

SDN Test Bench

Empiirinen tutkimus SDN-ohjainten järjestelmätestauksesta

Antti Lohtaja

Opinnäytetyö
Huhtikuu 2020
Tekniikan ala
Insinööri (AMK), tieto- ja viestintätekniikka

Tekijä(t) Lohtaja, Antti	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Huhtikuu 2020
	Sivumäärä 43	Julkaisun kieli Suomi
		Verkojulkaisulupa myönnetty: x
Työn nimi SDN Test Bench Empiirinen tutkimus SDN-ohjainten järjestelmätestauksesta		
Tutkinto-ohjelma Tieto- ja viestintätekniikan tutkinto-ohjelma		
Työn ohjaaja(t) Saharinen, Karo; Kotikoski, Sampo		
Toimeksiantaja(t) Alatalo, Janne; JAMK		
Tiivistelmä SDN eli Software-Defined Networking on teknologia tietoverkkojen käyttäytymisen hallintaan ohjelmallisesti. SDN-teknologian keskellä on SDN-ohjain -ohjelmisto, joka keskitetysti ohjaa siihen liitetyn tietoverkon liikennettä ja käyttäytymistä. SDN perustuu tietoverkossa olevien laitteiden kanssa kommunikoimiseen yhteisellä protokollalla. Jos SDN-ohjain ei toimi, myöskään kyseinen tietoverkko ei toimi. Toimeksiannossa haluttiin tutkia millä tavoilla SDN-ohjainten järjestelmätestausta voidaan toteuttaa SDN-ohjaimen toimintavarmuuden lisäämiseksi. Tutkimuksen pohjalta toteutettiin sovellus, joka testaa SDN-ohjaimia keskustelemalla niiden kanssa OpenFlow -nimisellä SDN-protokollalla. Järjestelmätestausta toteutettiin vertailemalla SDN-ohjaimien lähettämiä viestejä OpenFlow-protokollan virallisen määrittelyn malleihin. Lopputuloksena kommunikointi OpenFlow-protokollalla onnistui kahden testatun SDN-ohjaimen kanssa. Sovelluksella ehdittiin toteuttaa kolme testitilannetta, joista kahdella onnistuttiin varmentamaan SDN-ohjaimen oikea toiminta. Kolmatta testitilannetta ei saatu toimimaan odotetulla tavalla testitulosten perusteella. Tutkimuksen ja tulosten perusteella saatiin tietoa kyseisen järjestelmätestauksen lähestymistavan soveltuvuudesta sekä mahdollisista lisähyödyistä.		
Avainsanat (asiasanat) tietoverkot, tietoliikenne, SDN, Software-Defined Networking, OpenFlow, testaus, järjestelmätestaus, ohjelmiston suunnittelu		
Muut tiedot (Salassa pidettävät liitteet)		

Author(s) Lohtaja, Antti	Type of publication Bachelor's thesis	Date March 2020
		Language of publication: Finnish
	Number of pages 43	Permission for web publication: x
Title of publication SDN Test Bench An empirical research about system testing SDN controllers		
Degree programme Information and communications technology		
Supervisor(s) Saharinen, Karo; Kotikoski, Sampo		
Assigned by Alatalo, Janne; JAMK		
<p>Abstract</p> <p>SDN (Software-Defined Networking) is a technology for programmatical controlling of the behavior of a telecommunications network. At the center of SDN technology is SDN controller software controlling network behavior and traffic in a centralized manner. SDN relies on communication between network devices via a common protocol. If an SDN controller does not function, then the whole network it is responsible for cannot function. The aim of the assignment was to research ways for conducting system testing on an SDN controller in order to increase their reliability.</p> <p>Based on the research, an application was created capable of testing SDN controllers by communicating to them via a protocol called OpenFlow. The system testing was conducted by comparing the messages sent by SDN controllers to the official specification of the OpenFlow protocol.</p> <p>As a result, the communication using OpenFlow protocol was successful with two tested SDN controllers. Three test situations were implemented, from which two successfully verified the correct behavior of an SDN controller. Based on the test results, the third test situation did not work correctly.</p> <p>The research and the test results resulted in knowledge on an approach to SDN controller system testing and its applicability.</p>		
Keywords/tags (subjects) Networks, SDN, Software-Defined Networking OpenFlow, Testing, Software Design		
Miscellaneous (Confidential information)		

Sisältö

SANASTO	4
1 Johdanto	5
1.1 Taustaa	5
1.2 SDN-teknologian haasteet.....	6
1.3 Toimeksiantajan tarve	7
1.4 Tutkimusmenetelmä ja aineisto	7
1.5 Test Bench -ratkaisu	7
2 Teoria ratkaisun taustalla	8
2.1 Yleistä	8
2.2 OpenFlow-protokolla	9
2.3 SDN-ohjaimen ja OpenFlow-kytkimen välinen yhteys	10
2.3.1 Hello-tilanne	10
2.3.2 Handshake-tilanne.....	11
2.3.3 Error-tilanne	12
3 Kehitetty SDN-ohjainten Test Bench testaussovellus	13
3.1 Test Bench ohjelmiston suunnittelu.....	13
3.2 C++:n yksityiskohtia toteutuksessa	16
3.2.1 Moderni C++	16
3.2.2 Tiedonsiirto C++:lla	16
3.2.3 Boost.Asio	17
3.3 OpenFlow'n tietorakenteet.....	21
3.4 Test Benchin rakenne	22
3.4.1 Test Suite	22
3.4.2 Hello-tilanteen testaus	23
3.4.3 Handshake-tilanteen testaus.....	27
3.4.4 Error-tilanteen testaus	30
3.4.5 Mahdollinen rajaamisen ongelma.....	32
3.4.6 Testauksen automatisointi	33
3.4.7 Testitulosten raportointi	34

	2
4 Testit ja testitulokset	34
4.1 Faucet SDN -ohjain	34
4.2 Puikkari SDN -ohjain	36
5 Tulosten pohdinta	37
5.1 Tulosten luotettavuus	37
5.2 Saavutetut tavoitteet	37
5.3 Jatkokehityskohteet	37
Lähteet	39
Liitteet	41
Liite 1. Esimerkki SDN-ohjatusta tietoverkosta	41
Liite 2. Test Benchin käyttöönotto ja tulosten toistaminen	42

Kuviot

Kuvio 1. Hello-tilanteen sekvenssikaavio	11
Kuvio 2. Handshake-tilanteen sekvenssikaavio	12
Kuvio 3. Vaatimusmäärittelyn kaksi ensimmäistä käyttäjävaatimusta	13
Kuvio 4. Test Benchin vuorovaikutussuhteet	14
Kuvio 5. Vaatimusmäärittelyn kolmas käyttäjävaatimus	14
Kuvio 6. OpenFlow-vaatimukset vaatimusmäärittelyssä	15
Kuvio 7. Boost.Asio-kirjaston toiminta (Kohlhoff, C. N.d., muokattu)	18
Kuvio 8. Yhteyden muodostaminen Boost.Asiolla.....	19
Kuvio 9. Lukutehtävän luonti Boost.Asion tehtäväjonoon	20
Kuvio 10. Vastaanotetun datan käsittely callback funktiossa	20
Kuvio 11. OpenFlow-viestin ylätunnisteen rakenne.....	21
Kuvio 12. Hello-viestin rakenne	23
Kuvio 13. OpenFlow-viestin ylätunnisteen sarjallistaminen	24
Kuvio 14. Esimerkki merkkijonon bittien leikaantumisesta, kun siitä otetaan yksi tavu	25

Kuvio 15. Esimerkki bittien ylivuotamisesta, kun bittejä siirretään 8 kertaa oikealle.....	25
Kuvio 16. Viestin pituuden parsiminen ylätunnisteeseen	26
Kuvio 17. OpenFlow-protokollan version tarkistus Handshake-tilanteessa	28
Kuvio 18. OpenFlow-viestin tyyppin tarkistus Handshake-tilanteessa	28
Kuvio 19. FeatureRes-viestin rakenne	29
Kuvio 20. FeatureRes-viestin rakentaminen.....	30
Kuvio 21. Error-viestin rakenne	31
Kuvio 22. Virheellisen EchoRequest-viestin muodostaminen.....	31
Kuvio 23. Ylätunnisteen tarkistaminen Error-viestistä	32
Kuvio 24. Esimerkki mahdollisista saapuvista OpenFlow-viesteistä yhteyden aikana.....	33
Kuvio 25. Tuloste Faucet SDN-ohjaimen testauksesta	35
Kuvio 26. Tuloste Puikkari SDN-ohjaimen testauksesta	36

Taulukot

Taulukko 1. Esimerkki tavujärjestyksistä	17
--	----

SANASTO

C++	Ohjelmointikieli
LSB	Least Significant Bit, vähiten merkitsevä eli bittijonon oikeanpuolimmais bitti
MSB	Most Significant Bit, eniten merkitsevä eli bittijonon vasemmanpuolimmais bitti
NFV	Network Functions Virtualization, termi virtuaalisille verkkotoiminnoille kuten esim. virtuaalinen palomuri
ONF	Open Networking Foundation, säätiö joka kehittää OpenFlow-protokollaa
OpenFlow	SDN-tekniologian protokolla verkkolaitteiden ohjaamiseen.
SDN	Software-Defined Networking
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1 Johdanto

1.1 Taustaa

Nykypäivän jatkuva multimediasisällön, mobiililaitteiden käytön sekä pilvipalveluiden kasvu tuovat palveluntarjoajille ja yrityksille uusia haasteita varsinkin kustannuksien kannalta. Perinteiset liiketoimintamallit alkavat heikentyä kun kustannukset kasvavat, mutta asiakkailta saatavat tulot pysyvät ennallaan. Siksi yhä useammat tahot ovat tutkimassa SDN-tekniikan mahdollisuuksia. (What is SDN? n.d.)

SDN eli Software-Defined Networking on mielenkiintoinen tekniikka, jolla voidaan hallita tietoverkon käyttäytymistä ohjelmistosovelluksen kautta älykkäästi ja keskitetysti. Tietoverkon käyttäytymisellä tarkoitetaan mm. tietoliikenteen ohjausta sekä verkon monitorointia ja analysointia. (Mt.)

Tietoverkon käyttäytymisen älykkäällä ohjauksella voidaan mm. jakaa tietoverkon kuormitusta tasaisesti, mikä vähentää kustannuksia ja voi nopeuttaa tiedonsiirtonopeuksia asiakkaille, jolloin taas mm. asiakastyytyväisyys lisääntyy. Yritykset voivat vaikka tarpeen mukaan taata palvelulle mahdollisimman pienen latenssin eli viiveen tai maksimaalisen kaistanleveyden. Palveluntarjoajat eivät valitettavasti saa tehdä tällaista suosimista kuluttajille, mutta esimerkiksi yksi eräältä yritykseltä kuulemistani käyttötapauksista on leikkausrobotin reaaliaikainen ohjaus etänä sairaalaympäristössä, jolloin viiveen täytyy olla minimaalinen (Laitila 2017).

Toinen SDN:n iso alue on Network Functions Virtualization (NFV) eli verkkotoimintojen virtualisointi. Sillä tarkoitetaan tietoverkon erilaisten komponenttien kuten reitittimien tai palomuurien virtualisointia. (Liimatainen 2017, 48.)

Näitä komponentteja voidaan nopeasti ajaa ylös pilveen ja liittää sitten osaksi SDN-ohjattua verkkoa. Esimerkiksi asiakkaat voisivat tilata ohjelmistosovelluksen kautta itselleen virtuaalisen palomuurin pilvestä, jonka jälkeen asiakkaan liikenne voidaan ensin aina ohjata palomuurin kautta ennen muualle internettiin pääsyä. Lisäksi

SDN:llä voidaan monitoroida tietoverkkoa mm. tietoturvan kannalta tai analysoida sen käyttöä ja pienentää kustannuksia. SDN säästää myös yrityksien kustannuksia olemalla yksi yhteinen alusta verkkojen hallitsemiselle, riippumatta sen alla olevasta tekniikasta. (Rash 2018.)

1.2 SDN-teknologian haasteet

Normaalisti kytkimet ja reitittimet tekevät verkkoliikenteen ohjauksen kokonaan itse. Siihen kuuluu optimaalisen reitin laskeminen ja pakettien välittäminen eteenpäin. Yksittäiset verkkolaitteet on tehty hyvin spesifisiin työtehtäviin, ja siksi niiden hallinta on hankalaa ja monimutkaista. Tämä on se asia, joka SDN-teknologialla halutaan ratkaista. Nämä kaksi asiaa (reititys ja välitys) halutaan erottaa yksittäisiltä laitteilta niin, että niitä voidaan hallita yhden yhteisen alustan (SDN) kautta. Esimerkiksi uusien reititysalgoritmien kokeileminen käytännössä on vaikeaa, koska konfiguroitavia laitteita voi olla paljon ja jokainen laite pitää konfiguroida laitekohtaisesti erikseen.

SDN-teknologiaa on paljon kritisoitu mm. sen toimintavarmuudesta ja turvallisuudesta. Keskeinen ongelma SDN-teknologiassa on se, että koko verkko on riippuvainen SDN-ohjaimesta. Verkolla voi olla jokin varasuunnitelma SDN-ohjaimen toiminnan pettäessä, mutta SDN-ohjaimen pettäminen saattaa pahimmillaan aiheuttaa koko verkon kaatumisen. Lisäksi SDN:llä on ongelmia tietoturvassa, sillä se on uusi teknologia ja kaikkia haavoittuvuuksia ei voida vielä tietää. Jos ulkopuolinen tunkeutuja saa SDN-ohjaimen haluunsa, sillä voi olla todella mittavat seuraukset. SDN-ohjaimella on kuva koko verkosta ja se näkee, millaista liikennettä verkossa tapahtuu. (Zurkus 2017.)

Toimintavarmuus on sanomattakin äärimmäisen tärkeää tietoliikenneverkoissa. SDN-ohjain on SDN-ohjatun tietoverkon kriittisin osa. Ilman sitä, koko tietoverkko ei voi toimia. Siksi sen toiminta täytyy testata mahdollisimman hyvin.

1.3 Toimeksiantajan tarve

Toimeksiantajan tavoite opinnäytetyölle oli lähinnä tutkia ja saada tietoa siitä miten ja millä menetelmillä SDN-tekniikan SDN-ohjain sovelluksia voitaisiin lähteä testaamaan. Tutkimuksen pohjalta oli tarkoitus luoda Test Bench niminen Proof-of-Concept testaussovellus. Testaamisen tarve ja tavoite on SDN-ohjainten laadun nostaminen. Opinnäytetyön tuloksia voisi myöhemmin käyttää pohjana testaussovelluksen jatkokehittämiselle.

1.4 Tutkimusmenetelmä ja aineisto

Opinnäytetyön aihe Test Bench on kehitystehtävä, jossa tutkimus on kokemuksellista eli empiiristä. Opinnäytetyössä tarkasteltiin ja analysoitiin sitä, miten testauskohteena oleva SDN-ohjain reagoi sille lähetettyihin viesteihin. Näistä havainnoista voidaan vetää erilaisia johtopäätöksiä sekä kehitystyöstä että testauskohteesta.

Kehitystyön kannalta tärkeimmät tutkimusaineistot olivat Internetistä löytyvät valmiit dokumentit. Jokaiselle opinnäytetyön tutkimusaiheelle on mahdollista löytää jokin tutkimusta tukeva asiakirja tai dokumentti.

1.5 Test Bench -ratkaisu

Test Bench -sovellus on tarkoitettu SDN-ohjainten järjestelmätestaukseen. Järjestelmätestauksella testataan ohjelmiston toimintaa kokonaisuudessaan. SDN-ohjaimet toimivat Test Benchille ns. mustina laatikoina eli se ei tiedä miten sovellus toimii sisäisesti, eikä sen tarvitsekaan tietää.

Test Benchin tarkoituksena on luoda erilaisia tilanteita SDN-ohjaimelle ja tutkia sitten miten SDN-ohjain reagoi ja vastaa pyyntöihin. SDN-ohjainten ja verkkolaitteiden täytyy ymmärtää toisiaan laitteesta riippumatta, joten kommunikoimiseen tarvitaan yhteinen protokolla. SDN-tekniikan suosituin ja eniten käytössä oleva protokolla on

nimeltään OpenFlow. OpenFlow’ia voidaan käyttää sekä verkon käyttäytymisen konfigurointiin että verkon monitorointiin.

OpenFlow-protokollaan on määritelty, millä tavalla protokollaa käyttävä yhteys alustetaan, mitä protokollan viestit sisältävät ja miten tiettyihin viesteihin kuuluu vastata. Test Bench rakentaa tilanteeseen liittyvät OpenFlow-viestit ja odottaa SDN-ohjaimen vastaavan OpenFlow-protokollan spesifikaation määrittämällä tavalla. Jos SDN-ohjain ei vastaa Test Benchille odotetulla tavalla testi merkitään epäonnistuneeksi. Toisin sanoen Test Bench varmistaa että SDN-ohjain noudattaa OpenFlow’n spesifikaatiota.

Test Benchissä keskitytään OpenFlow-protokollan versioon 1.3., koska opinnäytetyön tekijän kokemuksen mukaan se on tällä hetkellä vakain ja laajimmin käytössä oleva versio protokollasta. Koska edes yhden OpenFlow-protokollan version koko testaaminen on liian laaja kokonaisuus toimeksiannolle, rajoitettiin Test Bench kattamaan vain neljä erilaista OpenFlow’n viestiä.

2 Teoria ratkaisun taustalla

2.1 Yleistä

SDN-tekniikan keskellä on sen aivoiksi kutsuttu SDN-ohjainohjelmistosovellus. Ohjaimen päätarkoitus on hallita verkossa olevien SDN-tekniikkaa tukevien verkkolaitteiden käyttäytymistä. Nämä laitteet voivat olla mitä tahansa kuten esimerkiksi reititimiä, kytkimiä tai em. virtuaalisia verkkotoimintoja.

SDN-ohjaimella on kokonaiskuva tietoverkon tilanteesta. SDN-ohjain voi seurata mm. verkon kuormitusta tai mahdollisia linjojen katkoksia. Näiden tietojen perusteella SDN-ohjain pystyy määrittämään verkon parhaan mahdollisen käyttäytymisen. Esimerkiksi jos tietoverkon yhdellä reitillä on paljon kuormaa, SDN-ohjain pystyy tasoittamaan tilanteen ohjaamalla liikennettä muita reittejä pitkin määränpäähän.

2.2 OpenFlow-protokolla

OpenFlow on tietoliikenneprotokolla, jolla SDN-ohjain keskustelee sitä tukevien verkkokyttekinten kanssa. Sen perustarkoituksena on hallita verkon käyttäytymistä, mutta sillä voidaan myös tehdä verkon analysointia.

Opinnäytetyön tärkein aineisto on OpenFlow-protokollan version 1.3. virallinen määrittely. Se löytyy Internetistä julkisena PDF dokumenttina (OpenFlow Switch Specification 2012). Siinä kerrotaan millaisia viestejä OpenFlow-protokollassa on ja miten niitä kuuluu käyttää. Test Bench sovelluksen testitilanteet perustuvat ja ovat riippuvaisia juurikin tästä aineistosta.

Protokollan on kehittänyt voittoa tavoittelematon Open Networking Foundation (ONF) -säätiö. Sitä tukevat mm. Google, Intel sekä Microsoft (Our Members, n.d.). Säätiön tavoitteena on SDN-tekniikan edistäminen monenlaisten projektien kautta (Our Mission, n.d.).

OpenFlow'n terminologiassa ohjattavia verkkolaitteita kutsutaan yleisesti OpenFlow-kytkimiksi (OpenFlow Switch), ovat ne sitten varsinaisia kytkimiä tai reitittimiä.

OpenFlow-kytkin käsitteenä on vain jokin abstrakti verkkopaketteja käsittelevä kone tai laite. OpenFlow-kytkimen käyttäytyminen perustuu pitkälti sen sisällä oleviin vuotauluihin (Flow Table).

Kun OpenFlow-kytkimelle saapuu verkkopaketti, sitä verrataan vuotaulun sääntöihin. Vuotaulu on vähän kuin perinteinen reititystaulu, mutta se sisältää enemmän kenttiä. Vuotaulussa olevia tietueita taas sanotaan säännöiksi (Flow Table Rule). Vuotaulun kenttiä voivat olla mm. IP-osoite, MAC-osoite, VLAN ID eli virtuaalisen lähiverkon tunnus ja prioriteetti. VLAN ID:llä paketit voidaan yhdistää johonkin virtuaaliseen lähiverkkoon ja prioriteetillä voidaan priorisoida vuotaulun sääntöjä. Näiden kenttien lisäksi jokaisella vuotaulun säännöllä on käsky (Action). Käskyillä määrätään, mitä sääntöön osuvalle paketille tehdään. Käskyillä voidaan esimerkiksi lähettää paketti ulos jostakin portista, muokata paketin kenttiä, pudottaa se kokonaan tai ohjata se vaikka toiselle vuotaululle. Tällaisella vuotaulujen putkittamisella verkkopaketteja

voidaan käsitellä tarkemmin ja liikenteestä saadaan tarkempia tietoja vuotaulujen statistiikan kautta. (OpenFlow Switch Specification 2012, 12-13.)

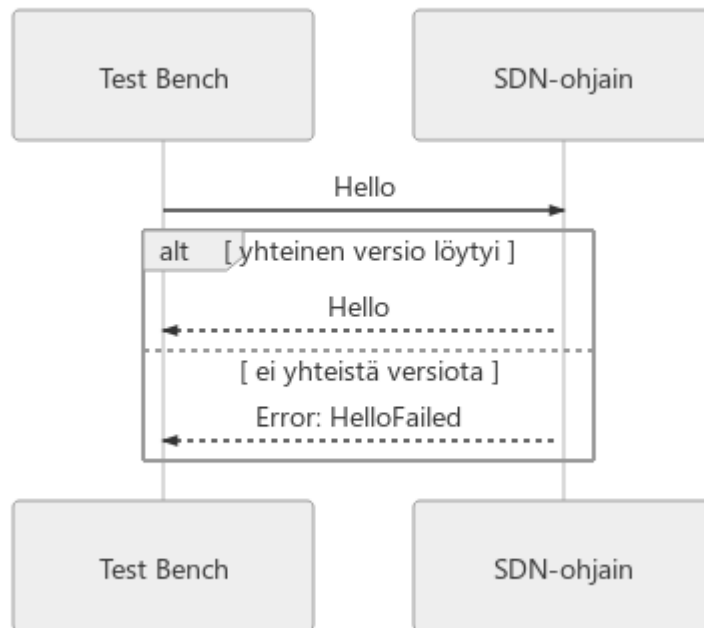
Kun kytkimelle saapunut paketti ei sovi yhteenkään vuotaulun sääntöön, eli kytkin ei tiedä mitä paketille tehdään, oletuksena kytkin pudottaa paketin kokonaan (OpenFlow Switch Specification 2012, 13). Yleensä näissä tapauksissa kuitenkin halutaan, että kytkin lähettää paketista kopion SDN-ohjaimelle tarkasteltavaksi. Tällöin SDN-ohjain voi päättää mitä paketille pitäisi tehdä. Jos paketti halutaan saada määränpäähän, SDN-ohjain voi laskelmoida sille optimaalisen reitin ja konfiguroida reitin varrella olevat kytkimet välittämään kyseisen paketin oikeisiin portteihin. Tai voi olla että pakettia ei haluta välittää eteenpäin ollenkaan. Tällä tavalla SDN ottaa reitityksen vastuun pois verkkolaitteilta, jolloin verkon hallinta yksinkertaistuu ja kustannukset alenevat.

2.3 SDN-ohjaimen ja OpenFlow-kytkimen välinen yhteys

SDN-ohjain ja OpenFlow-kytkin keskustelevat toistensa kanssa OpenFlow-protokollan määrittämällä viesteillä. Toisen osapuolen tekemiin pyyntöihin täytyy vastata oikean tyyppisellä vastauksella, tai poikkeustapauksissa oikealla virheviestillä. Erilaiset tilanteet tietoverkossa aiheuttavat tietynlaisia OpenFlow-pyyntöjä.

2.3.1 Hello-tilanne

Jokainen OpenFlow-yhteys aloitetaan Hello eli "tervehdys" -sekvenssillä. Sen ainoa tarkoitus on neuvotella osapuolien välillä mitä OpenFlow-protokollan versiota yhteydessä tullaan käyttämään. (Ks. kuvio 1.)

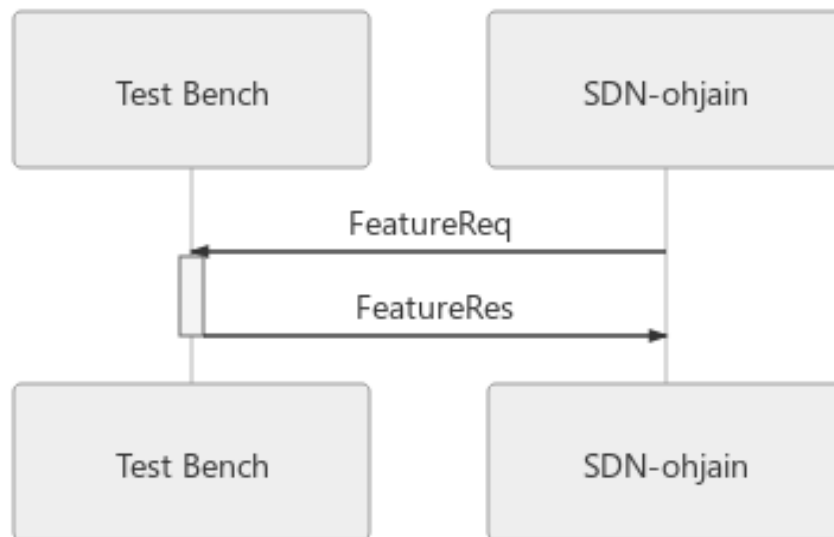


Kuvio 1. Hello-tilanteen sekvenssikaavio

Molemmat osapuolet lähettävät toisilleen Hello-viestissä korkeimman OpenFlow-protokollan version numeron jota ne tukevat, joista yhteyteen valitaan korkein yhteinen versio. Jos yhteistä versiota ei löydy, viestiin tulee vastata Error-viestillä jonka tyyppi on "HelloFailed". (OpenFlow Switch Specification 2012, 25.)

2.3.2 Handshake-tilanne

Onnistuneen Hello-tilanteen jälkeen tulee ns. Handshake eli "kädenpuristus" -sekvenssi (ks. kuvio 2), jolla SDN-ohjain käytännössä kysyy, mitä OpenFlow-protokollan ominaisuuksia OpenFlow-kytkin tukee. Ominaisuustiedoilla SDN-ohjain tietää, millaisia viestejä ja käskyjä se voi tulevaisuudessa lähettää OpenFlow-kytkimelle.



Kuvio 2. Handshake-tilanteen sekvenssikaavio

2.3.3 Error-tilanne

Seuraavassa testitapauksessa testataan OpenFlow-protokollan Error eli virhetilanne viestejä. Error-viestejä käytetään OpenFlow-protokollassa minkä tahansa protokollaan liittyvän operaation epäonnistumisen viestimiseen. Se voi tulla esimerkiksi epäyhteensopivista protokollan versioista Hello-tilanteessa. Tai jos vaikka jotakin OpenFlow-käskyä ei voida suorittaa verkkolaitteen virhetilan vuoksi.

Error-tilanteiden luomisen ja testaamisen tavoite on varmistaa että SDN-ohjain lähettää virhetilanteissa oikeanlaiset virheilmoitukset. Tämä on erittäin tärkeää sekä OpenFlow-kytkimen sekä kehittäjän kannalta. Jos SDN-ohjain vastaa virheellisellä virheilmoituksella, OpenFlow-kytkin saattaa reagoida siihen epätoivotulla tavalla tai kehittäjä voi luulla virheen johtuvan jostain aivan muusta asiasta. OpenFlow-kytkimen tapauksessa se voi sekoittaa tietoverkon toimintaa ja pahimmillaan ehkä jopa laumauttaa sen, ja kehittäjän tapauksessa väärän vian etsiminen maksaa aikaa ja mahdollisesti todella paljon rahaa.

3 Kehitetty SDN-ohjainten Test Bench testaussovellus

3.1 Test Bench ohjelmiston suunnittelu

Suunnittelun alussa kartoitettiin muutama Test Benchin vaatimus vaatimusmäärittelyn muodossa. Vaatimusmäärittelyn tekemisen tarkoitus oli saada jonkinlainen kokonaiskuva toteutuksen tarpeista ja siitä mitä ollaan tekemässä. Luotu vaatimusmäärittely ei ota kantaa C++-yksityiskohtiin, vaan ainoastaan muutamaa toiminnalliseen vaatimukseen.

3.1. User

3.1.1. Specifying a controller host address

ID: UFR1

Description: The user should be able to input the host address of a controller in case it's not hosted on the same system (localhost).

Rationale: In order to communicate with a remote controller.

Dependencies: None

Date: 22.6.2018

3.1.2. Specifying a controller protocol port

ID: UFR2

Description: The user should be able to input the port(s) used by a controller in case it's not the protocol default.

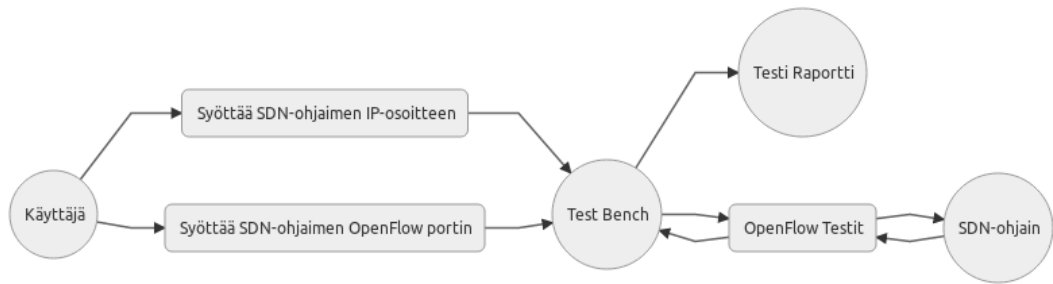
Rationale: In order to communicate with a controller using some other port.

Dependencies: None

Date: 22.6.2018

Kuvio 3. Vaatimusmäärittelyn kaksi ensimmäistä käyttäjävaatimusta

Ensimmäiset määritetyt vaatimukset ovat käyttäjävaatimuksia (ks. kuvio 3). Test Benchin käyttötarkoitukseen ei tarvita graafista käyttöliittymää, pelkkä komentoriviltä ajettava työkalu riittää.



Kuvio 4. Test Benchin vuorovaikutussuhteet

Teoriassa työkalulle tarvitsee syöttää parametrina ainoastaan testattavan SDN-ohjaimen osoite (IP-osoite tai isäntänimi). Lisäksi voidaan syöttää tarvittaessa portin numero, jossa SDN-ohjain kuuntelee OpenFlow-liikennettä, jos se on eri kuin standardi. Tämän jälkeen Test Bench hoitaa testien ajamisen ja testituloksien tulostamisen. (Ks. kuvio 4.) Nämä molemmat käyttäjävaatimukset toteutettiin.

3.1.3. Specifying a protocol to test

ID: UFR3

Description: The user should be able to limit which tests are performed on the controller by inputting a protocol and its version.

Rationale: The user might be interested to only see the results of a single protocol and its version.

Dependencies: None

Date: 29.6.2018

Kuvio 5. Vaatimusmäärittelyn kolmas käyttäjävaatimus

Kolmantena käyttäjävaatimuksena oli testattavan protokollan valinta (ks. kuvio 5), mutta sen toteuttamiselle ei ollut tarvetta.

3.2. OpenFlow Testing

3.2.1. Negotiating protocol versions

ID: OFFR1

Description: The system should be able to send OpenFlow Hello messages to learn which OpenFlow versions the controller claims to support.

Rationale: In order to learn which protocol versions the controller supports and limit testing to those protocol versions.

Dependencies: None

Date: 1.7.2018

3.2.2. Feature determination

ID: OFFR2

Description: The system should be able to respond to an OpenFlow FeatureReq message with a FeatureRes message. This is used to tell the controller which features can be used and therefore possibly tested.

Rationale: In order to have the controller send required messages for testing.

Dependencies: OFFR1

Date: 1.7.2018

Kuvio 6. OpenFlow-vaatimukset vaatimusmäärittelyssä

Käyttäjävaatimusten lisäksi määriteltiin kahden ensimmäisen OpenFlow-testin vaatimukset (ks. kuvio 6).

Koska työkalun täytyy pystyä kommunikoimaan testattavan SDN-ohjaimen kanssa, yksi Test Benchin käytön vaatimuksista on se, että Test Bench pitää pystyä sijoittamaan ohjaimen ohjauskerrokseen (Control Plane). Ohjauskerroksen tulee olla eristettynä asiakaskoneiden liikenteestä (Data Plane), jotta SDN-ohjainta vastaan hyökkäminen on hankalampaa. Eristys voidaan tehdä fyysisesti vetämällä molemmille kerroksille omat verkkojohdot, mikä on paljon turvallisempaa, tai sitten luomalla virtuaalisia lähiverkkoja (VLAN).

Mitään ohjelmistoa ei voida suunnitella kokonaan täydellisesti etukäteen, ja se pätee yhtä lailla Test Benchin. Test Benchin perusvaatimukset ja tarpeet voidaan rajata, mutta opinnäytetyön tekijän verkko-ohjelmoinnin kokemuksen puutteen vuoksi asioita täytyy vain testata ja kokeilla, miten ja mitkä asiat toimivat. Yksi tällainen asia oli SDN-ohjaimelta sisääntulevan datan käsittely. Toteutuksessa käytännössä katsottiin, millaista dataa SDN-ohjain lähetti, ja datan käsittelyä kehitettiin sen mukaan esimerkiksi, miten virheellinen data pitää ottaa huomioon lähdekoodissa tarkistuksineen.

3.2 C++:n yksityiskohtia toteutuksessa

3.2.1 Moderni C++

Test Benchin toteutuksen ohjelmointikieleksi valikoitui C++. Moderni C++ on erinomainen valinta tähän projektiin, koska sillä voidaan määrittää OpenFlow-protokollan viestien kenttien tietotyyppien koot bittitarkkuudella. C++:lla on ehkä vähän huono maine C++98:n monimutkaisuuden vuoksi, mutta C++11 ja sitä uudemmat versiot ovat tehneet C++:sta paljon helpomman ja turvallisemman kielen käyttää.

Open Networking Foundationin OpenFlow-protokollan referenssi-implementaatiossa käytetään C-ohjelmointikieltä, mutta C-kielellä kaiken itse tekeminen olisi vieläkin haastavampaa ja se veisi pitempään. Lisäksi C++ antaa tietynlaista turvallisuutta varsinkin muistinhallinnan kannalta.

C++ itsessään on vain standardi, jonka eri ns. kääntäjät (Compiler) toteuttavat omalla tavallaan. Kääntäjä itsessään on ohjelmisto, joka muuntaa ihmiskoodin tietokoneen ymmärtämään binääriseen muotoon. C++-kääntäjiä on olemassa mm. Microsoftin Visual C++, GCC eli GNU Compiler Collection, ja Clang.

C++ on jatkuvasti kehittyvä ohjelmointikieli, jonka uusimman, kääntäjien tukeman, standardin versio kirjoittamishetkellä on C++17. C++20 on vasta kääntäjillä toteuttamisvaiheessa. Standardin ensimmäinen version nimi on C++98. Siitä seuraava versio C++03 oli suurimmalta osin bugien korjaus päivitys standardiin.

3.2.2 Tiedonsiirto C++:lla

OpenFlow-protokollan yhteyksissä käytetään TCP (Transmission Control Protocol) protokollaa, jolla varmistetaan että viestit pääset kokonaisina perille (TCP/IP Protocol Architecture Model 2010). OpenFlow-viestit esitetään Test Benchissä luokkina. OpenFlow-viestin lähettäminen toimii muuntamalla luokista luodut objektit merkkijonoiksi ja kirjoittamalla ne yhteyden TCP-socket tiedostoon. Vastaavasti viestejä vastaanotetaan lukemalla merkkijonoja TCP-socketista ja parsimalla ne taas viestiobjekteiksi. Merkkijonot käsitellään yksi merkki eli tavu kerrallaan. Tämä tarkoittaa sitä,

että tavua isommat tietotyypit täytyy viestejä lähettäessä pilkkoa tavun kokoisiksi palloiksi ja viestejä vastaanottaessa tavut järjestellään jälleen isommiksi tietotyypeiksi tarpeen mukaan.

Tiedonsiirrossa yksi hyvin tärkeä asia joka pitää myös ottaa huomioon on tavujärjestys (Endianness tai byte order). Tietokoneen prosessori käsittelee tavuja muistissa arkkitehtuurista riippuen jossain tavujärjestyksessä. Big-endian järjestyksessä eniten merkitsevä tavu (MSB) tallennetaan muistiin ensin, eli vasemmalta oikealle. Little-endian järjestys on taas päinvastainen, eli vähiten merkitsevä (LSB) tulee ensin. (Byte Ordering, n.d.)

Jos tavujärjestystä ei oteta huomioon, merkkijonot voivat saapua vastaanottajalle päinvastaisessa järjestyksessä. OpenFlow-protokolla määrää että kaikki OpenFlow-viestit tulee lähettää big-endian järjestyksessä (OpenFlow Switch Specification 2012, 34).

Taulukko 1. Esimerkki tavujärjestyksistä

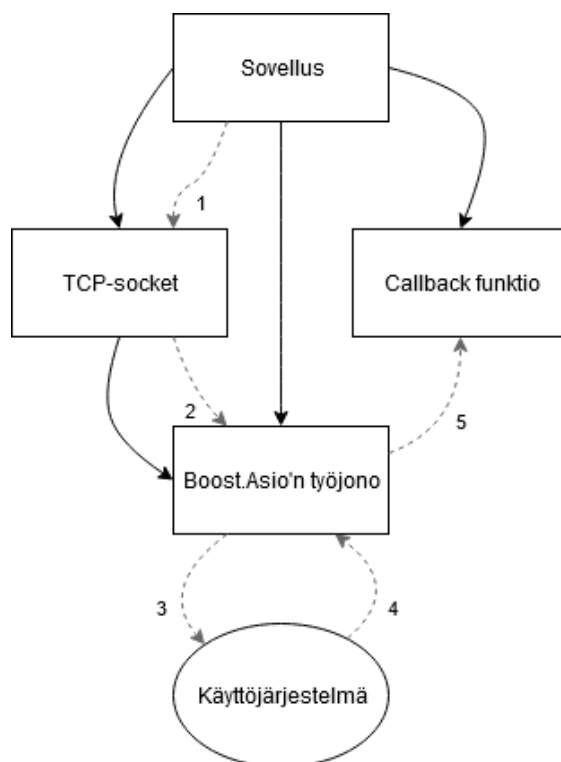
Muistipaikka	p	p+1	p+2	p+3
Big-endian	'S'	'D'	'N'	'\0'
Little-endian	'\0'	'N'	'D'	'S'

3.2.3 Boost.Asio

On monia tapoja luoda TCP-yhteyksiä C++:lla tiedonsiirtoon. Windowsilla ja Unix-pohjaisilla käyttöjärjestelmillä on omat kirjastonsa matalatason TCP-sockettien käyttämiseen. Test Bench -sovellukseen valikoitui kuitenkin korkeamman tason

Boost.Asio-kirjasto sen tuomien valmiiden ominaisuuksien vuoksi. Boost.Asio on alustariippumaton C++-kirjasto, joka on tarkoitettu yhteyksien luontiin ja tiedonsiirtoon. Kirjaston nimi tulee sanoista asynchronous input/output. Se tukee mm. TCP- ja UDP-protokollia, IPv4 sekä IPv6 osoitteita ja asynkronista koodia.

Asynkroninen koodi tarkoittaa käytännössä sitä, että asynkronisten koodinpätkien suoritusta ei jäädä odottamaan, vaan tulokset käsitellään myöhemmin, kun prosessorilla on ns. aikaa. Asynkronisuus mahdollistaa ohjelman tehokkaan suorittamisen ja jopa useamman tuhannen yhtäaikaista yhteyden. Jos prosessori jäisi odottamaan yksittäisen yhteyden tehtävien suorittamista, se pahimmillaan blokkaisi kokonaisen prosessorin ytimen ja yhtäaikaista yhteyksiä ei voisi prosessorista riippuen olla kuin muutama.



Kuvio 7. Boost.Asio-kirjaston toiminta (Kohlhoff, C. N.d., muokattu)

Boost.Asion toiminta on suhteellisen yksinkertainen. Boost.Asio luo itselleen työjonon, johon voidaan lisätä tehtäviä. Näitä tehtäviä voi olla mm. yhteyden avaaminen, datan lähettäminen tai vastaanottaminen, tai ehkäpä yhteyden aikakatkaisu. Ohjelmoijan päättää milloin Boost.Asio suorittaa työjonon tehtäviä, josta on ainakin

se hyöty että suoritus voidaan laittaa tauolle jos tarve vaatii. Työjonon tehtäville annetaan jokin valmistumiskriteeri. Kun kriteeri täyttyy, Boost.Asio kutsuu ohjelmoijan määrittämän funktion eli ns. callback funktion. (Ks. kuvio 7.)

Boost.Asio hoitaa alla olevan käyttöjärjestelmän/kernelin järjestelmäkutsut, jonka ansiosta Boost.Asio on alustariippumaton.

Boost.Asio komponenttien käyttäminen

Boost.Asion komponenttien käyttäminen sellaisenaan on hieman kömpelöä ja lisäksi se luo riippuvaisuuden Boost.Asioon kaikkialla koodissa. Tämän vuoksi kannattaa luoda Boost.Asion luokille omat ns. wrapperit eli luokat, jonka kautta Boost.Asion komponentteja käytetään. Tällä tavoin jos Boost.Asio joskus haluttaisiin korvata jollain muulla kirjastolla tai toteutuksella, tarvitsee vain muokata näitä ns. wrapper luokkia. Muutoin koodia jouduttaisiin kirjoittamaan uudelleen jokaisessa projektin tiedonsiirtoon liittyvässä luokassa tai funktiossa.

Boost.Asio tcp::socket

Olennaisin Boost.Asion komponentti on tcp::socket. Se on pohjana yhteyden muodostamiseen ja tiedonsiirtoon.

```
auto error = boost::system::error_code{};
auto host = boost::asio::ip::make_address(address, error);
if (error) {
    logger->error("Connection attempt failed: {}. ", error.message());
    return;
}
socket->async_connect({host, port}, on_connect);
```

Kuvio 8. Yhteyden muodostaminen Boost.Asioilla

Boost.Asio tekee yhteyden muodostamisesta helppoa hoitamalla alemman tason kutsut käyttöjärjestelmälle. (Ks. kuvio 8.)

```

auto constexpr action_timeout = std::chrono::seconds{10};
auto constexpr bytes_to_read = uint_fast8_t{8};

auto callback = std::bind(&handshake::handle_featurereq, this, std::placeholders::_1);
of_connection->async_read(bytes_to_read, callback);
logger->info("Waiting for FeatureReq..");

io_context.run_one_for(action_timeout);

```

Kuvio 9. Lukutehtävän luonti Boost.Asion tehtäväjonoon

Myös datan lukeminen TCP-socketista suhteellisen yksinkertaista. Lukutehtävän valmistumiskriteeriksi laitetaan esimerkiksi ”kun kahdeksan tavua on luettu TCP-socketista” (ks. kuvio 9).

```

logger->info("Received a response!");

auto response = of_connection->get_read_data();

auto message = common::network::openflow::message{};
message.parse_header(response);

if (not message.get_type()) {
    logger->error("Received an invalid response (invalid header or message type).");
    return;
}

auto header = message.get_header().value();

if (header.version != +common::openflow::version::openflow13) {
    logger->error("FeatureReq message did not have OpenFlow v1.3 version in the header!");
    // TODO: fault tolerance for tests, e.g. continue test in this case if wanted.
    return;
}

```

Kuvio 10. Vastaanotetun datan käsittely callback funktiossa

Callback funktiossa tiedetään aina, että valmistumiskriteeri on täytetty. Muutoin Boost.Asio ei kutsu sitä. Esimerkiksi em. lukutehtävän tapauksessa tiedetään, että TCP-socketissa on vähintään kahdeksan tavua dataa. (Ks. kuvio 10.)

3.3 OpenFlow'n tietorakenteet

Tiedonsiirtokomponenteilla voidaan lähettää mitä tahansa dataa vastaanottajalle. Kommunikointi SDN-ohjainten kanssa tapahtuu OpenFlow-protokollan viesteillä. Kommunikointia varten tarvitaan OpenFlow-viestien mukaiset tietorakenteet. Näillä tietorakenteilla muodostettavat OpenFlow-viestit laitetaan lähetettäessä OpenFlow-protokollan mukaiseen merkkijonoon. Eli OpenFlow-protokollan kannalta sillä ei ole väliä miten tietorakenteet toteutetaan, kunhan lopputulos, eli lähetettävä merkkijono on OpenFlow-protokollan mukainen.

Open Networking Foundationin -säätiön verkkosivuilta löytyy referenssi implementaatio tietorakenteista C-ohjelmointikielellä. Ne oltaisiin voitu kopioida suoraan, sillä C++:ssa voidaan käyttää suoraan C:n tietorakenteita, mutta oppimisen kannalta tietorakenteet haluttiin tehdä itse. Lisäksi niiden kopioiminen olisi vaatinut ONF:n koodin lisenssin lisäämistä Test Bench projektiin.

```

30  /// common header used in every openflow message independent of protocol version.
31  ///
32  struct alignas(openflow_header_size) header
33  {
34      /// constructor
35      ///
36      header();
37
38      /// constructor
39      ///
40      /// \param version_ is an openflow protocol version
41      /// \param type_ is an openflow message type
42      /// \param length_ is the total length of an openflow message
43      /// \param xid_ is a transaction identifier
44      header(uint8_t version_, uint8_t type_, uint16_t length_, uint32_t xid_);
45
46      uint8_t version; ///< protocol version
47      uint8_t type;    ///< message type
48      uint16_t length; ///< length of the message including this header
49      uint32_t xid;    ///< transaction identifier used to match requests to responses
50  };
51  static_assert(sizeof(header) == openflow_header_size);

```

Kuvio 11. OpenFlow-viestin ylätunnisteen rakenne

Jokainen OpenFlow-viesti alkaa yhteisellä header-tietorakenteella eli ylätunnisteella. Ylätunnisteen tarkoitus on välittää yleistä tietoa viestistä, sekä helpottaa OpenFlow-viestin parsimista, sillä ylätunniste on aina saman pituinen. Ylätunnisteen tietokentät kertovat mikä OpenFlow-protokollan versio on kyseessä, mikä kyseisen viestin tyyppi on, kuinka pitkä viesti on kokonaisuudessaan ja neljäntenä on "transaction identifier" eli ns. tapahtumatunniste, jolla viestien pyynnöt ja vastaukset voidaan yhdistää toisiinsa. (Ks. kuvio 11.)

Ylätunnisteen jälkeen tulee viestikohtainen tietorakenne (payload). Ylätunnisteen rakenne on sama jokaisessa protokollan versiossa, mutta "payload" vaihtelee riippuen viestistä sekä käytetystä protokollan versiosta.

OpenFlow-protokolla määrittää ylätunnisteen kenttien arvot hexadesimaali muodossa. Esimerkiksi OpenFlow-protokollan version 1.3 arvo on 0x04 (koska 1.0 on 0x01).

3.4 Test Benchin rakenne

Test Bench on sovellus, joka on tarkoitus ajaa vain kerran läpi, eli se ajaa testit järjestyksessä ja lopuksi palauttaa koodin joka kertoo menikö kaikki testit läpi. Tämä yksinkertaistaa ohjelman rakennetta, sillä ohjelma suorittaa vain yhden tehtävän kerrallaan.

3.4.1 Test Suite

Tarvitaan yleinen luokka, joka vastaa testien ajamisesta järjestyksessä ja niiden tuloksien kirjaamisesta. Testit itsessään jaetaan pienempiin osiin ja jokaiselle OpenFlow-protokollan versiolle on tietenkin omat testinsä. Test Suiten vastuulla on myös yhteyksien hallinnoiminen ja antaminen itse testitapauksille, jotta testejä voidaan tarpeen mukaan ajaa putkeen yhdellä samalla yhteydellä.

3.4.2 Hello-tilanteen testaus

OpenFlow-yhteyden Hello-tilanne on ensimmäinen testi, joka Test Benchillä voidaan tehdä. Testin tavoitteena on, että SDN-ohjain vastaa OpenFlow-protokollan mukaisesti tukevuksensa protokollan versiota 1.3.

Tilanteen luomiseksi Test Benchin tarvitsee vain rakentaa ja lähettää SDN-ohjaimelle OpenFlow'n Hello-viesti.

Hello-viesten rakentaminen

OpenFlow'n Hello-viestiin sisältyy OpenFlow Header eli ylätunniste sekä lista tuetuista protokollan versioista (ks. kuvio 12).

```

32  /// extend openflow hello element type with a list of protocol version bitmaps
33  ///
34  struct hello
35  {
36      /// constructor
37      ///
38      hello();
39
40      openflow::header header;          ///< openflow header
41      std::vector<openflow::hello::version_bitmap> elements; ///< sequence of hello elements
42  };

```

Kuvio 12. Hello-viestin rakenne

Ylätunnisteeseen täytyy asettaa OpenFlow-protokollan version 1.3 arvo, joka esittää OpenFlow-protokollassa heksadesimaalilukuna. Sen arvo on 0x04, sillä versio 1.0 on 0x01. Sen jälkeen asetetaan viestin tyyppi, joka on Hello. Se on OpenFlow-protokollan ensimmäinen viesti, joten sen arvoksi OpenFlow'n spesifikaatiossa on määritetty 0x0000. Kolmas kenttä on viestin kokonaispituus tavuina. Siihen lasketaan mukaan ylätunnisteen pituus sekä sen jälkeen tulevan OpenFlow-viestin pituus. Viimeinen kenttä on ns. Transaction ID. Sillä voidaan yhdistää yhteyden aikana lähetetyt viestit toisiinsa. Sen arvolla ei ole tässä tapauksessa väliä joten laitetaan sen arvoksi vain jokin satunnainen luku.

OpenFlow-protokollan versioon 1.3.0. asti Hello-viesti koostui pelkästä ylätunnisteesta. Versiossa 1.3.1. Hello viestiin lisättiin HelloElement-tietorakenne. HelloElement koostuu pituudesta, tyypistä sekä tyyppikohtaisesta tietorakenteesta. HelloElement -tyyppejä on tähän asti määritelty OpenFlow-protokollassa ainoastaan yksi: Version Bitmap. Se on yksinkertaisesti lista kaikista Hello-viestin lähettäjän tukemista OpenFlow-protokollan versioista. Pienenä lisänä tarkistetaan, että jokaisen Version Bitmapin vähiten merkitsevä bitti (LSB) on nolla, koska OpenFlow-protokolla on määrittänyt sen olevan aina nolla.

Kun Hello-viesti on rakennettu, voidaan TCP-yhteys muodostaa testattavaan SDN-ohjaimeen. Koska Test Bench aloittaa TCP-yhteyden, sen täytyy lähettää Hello-viesti ensimmäisenä. Viestin lähettämiseksi Hello-viesti täytyy ns. sarjallistaa (Serialize) merkkijonoksi (ks. kuvio 13). Käytännössä tämä tarkoittaa viestin tietokenttien tavujen laittamista merkkijonoon oikeassa järjestyksessä.

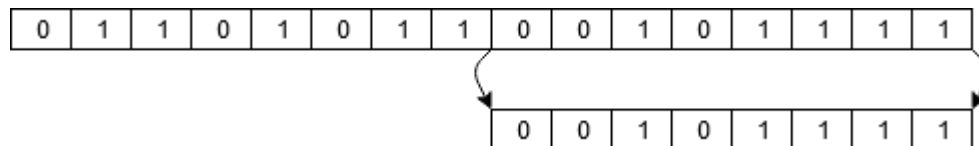
```

74  template<typename T>
75  auto inline message::message_model<T>::get_data() const -> std::optional<std::string>
76  {
77      if (not is_valid_header(message.header))
78          return std::nullopt;
79
80      auto data = std::string{};
81
82      // serialize header
83      data.push_back(message.header.version);
84      data.push_back(message.header.type);
85      data.push_back(message.header.length >> 8);
86      data.push_back(message.header.length);
87      data.push_back(message.header.xid >> 24);
88      data.push_back(message.header.xid >> 16);
89      data.push_back(message.header.xid >> 8);
90      data.push_back(message.header.xid);
91
92      // serialize message specific payload
93      switch(get_version().value()) {
94      case common::openflow::version::openflow13:
95          data += common::openflow::openflow13::message::serialize_payload(message);
96          break;
97      default:
98          return std::nullopt;
99      }
100
101      return data;
102  }

```

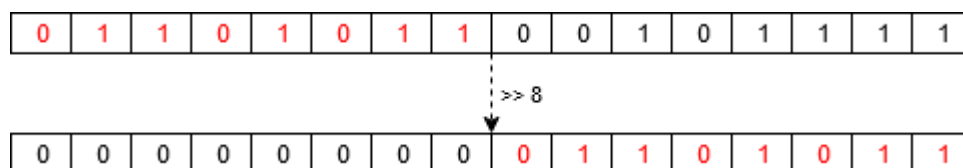
Kuvio 13. OpenFlow-viestin ylätunnisteen sarjallistaminen

Merkkijonoon lisätään ensin viestin kaikki ylätunnisteen tietokenttien tavut big-endian järjestyksessä. Protokollan versio sekä viestin tyyppi ovat yhden tavun pituisia, joten ne voidaan lisätä suoraan merkkijonoon. Ylätunnisteen viestin pituus -kenttä on kaksi tavua pitkä ja transaction ID on neljä tavua pitkä. Tässä voidaan käyttää ns. bit shift operaatiota, eli bittien siirtoa, yli yhden tavun pituisten tietokenttien pilkkomiseksi yksittäisiksi tavuiksi merkkijonoon. C++ standardi kirjaston `std::string` luokan `push_back` metodilla voidaan lisätä merkkijonoon vain yksi tavu. Tämä tarkoittaa sitä, että merkkijonoon lisättävän tietokentän bittejä luettaessa oikealta vasemmalle kaikki bitit kahdeksannen jälkeen putoavat pois (ks. kuvio 14).



Kuvio 14. Esimerkki merkkijonon bittien leikaantumisesta, kun siitä otetaan yksi tavu

Tämä on toivottava ominaisuus. Oikean tavun lisääminen merkkijonoon tapahtuu siirtämällä kaikkia tietokentän bittejä tietyn verran oikealle. Kun esimerkiksi 32 bittiä pitkän merkkijonon bittejä siirretään 24 kertaa oikealle, bittijonon 24 vähiten merkitsevää (LSB) bittiä ns. vuotaa yli ja häviää, jolloin kahdeksan eniten merkitsevää (MSB) bittiä ovat nyt kahdeksan vähiten merkitsevää bittiä. (Ks. kuvio 15.) Ja nyt kun kenttä lisätään merkkijonoon em. `push_back` metodilla, nämä kahdeksan bittiä päättyy merkkijonoon.



Kuvio 15. Esimerkki bittien ylivuotamisesta, kun bittejä siirretään 8 kertaa oikealle

Vastauksen vastaanotto

Kun merkkijono viestistä on muodostettu, se voidaan lähettää testattavalle SDN-ohjaimelle. Sen jälkeen jäädään odottamaan vastausta. Ohjaimen pitäisi vastata omalla Hello viestillään.

Vastauksen saavuttua on päinvastainen tilanne. Yhteyden TCP-socketista saadaan merkkijono, josta täytyy parsia OpenFlow-viesti. TCP-protokolla varmistaa että kaikki tieto tulee perille järjestyksessä, mutta se ei välttämättä tule perille kerralla. Tämän takia tarvitaan tilakoneen (State Machine) viestien parsimiseen tavu kerrallaan. Tilakoneen tarkoitus on tallentaa tieto siitä mitä viestin tavua ollaan parsimassa, jotta parsimisen suoritusta voidaan jatkaa oikeasta kohdasta tapauksissa, jossa koko viesti ei saavu kerralla kokonaan. (Ks. kuvio 16.)

```

136 auto header::parse_length(char input) -> boost::tribool
137 {
138     switch (parser_state) {
139     case state::parse_length_1:
140         openflow_header.length = (input >> 8) & 0xff;
141         parser_state = state::parse_length_2;
142         return boost::indeterminate;
143     case state::parse_length_2:
144         openflow_header.length = input & 0xff;
145         parser_state = state::parse_xid_1;
146         logger->trace("Header length parsed: {}", openflow_header.length);
147         return boost::indeterminate;
148     default:
149         return false;
150     }
151 }

```

Kuvio 16. Viestin pituuden parsiminen ylätunnistukseen

Tilakoneen täytyy myös pystyä myös edustamaan kolmea erilaista tilaa:

- Viestin parsiminen onnistui.
- Viestin parsiminen epäonnistui.
- Ei tarpeeksi tietoa viestin parsimiseksi.

Tätä varten voidaan käyttää Boost.Tribool kirjaston tribool luokkaa. Sillä voidaan edustaa kolmea boolean tilaa: true, false ja indetermined. True arvolla tiedetään,

että kokonainen OpenFlow-viesti on vastaanotettu, false kertoo että saapunut merkijono ei ole OpenFlow-viesti ja indeterminated arvon avulla voidaan jäädä odottamaan lopun datan saapumista.

Vastauksen käsittely

Hello-testin ei pitäisi koskaan epäonnistua, sillä se on edellytys koko OpenFlow-protokollalle ja kommunikoimiselle. Jos se jostain syystä epäonnistuu, SDN-ohjaimen toteutuksessa täytyy olla iso virhe.

Hello-tilanteesta saadaan tarkistettua, että SDN-ohjain osaa vastata Hello-viestillä ja että Hello-viestin ylätunnisteen kentät on laitettu oikein. Ylätunnisteessa pitäisi olla oikea viestin tyyppi (Hello), ja viestin kokonaispituuden pitäisi vastata viestin oikeaa pituutta. OpenFlow-viestin ylätunnisteessa on normaalisti se OpenFlow-protokollan versio jota viesti käyttää, mutta Hello-viestin kohdalla se on korkein versio, jota sen lähettäjä tukee. Lisäksi OpenFlow-protokollan versiosta 1.3.1. lähtien Hello-viesti sisältää myös listan ns. bittikartoista (bitmap), joka on käytännössä lista tuetuista OpenFlow-protokollan versioista. Versiossa 1.3.0. tätä ominaisuutta ei vielä ole. Jos testattava SDN-ohjain lähettää tällaisen listan, myös se voidaan testata Test Benchillä. Listalle on annettu tyyppi (Version Bitmap) ja pituus. Test Bench tarkistaa, että listan tyyppi on oikea ja listan pituus on oikea ja että se on huomioitu myös ylätunnisteessa.

3.4.3 Handshake-tilanteen testaus

Handshake-tilanteessa SDN-ohjaimen tulee lähettää OpenFlow-kytkimelle FeatureReq-viesti, joka koostuu vain OpenFlow Headeristä. FeatureReq-viestiin pitää vastata FeatureRes-viestillä, jossa kerrotaan OpenFlow-kytkimen ominaisuuksista. (OpenFlow Switch Specification 2012, 52.)

Handshake-tilanteessa testataan vain, että SDN-ohjain ylipäättään lähettää FeatureReq-viestin ja osaa sen jälkeen käsitellä lähetetyn FeatureRes-viestin. Käytännössä katsotaan, että SDN-ohjain ei lähetä FeatureRes-viestin jälkeen enää mitään virheilmoitusta, joka voisi johtua SDN-ohjaimessa olevasta virheestä.

Aluksi tarkistetaan saapuneen FeatureReq-viestin ylätunnisteen tiedot. Ylätunnisteen OpenFlow-protokollan version tulee edelleen olla aiemmin neuvoteltu v1.3. (Ks. kuvio 17.)

```
auto header = message.get_header().value();

if (header.version != +common::openflow::version::openflow13) {
    logger->error("FeatureReq message did not have OpenFlow v1.3 version in the header!");
    // TODO: fault tolerance for tests, e.g. continue test in this case if wanted.
    return;
}
```

Kuvio 17. OpenFlow-protokollan version tarkistus Handshake-tilanteessa

Seuraavaksi katsotaan ylätunnisteesta että viestin tyyppi on FeatureReq. Jos ei, tarkistetaan onko saapunut viesti mahdollisesti Error-viesti. Jos viestin tyyppi on Error-viesti, dataa täytyy lukea lisää TCP-socketista koko viestin saamiseksi. (Ks. kuvio 18.) Tähän mennessä socketista on luettu vain ylätunnisteen pituuden verran dataa, sillä FeatureReq koostuu vain siitä. Mahdollisesta Error-viestistä tulostetaan virheen tyyppi ja koodi sekä testi katsotaan epäonnistuneeksi.

```
if (message.get_type().value() != common::openflow::message::get_switch_features_request) {
    logger->error("Received a wrong type of OpenFlow message.");

    if (message.get_name())
        logger->error("Message type: {}. ", message.get_name().value());

    if (message.get_type().value() == common::openflow::message::error) {
        error_msg = message;

        auto const bytes_to_read = int{header.length - 8};
        auto callback = std::bind(&handshake::get_error_payload, this, std::placeholders::_1);
        of_connection->async_read(bytes_to_read, callback);

        io_context.run_one_for(action_timeout);
    }

    return;
}
```

Kuvio 18. OpenFlow-viestin tyypin tarkistus Handshake-tilanteessa

Jos saapunut viesti on FeatureReq, muodostetaan FeatureRes-viesti lähetettäväksi SDN-ohjaimelle. FeatureReq koostuu erilaisista kentistä, jotka kertovat OpenFlow-kytkimen ominaisuuksista. (Ks. kuvio 19.)

```

31  /// present a get switch features response message
32  ///
33  struct get_switch_features_response
34  {
35      /// constructor
36      ///
37      get_switch_features_response();
38
39      openflow::header header;          ///< openflow header
40      uint64_t datapath_id;             ///< unique datapath id
41      uint32_t n_buffers;               ///< number of packet_in messages the switch can queue
42      uint8_t n_tables;                ///< number of tables in the switch
43      uint8_t auxiliary_id;            ///< how the switch treats the transport channel (master/auxiliary)
44      uint16_t padding;                ///< data padding
45      uint32_t capabilities;            ///< bitmap of switch capabilities
46      uint32_t reserved;               ///< reserved bits
47  };

```

Kuvio 19. FeatureRes-viestin rakenne

Datapath ID on uniikki tunniste, jolla tunnistetaan OpenFlow-kytkin ja sen mahdolliset virtuaaliset kytkimet. Virtuaalisia kytkimiä käytetään esimerkiksi virtuaalisissa lähiverkoissa (VLAN). Datapath ID on 8 tavua eli 64 bittiä pitkä. Sen ähiten merkitsevät (LSB) 48 bittiä on fyysisen OpenFlow-kytkimen MAC-osoite. Loput 16 bittiä voi olla mitä vain, kunhan se tekee siitä uniikin. Yleensä siihen laitetaan virtuaalisen lähiverkon tunniste (VLAN ID). Capabilities kenttään laitetaan bitmaskina kaikki tuetut ominaisuudet.

Ominaisuuksia on:

- FLOW STATS eli OpenFlow-kytkin voi kertoa ohjaimelle vuostatistiikkaa.
- TABLE STATS eli OpenFlow-kytkin voi ilmoittaa tietoja vuotaulujen tiloista.
- PORT STATS eli OpenFlow-kytkin voi ilmoittaa tietojatietoja porttien tiloista.
- GROUP STATS eli OpenFlow-kytkin voi ilmoittaa tietoja käskyryhmien tiloista.
- IP REASM kertoo että OpenFlow-kytkin voi uudelleen rakentaa IP-osoitteita.
- QUEUE STATS eli OpenFlow-kytkin voi ilmoittaa tietoja pakettijonojen tiloista.
- PORT BLOCKED kertoo että OpenFlow-kytkin voi estää ns. packet loopit kun käytössä on jokin muu protokolla kuin OpenFlow.


```

auto features_response = common::openflow::openflow13::message::get_switch_features_response{};

features_response.header.length = total_message_length;
features_response.header.xid = xid;

features_response.datapath_id = 123;
features_response.n_buffers = 0;
features_response.n_tables = 1;
features_response.auxiliary_id = 0;          ///< how the switch treats the transport channel (master/auxiliary)
features_response.capabilities |= (uint32_t)common::openflow::openflow_switch::capabilities::flow_statistics;
features_response.capabilities |= (uint32_t)common::openflow::openflow_switch::capabilities::table_statistics;
features_response.capabilities |= (uint32_t)common::openflow::openflow_switch::capabilities::port_statistics;
features_response.capabilities |= (uint32_t)common::openflow::openflow_switch::capabilities::group_statistics;
features_response.capabilities |= (uint32_t)common::openflow::openflow_switch::capabilities::ip_reassembly;
features_response.capabilities |= (uint32_t)common::openflow::openflow_switch::capabilities::queue_statistics;
features_response.capabilities |= (uint32_t)common::openflow::openflow_switch::capabilities::port_blocked;

```

Kuvio 20. FeatureRes-viestin rakentaminen

Tässä tapauksessa voidaan huoletta kertoa ohjaimelle Test Benchin tukevan kaikkia näitä ominaisuuksia, vaikka näin ei oikeasti ole (ks. kuvio 20). Test Benchillä ajetaan vain tietyt tilanteet ja sen jälkeen yhteys suljetaan. Oikeasti tuetuiden ominaisuuksien ilmoittaminen on tärkeää ainoastaan aiduille OpenFlow-kytkimille, jotka jäävät kuuntelemaan SDN-ohjaimen käskyjä loputtomiin.

Rakennetun FeatureRes-viestin sarjallistamisen ja lähettämisen jälkeen kaiken oletetaan menneen kuten pitikin, ja testi katsotaan menneen lävitse.

3.4.4 Error-tilanteen testaus

Kolmannessa tilanteessa testataan OpenFlow-protokollan Error eli virhetilanne viestejä. Tämä tilanne luodaan rakentamalla muutamia tahallisesti mahdottomia käskyjä SDN-ohjaimelle OpenFlow-viesteinä, ja toivotaan että ohjain vastaa niihin oikean tyyppisillä Error-viesteillä.

Error-viesti koostuu OpenFlow Headeristä, virhetyypistä, virhetyyppiin liittyvästi koodista sekä määrittelemättömästä merkkijonosta, johon voidaan laittaa jokin haluttu virheviesti (ks. kuvio 21).

```

31  /// present openflow error messages
32  ///
33  struct error
34  {
35      /// constructor
36      ///
37      error();
38
39      openflow::header header;      ///< openflow header
40      uint16_t type;                ///< high level error type
41      uint16_t code;               ///< type specific error code
42      std::string data;            ///< arbitrary error message string
43  };

```

Kuvio 21. Error-viestin rakenne

Testissä luodaan virheellinen EchoRequest-viesti. EchoRequest on OpenFlow:ssa ns. keepalive-toiminto, jolla yhteyttä pidetään yllä. Siksi se sopii hyvin ensimmäisten Error-viestien testaukseen, sillä sen ei pitäisi aiheuttaa muuta kuin korkeintaan yhteyden katkaisun.

Error-viestien tyyppi ja koodi itsessään kertovatkin miten Test Benchillä voidaan mahdollisesti tuottaa sellainen virheilmoitus. Testissä asetetaan EchoRequest-viestin ylätunnisteeseen OpenFlow-protokollan versioksi 1.0, vaikka Hello-tilanteessa versioiksi neuvoteltiin 1.3. (ks. kuvio 22).

```

// send an EchoRequest with invalid protocol version
auto echo_request = common::openflow::openflow13::message::echo_request{};

echo_request.header.version = +common::openflow::version::openflow10; // wrong version
echo_request.header.length = total_message_length; // total length of message with one version bitmask element
echo_request.header.xid = 456; // just some random number

```

Kuvio 22. Virheellisen EchoRequest-viestin muodostaminen

Myös Error-viesteistä tarkistetaan ylätunnisteen kentät että ne ovat asetettu oikein, ja olennaisesti että virheilmoituksen kategoria ja kategoriakohtainen virhekoodi on oikein (ks. kuvio 23).

```

auto header = message.get_header().value();

if (header.version != +common::openflow::version::openflow13) {
    logger->error("Error message did not have OpenFlow v1.3 version in the header!");
    // TODO: fault tolerance for tests, e.g. continue test in this case if wanted.
    return;
}

if (header.type != +common::openflow::message::error) {
    logger->error("Did not receive an Error for an ill-formed EchoRequest message!");
    // TODO: fault tolerance for tests, e.g. continue test in this case if wanted.
    return;
}

```

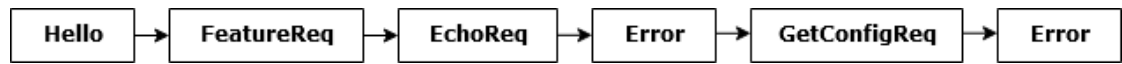
Kuvio 23. Ylätunnisteen tarkistaminen Error-viestistä

Test Benchillä ei voida testata kaikkia Error-viestin virhekoodeja. Jokaiselle OpenFlow-protokollan viestille löytyy sitä koskeva virhekoodi, jolloin kaikkien niiden testaaminen vaatisi että Test Bench pystyy rakentamaan kaikki OpenFlow-protokollan viestit.

3.4.5 Mahdollinen rajaamisen ongelma

Hello- ja Handshake-tilanteiden jälkeen OpenFlow-yhteys on ns. alustettu ja kaikki muu OpenFlow-kommunikointi voi alkaa, eli kaikki muu testaaminen voidaan aloittaa.

Nyt kumpi tahansa osapuoli voi lähettää toisilleen OpenFlow-viestejä milloin tahansa. Tästä muodostuukin yksi haaste testaamiseen. Testauksen aikana Test Bench saa todennäköisesti SDN-ohjaimelta testitapaukseen liittymättömiä OpenFlow-viestejä (ks. kuvio 24), liittyen tietoverkon käyttäytymisen määrittämiseen. Test Benchin täytyy pystyä erottamaan SDN-ohjaimelta tulevasta datavirrasta testiin kuuluvat viestit, sekä käsittelemään testiin kuulumattomat viestit jotenkin niin, että se ei vaikuta testin lopputulokseen. Tätä ongelmaa ei saatu ratkaistua opinnäytetyön aikana, mutta siitä ollaan kuitenkin tietoisia, ja se on yksi jatkokehityksen tehtävä.



Kuvio 24. Esimerkki mahdollisista saapuvista OpenFlow-viesteistä yhteyden aikana

Tähän liittyen toiseksi haasteeksi saattaa myös myöhemmin muodostua se, että Test Bench ei pysty täysin simuloimaan OpenFlow-kytkintä. On hyvin mahdollista että SDN-ohjain lähettää jonkin OpenFlow-viestin, ja jos se vain jätetään huomioimatta, SDN-ohjain saattaa ehkä sulkea yhteyden ja siten varsinainen testaaminen ei onnistu. Tätä tapausta ei ehditty todentamaan, mutta Test Benchin kehityksen aikana havaittiin, että testattava SDN-ohjain saattaa välillä sulkea yhteyden kesken testauksen. Tätä tapahtui harvoin, ja se saattoi johtua em. tapauksesta, mutta Test Benchin testien vähäisen määrän vuoksi OpenFlow-yhteys ei ole auki kuin ehkä joitakin sekunnin sadasosia. On hyvin epätodennäköistä, että SDN-ohjain sulkisi yhteyden, jos se ei saa vastausta tässä ajassa, mutta tästä voi tulla myöhemmin ongelma jos testejä tulee enemmän ja niiden ajo kestää huomattavasti pidempään.

Tämän ongelman ratkaisemista ei myöskään ehditty miettimään. Saattaa ehkä olla hyvin mahdollista että Test Benchin pitäisi pystyä täysin simuloimaan OpenFlow-kytkintä, että kaikkiin SDN-ohjainten viesteihin pystytään vastaamaan, jotta SDN-ohjain ei mahdollisesti katkaise yhteyttä.

OpenFlow-kytkimen täysi simuloiminen taas on aivan liian laaja työ opinnäytetyön resurssien puitteissa. Käytännössä tämä hoidetaan Test Benchissä vain luomalla rajatut testit ja sitten manuaalisesti tarkkailemalla miten käy kun kaikkiin OpenFlow-viesteihin ei voida vastata.

3.4.6 Testauksen automatisointi

Kun ohjelman suoritus loppuu, sen täytyy palauttaa sitä suorittavalle prosessille jokin kokonaisluku, joka kertoo ohjelman suorituksen onnistumisesta. Palautetun arvon

merkitys riippuu täysin toteutuksesta, mutta yleisesti 0 tarkoittaa onnistunutta suoritusta. Kaikki muut arvot lähes aina tarkoittavat epäonnistumista, ja eri arvoilla ohjelmat voivat viestiä erilaisista virheistä. (std::exit, 2019.)

Test Benchin suoritus voidaan automatisoida. Testausta suorittava prosessi tietää ohjelman palauttamasta arvosta onnistuiko testit vai ei. Test Bench palauttaa koodin 0, kun kaikki testitilanteet ovat menneet läpi, ja koodin 1, jos yksikin testitilanne epäonnistuu.

3.4.7 Testitulosten raportointi

Test Bench tulostaa testitulokset käyttäjälle sovelluksen suorituksen aikana. Test Bench ilmoittaa käyttäjälle aina mitä testitilannetta se suorittaa, sekä sen lopputuloksen. Testitilanteissa voidaan myös tehdä erilaisia päätelmiä testattavasta SDN-ohjaimesta ja myös nämä havainnot tulostetaan käyttäjälle.

Test Bench kirjoittaa tulokset stdout (standard output) nimiseen ulostuloon. Stdout ulostulo näkyy oletuksena käyttäjän komentorivillä (Stdout, 2019). Testiautomaatiota suorittavat prosessit yleensä tallentavat stdout'in tulosteet. Jos testit eivät mene läpi, käyttäjä voi jälkikäteen käydä lukemassa Test Benchin suorituksen tulosteet.

4 Testit ja testitulokset

4.1 Faucet SDN -ohjain

Kehityksen aikana Test Benchin testauskohteena toimi Faucet-niminen SDN-ohjain. Faucet valittiin siksi, että sen saa helposti Docker-konttina ja se toimii kätevästi suoraan pelkällä kontin käynnistyksellä.

Test Benchin ns. kääntämisen jälkeen ajettava tiedosto ilmestyy bin nimiseen kansioon, josta sitä voidaan ajaa komentoriviltä. Test Benchille tarvitsee ainoastaan syöttää Faucet'in IP-osoite ja portti.

```
./bin/weftworks-test-bench -c 127.0.0.1 --of-port 6653
```

```

X antza@antza: ~/weftworks/test-bench/build  a dev  ./bin/weftworks-test-bench -c 127.0.0.1 -p 6653
[2020-01-15 18:06:30.633] [info]
[2020-01-15 18:06:30.633] [info] Weftworks SDN Test Bench - Copyright (C) 2019 Antti Lohtaja
[2020-01-15 18:06:30.633] [info] This program comes with ABSOLUTELY NO WARRANTY.
[2020-01-15 18:06:30.633] [info] This is free software, and you are welcome to redistribute
[2020-01-15 18:06:30.633] [info] it under certain conditions.
[2020-01-15 18:06:30.633] [info]
[2020-01-15 18:06:30.633] [info] Running tests..
[2020-01-15 18:06:30.633] [info] Testing for OpenFlow protocol v1.3..
[2020-01-15 18:06:30.633] [info] Connected to target controller.
[2020-01-15 18:06:30.633] [info] Running OpenFlow protocol v1.3 Hello test..
[2020-01-15 18:06:30.633] [info] Sending OpenFlow Hello..
[2020-01-15 18:06:30.633] [info] OpenFlow Hello sent.
[2020-01-15 18:06:30.633] [info] Waiting for response..
[2020-01-15 18:06:30.634] [info] Received a response!
[2020-01-15 18:06:30.634] [info] Target controller seems to only support OpenFlow v1.3.0 or earlier (No payload / HelloElements).
[2020-01-15 18:06:30.634] [info] OpenFlow v1.3 Hello passed!
[2020-01-15 18:06:30.634] [info] Running OpenFlow protocol v1.3 Handshake test..
[2020-01-15 18:06:30.634] [info] Waiting for FeatureReq..
[2020-01-15 18:06:30.634] [info] Received a response!
[2020-01-15 18:06:30.634] [info] Successfully received a FeatureReq message!
[2020-01-15 18:06:30.634] [info] Sending OpenFlow FeatureRes in response..
[2020-01-15 18:06:30.634] [info] OpenFlow FeatureRes sent.
[2020-01-15 18:06:30.634] [info] OpenFlow v1.3 Handshake passed!
[2020-01-15 18:06:30.634] [info] Running OpenFlow protocol v1.3 Errors test..
[2020-01-15 18:06:30.634] [info] Sending OpenFlow EchoRequest..
[2020-01-15 18:06:30.634] [info] OpenFlow EchoRequest sent.
[2020-01-15 18:06:30.634] [info] Waiting for response..
[2020-01-15 18:06:30.635] [info] Received a response!
[2020-01-15 18:06:30.635] [error] Received an invalid response (invalid header or message type).
[2020-01-15 18:06:30.635] [error] OpenFlow v1.3 Errors failed.
[2020-01-15 18:06:30.635] [error] One or more tests failed for OpenFlow protocol v1.3.
X antza@antza: ~/weftworks/test-bench/build  a dev

```

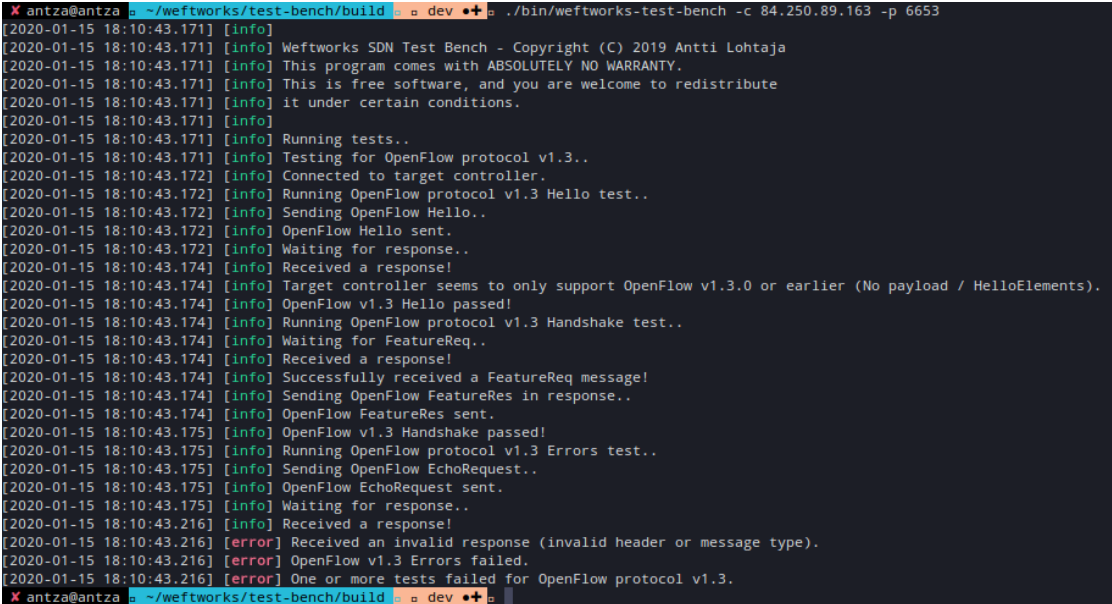
Kuvio 25. Tuloste Faucet SDN-ohjaimen testauksesta

Tuloksista nähdään, että Test Bench onnistuu luomaan yhteyden Faucet'iin ja pystyy keskustelemaan sen kanssa. Test Bench suorittaa testitapaukset järjestyksessä, joista Hello ja Handshake onnistuu. Errors-testitapaus ei kuitenkaan mene läpi. (Ks. kuvio 25.)

Hello-testitapauksesta selviää, että Faucet käyttää OpenFlow-protokollan versiota 1.3.0. Versiossa 1.3.1 protokollan Hello viesteihin tuli lisäksi lista tuetuista protokollan versioista (HelloElement), mutta Faucetin lähettämässä viestissä niitä ei ollut. Tästä voi päätellä että, Faucet todennäköisesti tukee tarkalleen vain versiota 1.3.0, mutta on myös täysin mahdollista, että Faucetin toteutuksessa on virhe jos Faucetin tarkoitus on käyttää esimerkiksi versiota 1.3.3. Faucetin kehittäjät voisivat nyt käyttää tätä tietoa hyödyksi kehitystyössä.

4.2 Puikkari SDN -ohjain

Puikkari SDN -ohjainta ei ole kontitettu, joten sen asentaminen vaatii enemmän työtä. Loin VirtualBoxilla Ubuntu 18.04 virtuaalikoneen ja pakettiriippuvuuksien selvittelyjen jälkeen Puikkariakin saatiin asennettua testauskohteeksi.



```
antza@antza: ~/weftworks/test-bench/build - a dev ●+ . ./bin/weftworks-test-bench -c 84.250.89.163 -p 6653
[2020-01-15 18:10:43.171] [info]
[2020-01-15 18:10:43.171] [info] Weftworks SDN Test Bench - Copyright (C) 2019 Antti Lohtaja
[2020-01-15 18:10:43.171] [info] This program comes with ABSOLUTELY NO WARRANTY.
[2020-01-15 18:10:43.171] [info] This is free software, and you are welcome to redistribute
[2020-01-15 18:10:43.171] [info] it under certain conditions.
[2020-01-15 18:10:43.171] [info]
[2020-01-15 18:10:43.171] [info] Running tests..
[2020-01-15 18:10:43.171] [info] Testing for OpenFlow protocol v1.3..
[2020-01-15 18:10:43.172] [info] Connected to target controller.
[2020-01-15 18:10:43.172] [info] Running OpenFlow protocol v1.3 Hello test..
[2020-01-15 18:10:43.172] [info] Sending OpenFlow Hello..
[2020-01-15 18:10:43.172] [info] OpenFlow Hello sent.
[2020-01-15 18:10:43.172] [info] Waiting for response..
[2020-01-15 18:10:43.174] [info] Received a response!
[2020-01-15 18:10:43.174] [info] Target controller seems to only support OpenFlow v1.3.0 or earlier (No payload / HelloElements).
[2020-01-15 18:10:43.174] [info] OpenFlow v1.3 Hello passed!
[2020-01-15 18:10:43.174] [info] Running OpenFlow protocol v1.3 Handshake test..
[2020-01-15 18:10:43.174] [info] Waiting for FeatureReq..
[2020-01-15 18:10:43.174] [info] Received a response!
[2020-01-15 18:10:43.174] [info] Successfully received a FeatureReq message!
[2020-01-15 18:10:43.174] [info] Sending OpenFlow FeatureRes in response..
[2020-01-15 18:10:43.174] [info] OpenFlow FeatureRes sent.
[2020-01-15 18:10:43.175] [info] OpenFlow v1.3 Handshake passed!
[2020-01-15 18:10:43.175] [info] Running OpenFlow protocol v1.3 Errors test..
[2020-01-15 18:10:43.175] [info] Sending OpenFlow EchoRequest..
[2020-01-15 18:10:43.175] [info] OpenFlow EchoRequest sent.
[2020-01-15 18:10:43.175] [info] Waiting for response..
[2020-01-15 18:10:43.216] [info] Received a response!
[2020-01-15 18:10:43.216] [error] Received an invalid response (invalid header or message type).
[2020-01-15 18:10:43.216] [error] OpenFlow v1.3 Errors failed.
[2020-01-15 18:10:43.216] [error] One or more tests failed for OpenFlow protocol v1.3.
antza@antza: ~/weftworks/test-bench/build - a dev ●+ 
```

Kuvio 26. Tuloste Puikkari SDN-ohjaimen testauksesta

Puikkarin testaamisella havaittiin että tulos on täsmälleen sama (ks. kuvio 26.) Errors-testitilanne ei onnistunut myöskään Puikkarilla. Tästä voidaan päätellä, että Test Benchin Errors-testitilanteessa on todennäköisesti virhe. Otanta ei ole suuri ($n=2$), mutta virhe Test Benchissä on todennäköisempi, kuin että kahdessa testatussa SDN-ohjaimessa olisi sama virhe. Näin myös Test Benchin kehitystä varten saatiin tärkeää tietoa.

5 Tulosten pohdinta

5.1 Tulosten luotettavuus

Jotta voidaan olla varmoja että Test Benchin testaustuloksiin voidaan luottaa, täytyy myös itse Test Bench testata. Yksikkötestit on luotu tiedonsiirtoa käsitteleville komponenteille ja OpenFlow-viestien luokille. Opinnäytetyössä käytettävissä olevien resurssien vähyyden vuoksi Test Benchiin ei kuitenkaan luotu mitään integraatio- tai regressiotestejä. Testikattavuuden parantaminen on yksi jatkokehityksen kohde.

Lisäksi liitteessä 2 on ohjeet miten Test Benchin voi saada itselleen käyttöön ja toistaa opinnäytetyössä mainitut testitapaukset. Näin opinnäytetyön tulokset voi myös lukija itse varmistaa ja todentaa.

5.2 Saavutetut tavoitteet

Opinnäytetyön tuotoksena on tutkimus ja pohdinta siitä, miten SDN-ohjaimien järjestelmätestausta voitaisiin tehdä sekä Proof-of-Concept testaussovellus, joka toteuttaa opinnäytetyöhön valitut testaustilanteet.

Test Bench sovellus on opinnäytetyön tekijän mielestä hyvä pohja lähteä rakentamaan tällaista testaustyökalua. Se ei ole läheskään valmis, mutta toimii Proof-of-Conceptina. Edes yhden koko OpenFlow-protokollan version kaikkien testien laatiminen vaatii niin paljon aikaa ettei se ole opinnäytetyön resurssien puitteissa mahdollista.

Työn aikana opinnäytetyön kirjoittaja on oppinut paljon matalamman tason ohjelmoinnista, varsinkin muistin käsittelystä ja tiedonsiirrosta tietoverkoissa.

5.3 Jatkokehityskohteet

Tämänhetkinen toteutus kattaa vain pienen osan OpenFlow-protokollan versiosta 1.3. Ensimmäinen jatkokehityksen kohde on luoda kaikki muut mahdolliset testit 1.3.

versiosta. Sen jälkeen olemassa olevista testeistä voisi todennäköisesti vain hieman muokkaamalla luoda testit muille OpenFlow-protokollan versioille.

OpenFlow-protokollan testaamisen jälkeen voi harkita myös muiden SDN protokollien testaamista.

Test Bench itsessään voi esittää SDN-ohjaimelle vain yhtä OpenFlow-kytkintä kerrallaan verkkosovittimen IP- ja MAC-osoitteen takia. Jos Test Benchin saisi hajautettua useammalle tietokoneelle tai vain verkkosovittimelle, sillä voitaisiin simuloida kokonaista tietoliikenneverkkoa asiakaskoneineen. Tämä taas mahdollistaisi uuden tyyppisen testitapauksien luomisen, jossa on mukana useampi simuloitu OpenFlow-kytkin. Tätä aiheen mahdollisuuksia pitäisi tutkia enemmän.

Toiminnallisuuden testaamisen lisäksi Test Bench voisi mitata SDN-ohjainten suorituskykyä. Test Bench voisi mitata miten kauan SDN-ohjaimelta menee vastata kuhunkin pyyntöön ja lisätä ne tiedot raporttiin. Sen jälkeen Test Benchin käyttäjät voisivat käyttää sitä vaikkapa SDN-ohjaimen kehityksen apuna uusien muutosten suorituskykyvaikutusten mittaamiseen.

Test Benchin tuottamat raportit voisi tulostaa erilaisiin muotoihin. Test Bench tulostaa testaustulokset pääpiirteittäin terminaaliin, mutta tuloksista voitaisiin myös tuottaa kattavampia raportteja esimerkiksi PDF muodossa. Kattavammasta raportista voisi ilmetä listattuna kaikki ajatut testit, ajankohdat, ongelmat ja muut analyysit. Sitten nämä raportit voitaisiin automaattisesti lähettää eri paikkoihin. Test Benchin voitaisiin myös liittää paikallinen tietokanta, johon tulokset tallennetaan. Tämän jälkeen voitaisiin luoda mikropalvelu, joka lukee testidataa tietokannasta ja puolestaan raportoi sen jonnekin muualle, esimerkiksi Apache Kafka -palveluun. Tällöin Test Bench ohjelmaa voitaisiin ajaa säännöllisin väliajoin jotakin SDN-kontrolleria vastaan, ja tulokset voitaisiin käsitellä keskitetysti jossain muualla.

Näistä jatkokehityksen kohteista mielestäni kaikista tärkein on OpenFlow-version 1.3. testien loppuun tekeminen. Mielenkiintoisin kohde mielestäni on suorituskyvyn mittaaminen.

Lähteet

Byte Ordering. N.d. Artikkelitavujärjestyksestä keil.com verkkosivulla. Viitattu 19.1.2020. http://www.keil.com/support/man/docs/c51/c51_xe.htm

Caratelli, J. 2019. The forgotten art of Struct Packing in C/C++. Viitattu 26.10.2019. <http://www.joshcaratelli.com/blog/struct-packing>

Kohlhoff, C. N.d. Basic Boost.Asio Anatomy. Viitattu 19.1.2020. https://www.boost.org/doc/libs/1_72_0/doc/html/boost_asio/overview/core/basics.html

Laitila, T. 2017. Telia myy kiellettyä 4g-liittymää – rikkoo internetin tärkeää periaatetta. Tivi 27.11.2017. Viitattu 10.11.2019. <https://www.tivi.fi/uutiset/telia-myy-kiellettya-4g-liittymaa-rikkoo-internetin-tarkeaa-periaatetta/565f1ff0-df76-3ff6-a8c9-961db02a8e6d>

Liimatainen, E. 2017. Ohjelmoitava maailma – SDx-tuotekehitysympäristö pilvipalvelin-, SDN/NFV- ja IoT-teknologioin. Opinnäytetyö, AMK. Metropolia ammattikorkeakoulu, tietotekniikan koulutusohjelma. Viitattu 27.10.2019. <http://urn.fi/URN:NBN:fi:amk-201702011862>

Message layer. N.d. Flowgrammable.org verkkosivu. Viitattu 10.7.2019. <http://flowgrammable.org/sdn/openflow/message-layer/>

OpenFlow Switch Specification. 2012. OpenFlow-protokollan spesifikaatio. Viitattu 19.1.2020. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf>

Ott, A. 2011. What is Boost.Asio, and why we should use it. Viitattu 9.7.2019. <http://alexott.net/en/cpp/BoostAsioNotes.html>

Our Members. N.d. Open Networking Foundation -säätiön verkkosivu. Viitattu 19.1.2020. <https://www.opennetworking.org/member-listing/>

Our Mission. N.d. Open Networking Foundation -säätiön verkkosivu. Viitattu 19.1.2020. <https://www.opennetworking.org/mission/>

Rash, Wayne. 2018. SDN Is the Future, Plan Your Migration Now. Artikkelitavujärjestyksestä PCMag UK:n verkkosivulla. Viitattu 19.4.2020. <https://uk.pcmag.com/it-watch/116241/sdn-is-the-future-plan-your-migration-now>

std::exit. 2019. cppreference.com verkkosivu. Viitattu 1.3.2020. <https://en.cppreference.com/w/cpp/utility/program/exit>

Stdout, 2019. Computer Hope artikkeli. Viitattu 1.3.2020. <https://www.computerhope.com/jargon/s/stdout.htm>

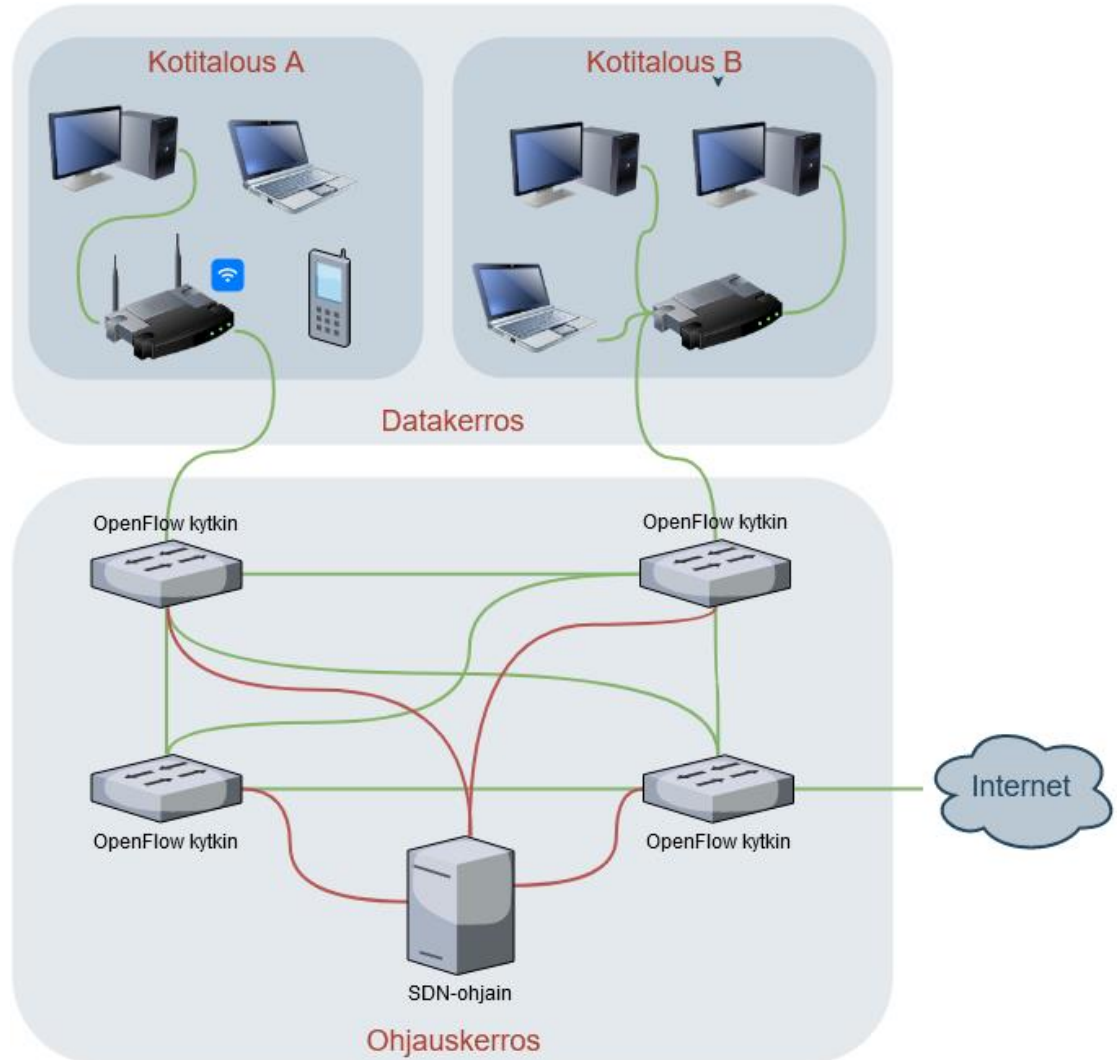
TCP/IP Protocol Architecture Model. 2010. Viitattu 5.10.2019. <https://docs.oracle.com/cd/E19455-01/806-0916/ipov-10/index.html>

What is SDN? N.d. ciena.com-artikkeli. Viitattu 26.10.2019. <https://www.ciena.com/insights/what-is/What-Is-SDN.html>

Zurkus, K. 2017. SDN solves a lot of network problems, but security isn't one of them. Artikkel SDN'n turvallisuudesta. Viitattu 10.11.2019. <https://www.csoonline.com/article/3179637/sdn-solves-a-lot-of-network-problems-but-security-isnt-one-of-them.html>

Liitteet

Liite 1. Esimerkki SDN-ohjatusta tietoverkosta



Liite 2. Test Benchin käyttöönotto ja tulosten toistaminen

Test Benchiiä on kehitetty ainoastaan Linux alustalla. Test Benchin ”kääntämiseen” tarvitset myös erinäisiä työkaluja. Lisäksi käyttäjäsi täytyy olla ns. sudoer, jotta voit asentaa ohjelmia.

Vaatimukset:

- Git
- CMake, versio 3.13. tai uudempi
- Clang Compiler, versio 8.0 tai uudempi
- libc++ -kirjasto
- Boost -kirjastot, versio 1.70 tai uudempi
- Vaihtoehtoisesti Catch2, versio 2.6.0 tai uudempi (omien testien ajamiseen)

CMake’n asennus

```
wget
https://github.com/Kitware/CMake/releases/download/v3.13.4/cm
ake-3.13.4.tar.gz
tar xf cmake-3.13.4.tar.gz
cd cmake-3.13.4/
./configure
sudo make install
```

Clang ja libc++ -kirjasto pitäisi löytyä useimpia Linux distribuutioiden pakettien hallinnasta, esim:

```
sudo apt install clang libc++-dev
```

Boost -kirjastot pitää asentaa käyttäen Clang Compileria.

```
cd /tmp # optional
wget
https://dl.bintray.com/boostorg/release/1.70.0/source/boost_1
_70_0.tar.gz
tar xf boost_1_70_0.tar.gz
cd boost_1_70_0/
./bootstrap.sh --with-toolset=clang
./b2 clean
./b2 toolset=clang cxxflags="-stdlib=libc++" linkflags="-
stdlib=libc++"
sudo ./b2 install
```

Catch2, vaihtoehtoinen:

```
git clone https://github.com/catchorg/Catch2.git
cd Catch2/
cmake -Bbuild -H. -DBUILD_TESTING=OFF
sudo cmake --build build/ --target install
```

Test Bench on riippuvainen Weftworks projektin Common Library'stä. Common Library'n asennus:

```
git clone https://gitlab.com/weftworks/common.git
cd common/
mkdir build
cd build/
cmake -DWWCL_ENABLE_TESTING=OFF ..
sudo make install
```

Jos haluat kääntää Catch2 testit, muuta WWCL_ENABLE_TESTING arvoksi ON.

Test Benchin asennus ja ajaminen:

```
git clone https://gitlab.com/weftworks/test-bench.git
cd test-bench/
mkdir build
cd build/
cmake -DWWTB_ENABLE_TESTING=OFF ..
make
./bin/weftworks-test-bench --help
./bin/weftworks-test-bench -c <host> -p <port>
```