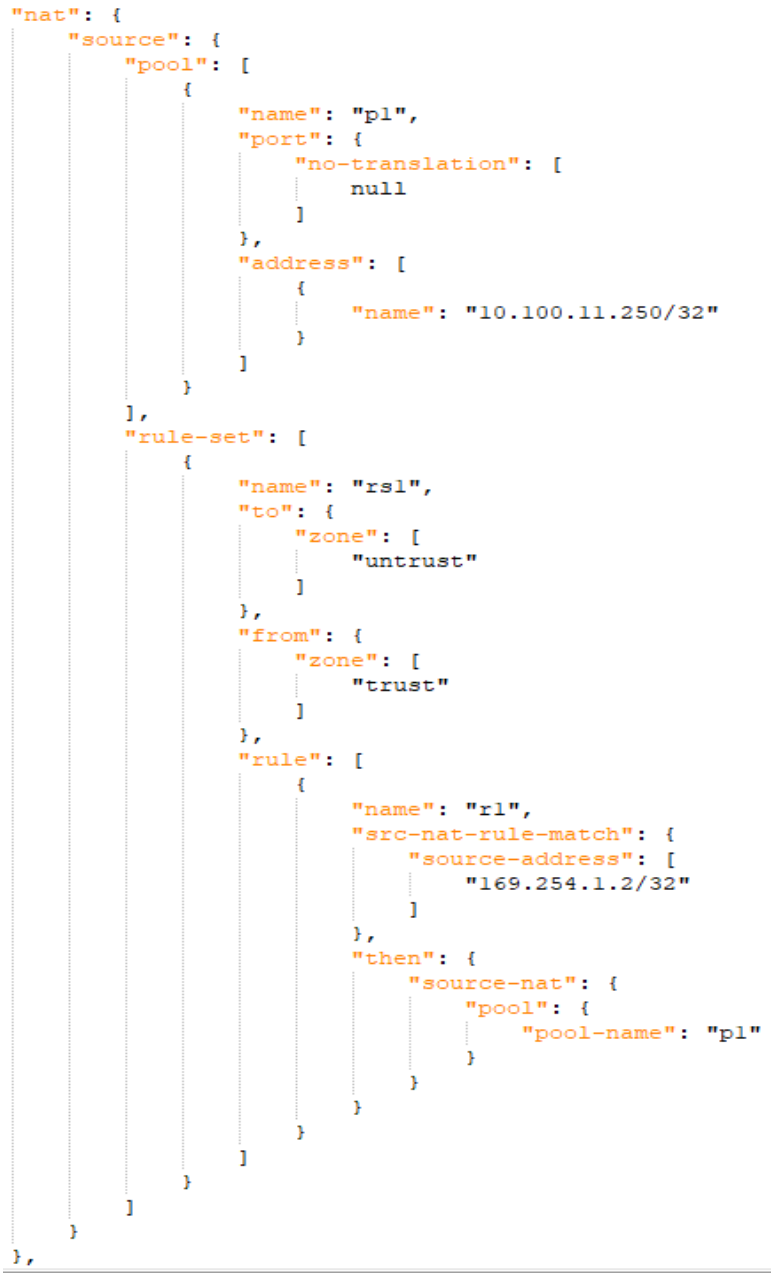
Käyttäjä kiinnittää verkkotesterin reititin C:n LAN:iin ja asettaa verkkotesterin osoitteen samaan verkkoon reititin C:n kanssa joko manuaalisesti tai automaattisesti Dynamic Host Configuration Protocol (DHCP) -protokollan avulla. Tämän jälkeen käyttäjä avaa verkkotesterin asiakassovelluksen ja asettaa sen kautta verkon esitiedot, jotka ovat OSPF-protokollan salasana ja lista laitteista, joiden saavutettavuus halutaan testata. Verkkotesterillä voi testata samalla ajolla useampia verkossa rinnan toimivia palveluita, jotka on eristetty toisistaan Virtual Routing and Forwarding (VRF) -virtuaalireitityksellä. Jokaiselle palvelulle tulee antaa aiemmin mainitut esitiedot erikseen. Kun esitiedot on annettu, käyttäjä käynnistää verkkotestin.

```
{
    "name": "ge-0/0/0",
    "unit": [
        {
            "name": "1306",
            "vlan-id": "1306",
            "family": {
                "inet": {
                    "unnumbered-address": {
                        "source": "lo0.31"
                    }
                }
            }
        }
    ],
    "vlan-tagging": [
        null
    ]
},
{
    "name": "lo0",
    "unit": [
        {
            "name": "31",
            "family": {
                "inet": {
                    "address": [
                        {
                            "name": "10.100.11.250/32"
                        }
                    ]
                }
            }
        }
    ]
}
```


Kuva 10. *Liikennepalvelun numeroimattoman sovittimen asetukset.*

Verkkotesterin Linux-palvelin asettaa NETCONF:n avulla sisäisen virtuaalireitittimen asetukset liikennepalvelulle, jotka esitetään tämän työn liitteessä. Linux-palvelin saa tiedot virtuaalireitittimeltä pyytämällä siltä kyseisen palvelun asetukset NETCONF:lla. Kuvassa 10 nähdään esimerkiksi palvelun numeroimattoman sovittimen asetukset. Linux-palvelin asettaa loopback-sovittimen osoitteen, joka lainataan WAN:n suuntaiseen sovittimeen.

Asetusten laiton jälkeen virtuaalireitittimen OSPF- ja PIM-instanssit synkronoituvat reitittin C:n vastaavien protokollainstanssien kanssa. Linux-palvelin hakee OSPF-naapurit, BSR-reitittimien ja PIM-naapurien tiedot asiakassovellukselle käyttämällä verkkotesterin virtuaalireitittimen NETCONF-rajapinnan funktioita get-ospf-route-information, get-pim-bootstrap-information ja get-pim-neighbors-information.



```
"nat": {
    "source": {
        "pool": [
            {
                "name": "p1",
                "port": {
                    "no-translation": [
                        null
                    ]
                },
                "address": [
                    {
                        "name": "10.100.11.250/32"
                    }
                ]
            }
        ],
        "rule-set": [
            {
                "name": "rs1",
                "to": {
                    "zone": [
                        "untrust"
                    ]
                },
                "from": {
                    "zone": [
                        "trust"
                    ]
                },
                "rule": [
                    {
                        "name": "r1",
                        "src-nat-rule-match": {
                            "source-address": [
                                "169.254.1.2/32"
                            ]
                        },
                        "then": {
                            "source-nat": {
                                "pool": {
                                    "pool-name": "p1"
                                }
                            }
                        }
                    }
                ]
            }
        ]
    }
},
```

Kuva 11. *Linux-palvelimen osoitteen piilotus osoitteenmuunnoksella.*

Käyttäjälle annetaan käyttöliittymässä tieto, että OSPF- ja PIM-naapuruudet sekä BSR-reitittimen tiedot on löydetty. Lisäksi Linux-palvelin tekee virtuaalireitittimen läpi saavutettavuustestin kaikille esiasetetuille laitteille, joiden odotetaan löytyvän palvelusta ja kerrotaan, saavutettiinko niitä vai ei. Kuvassa 11 nähdään, kuinka Linux-palvelimen osoite piilotetaan runkoverkolta osoitteenmuutoksella. Kuvassa osoitteenmuutossäännöstössä viitattu luotettava alue on Linux-palvelimen aliverkko ja epäluotettava alue on WAN-verkko. Linux-palvelin hakee lisäksi vielä NTP-palvelimen Javan NTP server-client -kirjastolla. Sama testaus suoritetaan kaikille käyttäjän asetettamille palveluille.

6. YHTEENVETO

Tässä työssä esiteltiin verkonhallintaprotokollia, verkkojen testauksen menetelmiä ja laitteistoa sekä toteutus verkkotestauslaitteesta. Verkonhallintaprotokollista esiteltiin NETCONF, WMI, SNMP ja kontrolliprotokolla ICMP. Näistä protokollista erityisesti NETCONF ja SNMP tarjoavat tehokkaita työkaluja laajojen verkkojen hallintaan. WMI ei yllä samalle käyttöasteelle näiden kahden kanssa, vaikka se tarjoaakin hyvin samanlaisia työkaluja kuin NETCONF, sillä WMI:n käyttö rajoittuu Windows-ympäristöihin.

Testauksen menetelmistä tarkasteltiin yksikerroksista ja monikerroksista lähestymistapaa. Monikerroksisen lähestymistavan todettiin sopivan paremmin järjestelmätestaukseen, sillä OSI-mallin kerrosten yhteistoimintaa ei pysty arvioimaan vain yhden kerroksen toiminnan perusteella. Lisäksi todettiin, että verkkotestausta kannattaa tehdä alemmista kerroksista ylöspäin suuntautuen, sillä alemmissa kerroksissa tapahtuvat virheet voi olla hyvin hankala paikallistaa oikein ylemmillä kerroksilla.

Testiverkkojen rakentamisen helpottamiseksi esitettiin virtualisointiteknologioita, joilla voidaan edullisemmin rakentaa laajoja verkkoja vastaavia testiympäristöjä. Näiden lisäksi esiteltiin myös muutamia yleisesti verkkotestauksessa käytettäviä laitteita.

Työssä esiteltiin myös mobiilin verkkotestauslaitteen toteutus, jossa hyödynnettiin virtualisointiteknologioita ja NETCONF-protokollaa. Verkkotesteri kykenee todentamaan siihen kytketyn verkon OSPF-, PIM- ja NTP-protokollien toimivuuden, sekä todentamaan esiasetettujen IP-osoitteiden saavutettavuuden kyseisessä verkossa. Verkkotesterin toiminta on vielä aika suppeaa, sillä se ei tue kovin montaa protokollaa, eikä se esimerkiksi osaa antaa yksityiskohtaista tietoa virhetilanteissa. Verkkotesterin toimintaa voisikin tulevaisuudessa kehittää tukemaan useampia protokollia, kuten esimerkiksi Border Gateway Protocol (BGP) –reititysprotokollaa. Lisäksi verkkotesteriä voisi kehittää antamaan yksityiskohtaisempia virheilmoituksia, jotta käyttäjän olisi helpompi paikallistaa verkon virheet.

LÄHTEET

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, & V. Sekar. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service, ACM SIGCOMM Computer Communication Review, volume 42, pp. 13–24, 2012. DOI: 10.1145/2342356.2342359

[2] A. Dominguez, Understanding Network Management Protocols, 2019. Viitattu 5.11.2019. Saatavissa: https://pandorafms.com/blog/network-management-protocols

[3] J. Postel. RFC 792: Internet Control Message Protocol, 1981. Viitattu 24.11.2019. Saatavissa: https://tools.ietf.org/html/rfc792

[4] Douglas R. Mauro & Kevin J. Schmidt. Essential SNMP (1st ed.), Sebastopol, CA: O'Reilly & Associates, 2001.

[5] W. Stallings, SNMPv3: A security enhancement for SNMP, IEEE Communications Surveys, Volume 1, Issue 1, 1998, DOI: 10.1109/COMST.1998.5340405

[6] D. Harrington, R. Presuhn & B. Wijnen, RFC 2271: An Architecture for Describing SNMP Management Frameworks, IETF, 1998. Viitattu 24.1.2020. Saatavissa https://tools.ietf.org/html/rfc2271

[7] M. Rose & K. McCloghrie, RFC 1155: Structure and Identification of Management Information for TCP/IP-based Internets, IETF, 1990. Saatavissa: https://tools.ietf.org/html/rfc1155

[8] R. Presuhn, J. Case, M. Rose & S Waldbusser, RFC 3416: Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP), IETF, 2002. Viitattu 29.1.2020. Saatavissa: https://tools.ietf.org/html/rfc3416

[9] J. Schoenwalder, RFC 3535: Overview of the 2002 IAB Network Management Workshop, IETF, 2003. Viitattu 3.2.2020. Saatavissa: https://tools.ietf.org/html/rfc3535

[10] The Comprehensive Guide to WMI, LogicMonitor, 2015. Viitattu 26.2.2020. Saatavissa: https://logicmonitor.com/wp-content/uploads/2015/07/Comprehensive-Guide-to-WMI-1.pdf

[11] M. Jacobs, M. Satran & D. Coulter, WMI Architecture, Microsoft, 2018. Viitattu 26.2.2020. Saatavissa: https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-architecture

[12] M. Jacobs, M. Satran, D. Coulter & D. Batchelor WMI Classes, Microsoft, 2018. Viitattu 28.2.2020. Saatavissa: https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-classes

[13] M. Jacobs, M. Satran & D. Coulter, WMI Infrastructure, Microsoft, 2018. Viitattu 2.3.2020. Saatavissa: https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-infrastructure

[14] J. Schönwälder, M. Björklund & P. Shafer, Network configuration management using NETCONF and YANG, IEEE Communications Magazine, vol. 48, issue 9, pp. 166–173, 2010. DOI: 10.1109/MCOM.2010.5560601

[15] Tail-f Systems, NETCONF Tutorial, 2015. Saatavissa: https://youtu.be/N4vov1mI14U

[16] R. Enns, M. Bjorklund, J Schoenwaelder & A. Bierman, RFC 6241: Network Configuration Protocol, IETF, 2011. Viitattu 14.11.2019. Saatavissa: https://tools.ietf.org/html/rfc6241

[17] M. Bjorklund, RFC 6020: YANG – A Data Modeling Language for the Network Configuration Protocol, IETF, 2010. Viitattu 22.11.2019. Saatavissa: https://tools.ietf.org/html/rfc6020

[18] B. Nelson, Remote procedure call, luku 8, 1981, Viitattu 14.11.2019. DOI: 10.1007/1-4020-8086-7_8

[19] R. Enns, RFC 4741: NETCONF Configuration Protocol, IETF, 2006. Viitattu 5.2.2020. Saatavissa: https://tools.ietf.org/html/rfc4741

[20] L. Angrisani & C. Narduzzi, Testing Communication and Computer Networks: an overview, IEEE Instrumentation and Measurement Magazine, vol. 11, issue 5, pp. 12–24, 2008. Viitattu 14.4.2020. DOI: 10.1109/MIM.2008.4630738

[21] S. Kalidindi, N. Hyunh, B. Eklow, J. Goldstein, “Real life” System Testing of Network Equipment, 2004 International Conference on Test, pp. 1072–1077, 26–28 Oct. 2004, Charlotte, NC, USA, IEEE, 2004. Viitattu 21.4.2020. DOI: 10.1109/TEST.2004.1387380

[22] S. Manishankar, S. Harshitha, A. Mukhopadhyay, A. Anoop, Technologies for Network Testing: A Hybrid Approach, 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), pp. 946–951, 27–29 March 2019, Erode, India, IEEE, 2019. Viitattu 14.4.2020. DOI: 10.1109/ICCMC.2019.8819648

LIITE: LIIKENNEPALVELUN KONFIGURAATIO


```
"junos-es-conf-root:configuration": {
    "version": "20190606.224121_builder.r1033375",
    "junos-es-conf-interfaces:interfaces": {
        "interface": [
            {
                "name": "fxp0",
                "unit": [
                    {
                        "name": "0"
                    }
                ]
            },
            {
                "name": "ge-0/0/1",
                "unit": [
                    {
                        "name": "0",
                        "family": {
                            "inet": {
                                "address": [
                                    {
                                        "name": "169.254.1.1/29"
                                    }
                                ]
                            }
                        }
                    }
                ]
            },
            {
                "name": "ge-0/0/0",
                "unit": [
                    {
                        "name": "1306",
                        "vlan-id": "1306",
                        "family": {
                            "inet": {
                                "unnumbered-address": {
                                    "source": "lo0.31"
                                }
```

```
                        }
                    }
                }
            ],
            "vlan-tagging": [
                null
            ]
        },
        {
            "name": "lo0",
            "unit": [
                {
                    "name": "31",
                    "family": {
                        "inet": {
                            "address": [
                                {
                                    "name": "10.100.11.250/32"
                                }
                            ]
                        }
                    }
                }
            ]
        }
    ]
},
"junos-es-conf-system:system": {
    "root-authentication": {
        "encrypted-password": "$6$hQRQbwX0$xFWX7XLjchATumkzCali0j56m1SHvQFhhAnD-NkqR6QpoOMMjk4/uOENMHDf8/qwbryNI0P7leY5CMbKGCjI/1/"
    },
    "login": {
        "user": [
            {
                "name": "admin",
                "authentication": {
                    "encrypted-password": "$6$WzPbh8y5$EwVBsKnTCqypAEAyvtyCG7uApMfITXi9hJbgrM-rpsSCY/NHto07XXOrbsI0jJ1mBmuIKOPzAi7qZv6l.wC2fD."
                },
                "class": "super-user",
                "uid": 2000
            }
```

```
    ]
},
"syslog": {
    "file": [
        {
            "name": "messages",
            "contents": [
                {
                    "name": "any",
                    "any": [
                        null
                    ]
                },
                {
                    "name": "authorization",
                    "info": [
                        null
                    ]
                }
            ]
        },
        {
            "name": "interactive-commands",
            "contents": [
                {
                    "name": "interactive-commands",
                    "any": [
                        null
                    ]
                }
            ]
        }
    ],
    "user": [
        {
            "name": "*",
            "contents": [
                {
                    "name": "any",
                    "emergency": [
                        null
                    ]
                }
```

```
            ]
        }
    ]
},
"services": {
    "web-management": {
        "http": {
            "interface": [
                "fxp0.0",
                "ge-0/0/1.0"
            ]
        }
    },
    "netconf": {
        "yang-compliant": [
            null
        ],
        "traceoptions": {
            "file": {
                "files": 20,
                "filename": "netconf-ops.log",
                "size": "3m",
                "world-readable": [
                    null
                ]
            },
            "flag": [
                {
                    "name": "all"
                }
            ]
        },
        "ssh": {
            "connection-limit": 5,
            "port": 830,
            "rate-limit": 150
        },
        "rfc-compliant": [
            null
        ]
    },
    "ssh": {
        "protocol-version": [
```

```
            "v2"
          ],
          "hostkey-algorithm": {
            "no-ssh-dss": [
              null
            ],
            "no-ssh-ecdsa": [
              null
            ],
            "no-ssh-ed25519": [
              null
            ],
            "ssh-rsa": {}
          }
        }
      }
    },
    "junos-es-conf-routing-instances:routing-instances": {
      "instance": [
        {
          "name": "traffic-wan",
          "protocols": {
            "ospf": {
              "area": [
                {
                  "name": "0.0.0.0",
                  "interface": [
                    {
                      "name": "lo0.31",
                      "passive": {}
                    },
                    {
                      "name": "ge-0/0/0.1306",
                      "authentication": {
                        "md5": [
                          {
                            "name": 1,
                            "key": "$9$1qRIclKvL7NbevWxN-wsP5T3Ct"
                          }
                        ]
                      },
                      "priority": 1,
                      "dead-interval": 40,
```

```
                        "transit-delay": 1,
                        "retransmit-interval": 5,
                        "metric": 10,
                        "interface-type": "p2p",
                        "hello-interval": 10
                    }
                ]
            }
        ]
    },
    "pim": {
        "interface": [
            {
                "name": "ge-0/0/0.1306",
                "mode": "sparse"
            }
        ]
    }
},
"interface": [
    {
        "name": "ge-0/0/1.0"
    },
    {
        "name": "ge-0/0/0.1306"
    },
    {
        "name": "lo0.31"
    }
],
"instance-type": "virtual-router",
"routing-options": {
    "router-id": "10.100.11.250"
}
}
]
},
"junos-es-conf-security:security": {
"zones": {
    "security-zone": [
        {
            "name": "trust",
            "tcp-rst": [
```

```
            null
        ],
        "interfaces": [
            {
                "name": "ge-0/0/1.0",
                "host-inbound-traffic": {
                    "protocols": [
                        {
                            "name": "all"
                        }
                    ],
                    "system-services": [
                        {
                            "name": "all"
                        }
                    ]
                }
            }
        ]
    },
    {
        "name": "untrust",
        "screen": "untrust-screen",
        "interfaces": [
            {
                "name": "ge-0/0/0.1306",
                "host-inbound-traffic": {
                    "protocols": [
                        {
                            "name": "all"
                        }
                    ],
                    "system-services": [
                        {
                            "name": "ping"
                        }
                    ]
                }
            },
            {
                "name": "lo0.31",
                "host-inbound-traffic": {
                    "protocols": [
```

```
                    {
                        "name": "all"
                    }
                ],
                "system-services": [
                    {
                        "name": "ping"
                    }
                ]
            }
        }
    ]
  }
]
},
"log": {
  "mode": "stream",
  "report": {}
},
"nat": {
  "source": {
    "pool": [
      {
        "name": "p1",
        "port": {
          "no-translation": [
            null
          ]
        },
        "address": [
          {
            "name": "10.100.11.250/32"
          }
        ]
      }
    ],
    "rule-set": [
      {
        "name": "rs1",
        "to": {
          "zone": [
            "untrust"
          ]
```

```
            },
            "from": {
                "zone": [
                    "trust"
                ]
            },
            "rule": [
                {
                    "name": "r1",
                    "src-nat-rule-match": {
                        "source-address": [
                            "169.254.1.2/32"
                        ]
                    },
                    "then": {
                        "source-nat": {
                            "pool": {
                                "pool-name": "p1"
                            }
                        }
                    }
                }
            ]
        }
    ]
  }
},
"policies": {
  "policy": [
    {
      "from-zone-name": "trust",
      "to-zone-name": "trust",
      "policy": [
        {
          "name": "default-permit",
          "match": {
            "destination-address": [
              "any"
            ],
            "source-address": [
              "any"
            ],
            "application": [
```

```
                    "any"
                ]
            },
            "then": {
                "permit": {}
            }
        }
    ]
},
{
    "from-zone-name": "trust",
    "to-zone-name": "untrust",
    "policy": [
        {
            "name": "default-permit",
            "match": {
                "destination-address": [
                    "any"
                ],
                "source-address": [
                    "any"
                ],
                "application": [
                    "any"
                ]
            },
            "then": {
                "permit": {}
            }
        }
    ]
}
]
},
"screen": {
    "ids-option": [
        {
            "name": "untrust-screen",
            "icmp": {
                "ping-death": [
                    null
                ]
            },
```

```
                "ip": {
                    "source-route-option": [
                        null
                    ],
                    "tear-drop": [
                        null
                    ]
                },
                "tcp": {
                    "land": [
                        null
                    ],
                    "syn-flood": {
                        "source-threshold": 1024,
                        "timeout": 20,
                        "queue-size": 2000,
                        "destination-threshold": 2048,
                        "attack-threshold": 200,
                        "alarm-threshold": 1024
                    }
                }
            }
        ]
    }
  }
}
```