



# RESTCONF-kanavan WebSocket-yhteyden käyttö palvelinohjelmiston muutoksien dynaamiseen päivitykseen selaimessa

Teemu Loijas

OPINNÄYTETYÖ  
Huhtikuu 2020

Tieto- ja viestintätekniikka  
Ohjelmistotekniikka

## TIIVISTELMÄ

Tampereen ammattikorkeakoulu  
Tieto- ja viestintätekniikka  
Ohjelmistotekniikka

LOIJAS, TEEMU:

RESTCONF-kanavan WebSocket-yhteyden käyttö palvelinohjelmiston muutoksien dynaamiseen päivitykseen selaimessa

Opinnäytetyö 31 sivua, joista liitteitä 3 sivua  
Huhtikuu 2020

---

Opinnäytetyössä toteutettiin WebSocket-yhteys RESTCONF-kanavan kautta palvelinohjelmiston muutoksien dynaamista päivitystä varten selaimessa. Edeltävässä toteutuksessa ei oltu hyödynnetty WebSocket-yhteyttä, minkä takia tietojen päivitys selaimessa saattoi kestää huomattavan kauan. Työn tarkoituksena on tehdä reaaliaikaiseksi palvelinohjelmiston laitedatan muutoksia selaimessa ja samalla kerryttää asiantuntemusta Software Defined Networkingista. Työssä hyödynnettiin nykyaikaisia verkkoarkkitehtuurimenetelmiä, jotka mahdollistavat verkon älykkään ja keskitetyn hallinnan tai ohjelmoinnin käyttämällä ohjelmistoversioita.

Työ koostuu kahdesta osasta, jotka ovat Reactilla toteutettu web-käyttöliittymä sekä Java-pohjainen palvelinpuoli. Työssä toteutettiin halutut ominaisuudet, vaikka ongelmia tulikin vastaan. Toteutuksessa tarkoituksena oli hyödyntää Lighty.io-alustaa, joka jouduttiin vaihtamaan OpenDaylightiin Lightyssa esiintyneen ohjelmointivirheen takia. Alustan vaihtaminen ei estänyt ominaisuuksien toteuttamista, vaikka hieman sitä hidastikin. Työssä toteutetut ominaisuudet eivät suoraan siirtyneet tuotantoympäristöön, mutta tämä on hyvin mahdollista tulevaisuudessa, kun sovelluksen jatkokehittäminen aloitetaan. Ominaisuuksien siirtäminen tuotantoon onnistunee sujuvasti, sillä toteutetut ominaisuudet on testattu kattavasti.

Opinnäytetyössä käsitellään edellä mainittujen web-käyttöliittymän ja palvelinohjelmiston päivitysten lisäksi yksinkertaisen OpenDaylight-sovelluksen luominen. Ohjeistuksen avulla kuka tahansa voi luoda oman yksinkertaisen OpenDaylight-sovelluksen.

WebSocket-yhteyden avulla parannetaan käyttöliittymän luotettavuutta merkittävästi. Verkkolaitteiden tilasta saadaan tietoa lähes reaaliajassa. Nykyaikaisten verkkoarkkitehtuurimenetelmien avulla laitteiden sekä verkkojen älykkyys saadaan nostetuksi kokonaan uudelle tasolle. Verkkojen ohjelmitavuus helpottaa huomattavasti ja tarvittavat verkko-ominaisuudet voidaan julkaista entistä nopeammin.

---

Asiasanat: sdn, websocket, opendaylight

## ABSTRACT

Tampereen ammattikorkeakoulu  
Tampere University of Applied Sciences  
ICT Engineering  
Software Engineering

LOIJAS, TEEMU:

Using the RESTCONF WebSocket Connection to Dynamically Update Server Changes in Browser

Bachelor's thesis 31 pages, appendices 3 pages  
April 2020

---

The thesis carried out a WebSocket connection over the RESTCONF channel for dynamic updating of server head changes in the browser. The previous implementation did not utilize a WebSocket connection, which could take a considerable amount of time to refresh the data in the browser. The purpose of this work is to speed up the updating of server head changes in the browser. The thesis also provided adequate knowledge of SDN. Thus, the work utilized a network architecture method called SDN. SDN enables intelligent and centralized network management or programming using software applications.

The work consists of two parts, React implemented web interface and Java based server side. The work achieved the desired properties, even though problems were encountered. The purpose of the implementation was to use the Lighty.io platform, which had to be upgraded to OpenDaylight due to a bug in Lighty. Changing the substrate did not prevent the realization of the features, although it slightly slowed it down. Nevertheless, the work was carried out on schedule. The features implemented in the work did not directly migrate to the production environment, but it is very possible in the future as the application is being developed further. The transfer of features to production should be successful, as the implemented features have been extensively tested.

In addition to web interface and server side updated, the thesis deals with the creation of a simple OpenDaylight application. At the time of writing this thesis, with the help of this tutorial, anyone can create their own simple OpenDaylight application.

The WebSocket connection significantly improves the reliability of the user interface. Real time data of network devices is provided. With the modern network architecture SDN, the intelligence of hardware and software can be taken to a whole new level. The programmability of networks is greatly facilitated, and the necessary network features can be released more quickly.

---

Key words: sdn, websocket, opendaylight

## SISÄLLYS

1	JOHDANTO .....	7
2	KÄYTETYT TEKNOLOGIAT .....	8
2.1	WebSocket.....	8
2.2	OpenDaylight .....	9
2.3	Lighty.io.....	11
2.4	Java .....	11
2.5	YANG.....	12
2.6	NETCONF.....	12
2.7	RESTCONF .....	13
2.8	JavaScript .....	13
2.9	React.....	13
2.10	Software Defined Networking .....	14
2.10.1	Verkon ohjelmoitavuus.....	14
2.10.2	Looginen älyn ja kontrolloinnin keskittäminen .....	14
2.10.3	Verkon abstrahointi .....	15
2.10.4	Avoimuus .....	15
3	OPENDAYLIGHTIIN TUTUSTUMINEN .....	16
3.1	Tutustuminen .....	16
3.2	Edellytykset OpenDaylight-sovelluksen luomiseen .....	16
3.3	Hello-moduulin rakentaminen .....	16
3.4	RPC:n lisääminen YANG-malliin .....	17
3.5	Hello world -RPC:n implementointi.....	18
3.6	Hello world RPC:n testaaminen RESTin kautta .....	19
3.7	Ilmoitusten lisääminen.....	20
3.8	Ilmoitusten käsittely virtojen avulla käyttöliittymässä .....	21
4	PALVELINOHJELMISTO .....	24
4.1	Ohjelmistokehityksen valinta.....	24
4.2	Toiminta .....	24
5	KÄYTTÖLIITTYMÄ .....	26
5.1	Toiminta .....	26
5.2	Kutsut.....	26
6	POHDINTA .....	27
	LÄHTEET.....	28
	LIITTEET .....	29
	Liite 1. hello.yang .....	29
	Liite 2. HelloProvider.java .....	30

Liite 3. HelloWorldImpl.java .....	31
------------------------------------	----

**LYHENTEET JA TERMIT**

CORS	Cross Origin Resource Sharing, Mekanismi HTTP-kutsujen hallitsemiseksi
CRUD	Create, read, update and delete, Jatkuvan tallennuksen perustoiminnot
HTTP	Hypertext Transfer Protocol, Selainten ja WWW-palvelinten tiedonsiirtoon hyödyntämä protokolla
IETF	The Internet Engineering Task Force, Internet-protokollien standardoinnista vastaava organisaatio
JSON	JavaScript Object Notation, Avoimen standardin tiedostomuoto
MD-SAL	Model-Driven Service Abstraction Layer, Komponentti viestien välitykseen ja tiedon tallentamiseen
ODL	OpenDaylight, Alusta ohjelmistopohjaisille verkkoteknologioille
OSGi	Open Services Gateway initiative, Dynaaminen moduulijärjestelmä Javalle
RPC	Remote Procedure Call, Etäproseduurikutsu
SDN	Software defined networking, Ohjelmistopohjainen verkkoteknologia
SNMP	Simple Network Management Protocol, Verkkojen hallinnassa käytettävä tietoliikenneprotokolla
SPA	Single Page Application, Yhden sivun sovellus
TCP	Transmission Control Protocol, Tietoliikenneprotokolla luotettavaan tiedonsiirtoon
URI	Uniform Resource Identifier, Merkkijono tiedon paikan tai nimen kertomiseen
XML	Extensible Markup Language, Formaatti sekä tiedostomuoto

## 1 JOHDANTO

Opinnäytetyössä toteutettiin käyttöliittymän ja palvelinohjelmiston välinen WebSocket-yhteys, joka mahdollistaa palvelinohjelmiston muutosten dynaamisen päivityksen käyttöliittymässä. Käytännössä palvelin lähettää ilmoituksia web-käyttöliittymälle, joka vastaanottaa ilmoitukset reaaliajassa. Työ tehtiin yhteistyössä Insta DefSec Oy:n kanssa.

Työssä tutustuttiin laajalti Software Defined Networkingiin sekä sitä hyödyntäviin sovelluksiin. Aluksi rakennettiin esimerkkisovellus käyttäen OpenDaylightia, jonka jälkeen tarkoituksena oli muuntaa sama sovellus ajettavaksi Lightyllä. Lightyssa huomattiin kuitenkin huomattava ohjelmointivirhe, jonka takia WebSocket-yhteyttä ei pystytty muodostamaan. Tämän vuoksi palvelimen ohjelmisto päätettiin tehdä OpenDaylightilla.

Käyttöliittymätoteutuksessa hyödynnettiin Reactia, joka on nykyaikainen JavaScript-kirjasto käyttöliittymien rakentamiseen. React mahdollistaa WebSocket-yhteyden sujuvan luomisen. Työn pääpaino oli kuitenkin jo aiemmin mainitun SDN:n ja Java-pohjaisen palvelinohjelmiston tutkimisessa sekä WebSocket-yhteyden mahdollistavien muutoksien tekemisessä.

Käyttöliittymän sekä palvelinohjelmiston muutokset tehtiin aikataulun mukaisesti. Palvelinohjelmiston muuttaminen vei huomattavasti enemmän aikaa kuin käyttöliittymän muuttaminen.

Työssä esitellään tarvittut teknologiat, sovellukset sekä ohjelmointikielet. Edellä mainittujen lisäksi esitellään myös yksinkertaisen OpenDaylight-sovelluksen luominen sekä siihen tarvittavat teknologiat ja riippuvuudet. Lopuksi esitellään käyttöliittymän sekä palvelinohjelmiston muutokset ja pohditaan työtä, sen tarkoitusta ja siitä suoriutumista.

## 2 KÄYTETYT TEKNOLOGIAT

Tässä luvussa esitellään keskeisimmät opinnäytetyössä käytetyt teknologiat, joita ovat WebSocket, OpenDaylight, Lighty.io, Java, YANG, NETCONF, RESTCONF, JavaScript, React ja SDN.

### 2.1 WebSocket

WebSocket on tietokoneviestintäprotokolla, joka mahdollistaa kaksisuuntaisen kommunikoinnin yhden TCP-yhteyden avulla (Internet Engineering Task Force (IETF), 12/2011). Toisin sanoen WebSocket mahdollistaa reaaliaikaisen viestinnän selaimen tai nettisovelluksen ja palvelimen välillä. Tämän tekniikan tarkoituksena on tarjota ratkaisu selainpohjaisille sovelluksille, jotka tarvitsevat kaksisuuntaista viestintää palvelimien kanssa. WebSocketin avulla voidaan korvata esimerkiksi HTTP-kyselyt, joissa selain tai nettisovellus joutuu lähettämään palvelimelle kyselyn tarkoittaen sitä, että selain tai nettisovellus joutuu odottamaan palvelimen vastausta.

WebSocket on itsenäinen TCP-pohjainen protokolla. Sen ainoa yhteys HTTP:hen on se, että HTTP-palvelimet tulkitsevat kättelyt päivityspyyntöinä. Oletuksena WebSocket käyttää porttia 80. Suojattuja yhteyksiä varten WebSocket käyttää porttia 443. Oletusyhteyksissä URI:t ovat muotoa "ws://host:port/path?query" ja salatuissa yhteyksissä ne ovat muotoa "wss://host:port/path/query".

Protokollalla on kaksi vaihetta, jotka ovat kättely sekä datan siirto. Asiakas aloittaa kättelyn lähettämällä viestin palvelimelle, johon palvelin tämän jälkeen vastaa. Jos kättely onnistuu, voidaan dataa alkaa siirtää asiakkaan ja palvelimen välillä.

WebSocket-yhteyden avaaminen on todella yksinkertaista. Yhteyden avaamisessa voidaan hyödyntää WebSocket-ohjelmointirajapintaa. Yhteys avataan luomalla uusi WebSocket-olio, jolle annetaan parametreiksi WebSocket-URI sekä mahdolliset protokollat (kuva 1).



WebSocket-yhteyden avaamisen yhteydessä voidaan esimerkiksi lähettää palvelimelle viesti käyttämällä WebSocketin tapahtumaa "open". Alla olevan kuvan koodi lähettää asiakkaalta palvelimelle viestin "Hello Server!" yhteyden avautuessa.

Suurin osa nykyaikaisista selaimista tukee WebSocket-ohjelmointirajapintaa, joka mahdollistaa sen laaja-alaisen käytön.

```
1 // Create WebSocket connection.
2 const socket = new WebSocket('ws://localhost:8080');
3
4 // Connection opened
5 socket.addEventListener('open', function (event) {
6     socket.send('Hello Server!');
7 });
8
9 // Listen for messages
10 socket.addEventListener('message', function (event) {
11     console.log('Message from server ', event.data);
12 });
```

KUVA 1. WebSocket-yhteyden avaaminen ja käyttäminen (MDN, 2019)

## 2.2 OpenDaylight

OpenDaylight on modulaarinen ja avoin alusta, jonka avulla voidaan muokata sekä automatisoida verkkoja niiden koosta riippumatta. Kyseinen projekti syntyi SDN:n kasvusta keskittyen selkeästi verkon ohjelmoitavuuteen. Se on alusta alkaen suunniteltu perustaksi kaupallisille ratkaisuille, jotka käsittelevät erilaisia käyttötapauksia olemassa olevissa verkkoympäristöissä. ODL on laajimmin käytetty avoimen lähdekoodin SDN-alusta. (OpenDaylight, n.d, Platform Overview)

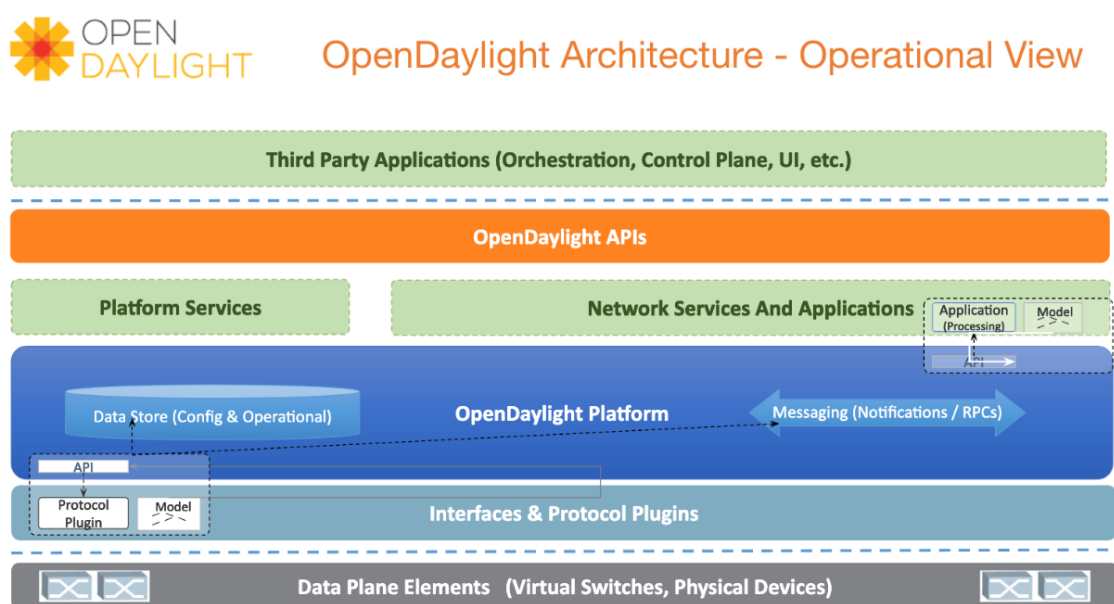
ODL-alustan ydin on MD-SAL (eli Model-Driven Service Abstraction Layer). OpenDaylight-sovelluksen taustalla olevat verkkolaitteet ja -sovellukset on esitetty malleina tai objekteina, joiden vaikutukset käsitellään SAL:in sisällä.

SAL on tiedonvaihto- ja mukautusmekanismi YANG-mallien (ks. 3.4.) välillä, jotka kuvaavat verkkolaitteita ja sovelluksia.

ODL on suunniteltu antamaan käyttäjille maksimaalinen joustavuus rakentaa ohjain heidän tarpeisiinsa. ODL:n modulaarinen arkkitehtuuri myös mahdollistaa kehittäjälle tai käyttäjälle myös muiden kehittäjien luomien palveluiden hyödyntämisen (kuva 2). ODL tarjoaa tuen laajalle protokollajoukolle kaikissa SDN-alus-  
toissa.

ODL jakaa komponentit Karaf-ominaisuuksiksi. Näin varmistetaan, että uusi ominaisuus ei pääse häiritsemään jo tehtyjä sekä testattuja ominaisuuksia. ODL käyttää OSGi:a ja Mavenia rakentamaan paketin, joka hallitsee edellä mainittuja Karaf-ominaisuuksia ja niiden vuorovaikutuksia.

ODL mahdollistaa myös yksikkö- ja integraatiotestauksen sovelluksen kääntämisen yhteydessä. Työn tuottavuus kasvaa huomattavasti, kun kehittäjän ei tarvitse itse alkaa etsiä koodivirheitä. Kääntämisen yhteydessä kehittäjä voi halutessaan myös ajaa niin sanotun linterin eli tarkastella koodin tyyllillisiä seikkoja. Testien ajaminen tuo mukanaan myös huonon puolen, sillä testien ajaminen kestää huomattavan kauan. Tämä siis hidastaa sovelluksen kääntämistä.



Kuva 2. OpenDaylightin arkkitehtuuri (OpenDaylight, n.d., Current release)

## 2.3 Lighty.io

Lighty.io on ohjelmistokehityspaketti, joka sisältää osan OpenDaylightin komponenteista. Lighty.io toimii nopeammin ja pienemmillä resursseilla OpenDaylightiin verrattuna. Tämä johtuu nimenomaan siitä, että ylimääräiset komponentit on karsittu pois.

Suurimpana erona Lightyn ja OpenDaylightin välillä voidaan mainita Karafin poistaminen. OpenDaylightissa Karafilla käynnistetään ominaisuudet ja komponentit, kun taas Lightyssa siihen voidaan käyttää mitä tahansa ohjelmistokehystä.

Lighty.io kuitenkin tarjoaa päivitettyjä ja paranneltuja versioita osasta OpenDaylightin komponentteja (PANTHEON.tech, 2020). Lightyn dokumentaatio on hyvinkin vähäistä, joka tekee Lightyn kanssa työskentelystä hieman hankalaa ja hidasta. Lighty-sovelluksen kehittäminen on hyvinkin samanlaista kuin ODL-sovelluksen. Ongelmaksi kuitenkin muodostuu se, että niissä on pieniä eroja, joita ei mainita välttämättä missään.

## 2.4 Java

Java on yleiskäyttöinen ohjelmointikieli, joka on samaan aikaan luokkaperusteinen, oliokeskeinen ja suunniteltu erityisesti niin, että siinä on mahdollisimman vähän toteutukseen liittyviä riippuvuuksia. Javan tarkoituksena on tarjota sovelluskehittäjälle mahdollisuus suorittaa koodia kaikilla Java-järjestelmiä tukevilla alustoilla ilman, että tarvitaan uudelleenkäntämistä.

Esimerkkinä tästä Java-ohjelma voidaan kirjoittaa ja kääntää Windowsilla, jonka jälkeen ohjelmaa voidaan suoraan suorittaa UNIX-koneessa. Tämä mahdollistetaan kääntämällä Java-ohjelma niin sanotuksi välikieleksi, jota kutsutaan tavukoodiksi. Tavukoodin muoto on alustasta riippumaton. Tavukoodin suorittamisessa jokaisella alustalla käytetään Java Virtual Machinea. (HowToDoInJava, 2016)

Javan alkuperäinen kehittäjä on James Gosling ja se on julkaistu vuonna 1995 Sun Microsystemsin Java-alustan ydinkomponenttina. Kielen syntaksi johtaa juurensa suurilta osin C- ja C++ -ohjelmointikielistä.

## 2.5 YANG

YANG on datan mallinnuksessa käytetty ohjelmointikieli. YANG:n avulla mallinetaan lähinnä konfiguraatio- ja tiladataa, jota lähetetään verkkojen hallintaan liittyvien protokollien avulla kuten NETCONF ja RESTCONF (Internet Engineering Task Force (IETF), 2010). Konfiguraatio- ja tiladatan lähettämisen lisäksi YANG:ia hyödynnetään mallintamaan Remote Procedure Callit ja ilmoitukset. Tämä mahdollistaa täydellisen kuvauksen kaikesta datasta, jota lähetetään NETCONF asiakkaan ja palvelimen välillä.

YANG mahdollistaa selkeät ja tiiviit kuvaukset jokaiselle oksalle ja oksien väliselle vuorovaikutukselle. Jokaiselle oksalle asetetaan nimi ja joko arvo tai joukko lapsioksia (kuva 3). YANG sisältää sisään rakennettuja tyyppejä, jonka lisäksi se sisältää mekanismin, jolla voidaan määritellä uusia tyyppejä.

```
leaf host-name {  
    type string;  
    description "Hostname for this system";  
}
```

KUVA 3. Yang-oksa (Internet Engineering Task Force (IETF), 10/2010)

## 2.6 NETCONF

NETCONF (Network Configuration Protocol) on verkonhallintaprotokolla, jonka on kehittänyt ja standardoinut IETF. NETCONF mahdollistaa mekanismit asentaa, muokata sekä poistaa verkkolaitteiden konfiguraatiot. Nämä edellä mainitut operaatiot realisoituvat RPC-kerroksella. Protokolla käyttää sekä XML- että JSON-enkoodausta konfiguraatiodatalle sekä protokollaviesteille. Protokollaviestit vaihdetaan suojatun kuljetusprotokollan avulla. (Internet Engineering Task Force (IETF), 2011)

## 2.7 RESTCONF

RESTCONF on suunniteltu muokkaamaan YANG:ssa määriteltyä dataa hyödyntäen DATASTORE-konsepteja, jotka on määritelty NETCONF:ssa. RESTCONF käyttää HTTP-metodeja mahdollistaakseen CRUD-operaatiot datastoreen, joka sisältää YANG:lla määriteltyä dataa, joka on yhteensopivaa palvelimen kanssa, joka implementoi NETCONF-datastoreja. (Internet Engineering Task Force (IETF), 2017)

## 2.8 JavaScript

JavaScript on kevyt ja tulkattava ohjelmointikieli, joka tunnetaan lähinnä nettisivuista sekä selainsovelluksista. Näiden lisäksi JavaScriptia hyödyntävät myös ei-selainpohjaiset sovellukset kuten Node.js, Apache CouchDB ja Adobe Acrobat. (Mozilla, 2019)

JavaScriptia ei tule sekoittaa Java-ohjelmointikielen kanssa. Vaikkakin edellä mainittujen ohjelmointikielten nimet ovat samankaltaiset, eroavat ne kuitenkin huomattavasti syntaksin ja käyttökohteiden osalta. JavaScript kuitenkin sisältää kirjoitusasussaan huomattavasti samankaltaisuuksia Javan kanssa. Molemmat ohjelmointikielet sisältävät esimerkiksi if- ja for-lauseet sekä try-catch-lauseen.

## 2.9 React

React on JavaScript-kirjasto käyttöliittymien rakentamiseen. Toisin sanoen Reactin avulla rakennetaan JavaScriptiin ohjaamia web-sovelluksia. React-sovellukset toimivat selaimessa eivätkä palvelimella. Tämä tarkoittaa sitä, että käyttäjän tekemät asiat tapahtuvat välittömästi, koska käyttäjän ei tarvitse odottaa palvelimen vastausta.

React hyödyntää komponentteja käyttöliittymien rakentamisessa. Käyttöliittymä voidaan siis jakaa eri komponentteihin, joka tekee koodin ylläpitämisestä sekä kehittämisestä huomattavasti helpompaa.

Reactin avulla voidaan luoda niin sanottuja yhden sivun sovelluksia (SPA) tai sivustoja, jotka toimivat vuorovaikutuksessa verkkoselaimen kanssa kirjoittamalla nykyisen verkkosivun päälle dynaamisesti uudella web-palvelimen tiedolla sen sijaan, että selain lataisi kokonaan uudet sivut. Tämän avulla voidaan saavuttaa muun muassa nopeammat siirtymät eri näkymien välillä ja käyttäjälle mahdollistetaan natiivin sovelluksen tunne.

## **2.10 Software Defined Networking**

SDN on verkkoarkkitehtuurimenetelmä, joka mahdollistaa verkon älykkään ja keskitetyn hallinnan tai ohjelmoinnin käyttämällä ohjelmistosovelluksia. Tämä auttaa hallitsemaan koko verkkoa johdonmukaisesti ja kokonaisvaltaisesti riippumatta taustalla olevasta verkkoteknologiasta. (Ciena, n.d.)

SDN sisältää neljä kriittistä aluetta. Nämä alueet ovat verkon ohjelmoitavuus, looginen älyn ja kontrolloinnin keskittäminen, verkon abstrahointi sekä avoimuus.

### **2.10.1 Verkon ohjelmoitavuus**

SDN mahdollistaa verkon käyttäytymisen kontrolloinnin sovellusten avulla. Tämä mahdollistaa verkkojen räätälöinnin uusien palveluiden ja jopa yksittäisten käyttäjien tukemiseksi. Eriyttämällä ohjelmistot laitteista mahdollistetaan uusien palveluiden nopea käyttöönotto.

### **2.10.2 Looginen älyn ja kontrolloinnin keskittäminen**

SDN on rakennettu loogisesti keskitettyihin verkkotopologioihin, jotka mahdollistavat verkkoresurssien älykkään kontrolloinnin ja hallinnan. Keskitetyn hallinnan avulla mahdollistetaan erittäin älykäs verkon nopeuden hallinta, palauttaminen,

turvallisuus ja käytännöt. Edellä mainittujen asioiden ansioista SDN:ia hyödyntävä käyttäjä tai organisaatio saa kokonaisvaltaisen kuvan hallinnoitavasta verkosta.

### 2.10.3 Verkon abstrahointi

Verkon abstrahoinnilla tarkoitetaan sitä, että palvelut ja sovellukset, jotka toimivat SDN-teknologialla, erotetaan muista teknologioista sekä laitteista. Sovellukset ovat vuorovaikutuksessa verkon kanssa ohjelmistorajapintojen kautta sen sijaan, että hallintarajapinnat olisivat kytketty tiukasti laitteistoon.

### 2.10.4 Avoimuus

SDN-arkkitehtuurit ovat aloittaneet uuden avoimuuden aikakauden mahdollistaen useiden toimittajien laitteiden yhteentoimivuuden ja edistävät myyjä-neutraalia ekosysteemiä. Avoimuus tulee itse SDN:n lähestymistavasta. Avoimet ohjelmointirajapinnat tukevat monenlaisia sovelluksia. Tämän lisäksi älykkäät ohjelmistot voivat hallita monen eri valmistajan laitteita ohjelmoitavien rajapintojen kautta. Älykkäät verkkopalvelut ja sovellukset voivat toimia SDN:n sisällä yhteisessä ohjelmistoympäristössä.

### 3 OPENDAYLIGHTIIN TUTUSTUMINEN

Tässä kappaleessa esitellään yksinkertaisen OpenDaylight-sovelluksen rakentaminen. Sovelluksen rakentamisessa on hyödynnetty OpenDaylightin omia esimerkkejä. Oletuksena on, että käyttöjärjestelmä on Linux-pohjainen.

#### 3.1 Tutustuminen

OpenDaylightiin tutustumiseen käytettiin huomattavan paljon aikaa. Valmiina olleen toteutuksen kimppuun hyppääminen olisi varmasti tuottanut suuria vaikeuksia sen monimutkaisuuden vuoksi. Valmis toteutus oli siis toteutettu Lightyllä, jonka hallitsemiseksi OpenDaylight-osaaminen on lähes välttämätöntä.

Työn alussa tuotettiin yksinkertainen sovellus, jonka avulla ymmärrystä saatiin kartutettua. Edellä mainittu sovellus päätettiin toteuttaa hello world –tyyppisenä sovelluksena.

#### 3.2 Edellytykset OpenDaylight-sovelluksen luomiseen

OpenDaylight-sovelluksen rakentamista varten on muutamia edellytyksiä. Sovelluksen rakentamiseen tarvitaan Maven 3.5.2 tai uudempi versio, Java 8 yhteensopiva JDK sekä sopiva Maven settings.xml-tiedosto. Sopivan settings.xml-tiedoston voi ladata suoraan netistä ja siirtää .m2-kansioon (kuva 4).

```
cp -n ~/.m2/settings.xml{,.orig} ; wget -q -O -  
https://raw.githubusercontent.com/opendaylight/odlparent/master/settings.xml > ~/.m2/settings.xml
```

KUVA 4. Settings.xml-tiedoston lataaminen ja siirtäminen oikeaan sijaintiin

#### 3.3 Hello-moduulin rakentaminen

Aluksi luodaan esimerkkiprojekti alla olevalla koodilla käyttäen Mavenia sekä arkkityyppiä 'opendaylight-startup-archetype'. Arkkityypin versio riippuu kyseisellä



hetkellä voimassa olevan julkaisun versiosta. Työn kirjoitushetkellä käytössä oli arkkityyppi 1.2.0 (kuva 5).

```
mvn archetype:generate -DarchetypeGroupId=org.opendaylight.archetypes
-DarchetypeArtifactId=opendaylight-startup-archetype \
-DarchetypeCatalog=remote -DarchetypeVersion=1.2.0
```

#### KUVA 5. Projektin luomiseen käytettävä komento

Tämän jälkeen asennus kysyy projektin ominaisuuksia. Tässä tapauksessa projektin nimeksi annettiin hello (kuva 6).

```
Define value for property 'groupId': org.opendaylight.hello
Define value for property 'artifactId': hello
Define value for property 'version': 1.0-SNAPSHOT
Define value for property 'package': org.opendaylight.hello
Define value for property 'classPrefix': ${artifactId.substring(0,1)
.toUpperCase()}${artifactId.substring(1)}
Define value for property 'copyright': Copyright (c) 2020 Teemu Loijas
```

#### KUVA 6. Asennuksen ominaisuudet

Asennuksen valmistuttua voidaan projekti rakentaa projektin juurikansiossa komennolla "mvn clean install."

Projektin rakentaminen ensimmäisen kerran voi kestää huomattavan kauan. Tämän jälkeen voidaan käynnistää projekti juurikansiosta komennolla "./karaf/target/assembly/bin/karaf".

### 3.4 RPC:n lisääminen YANG-malliin

RPC lisätään YANG-malliin päivittämällä yang-tiedostoa. Tiedoston nimi on projektin nimen mukainen. Tässä tapauksessa, kun projektin nimi on hello, on myös yang-tiedoston nimi hello (kuva 7).

```
api/src/main/yang/hello.yang
```

#### Kuva 7. YANG-mallin tiedostosijainti

Muokataan tiedoston sisältö liitteen 1 mukaiseksi. Tässä YANG-malliin lisätään hello-world-niminen RPC, jolle annetaan syötteenä tyyppiä string oleva nimi,

sekä tyyppiä uint32 oleva numero. Näitä vastaan RPC palauttaa tyyppiä string olevan tervehdyksen. YANG-mallissa ei kuitenkaan määritellä tervehdyksen sisältöä, vaan sisältö määritetään seuraavassa kappaleessa käsiteltävässä java-tiedostossa.

Edellä mainitun tiedoston muokkaamisen jälkeen tallennetaan tiedosto ja siirrytään api-kansioon, jonka jälkeen API voidaan rakentaa komennolla "mvn clean install".

### 3.5 Hello world -RPC:n implementointi

Määritellään HelloService, joka herätetään hello-world-ohjelmointirajapinnan kautta. Luodaan HelloWorldImpl.java-tiedosto (liite 3), johon lisätään alla oleva koodi (kuva 8).

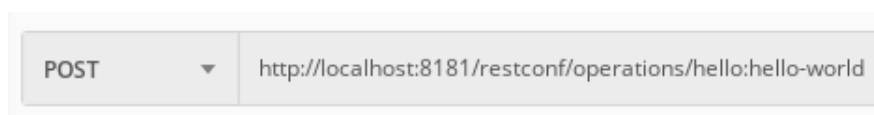
```
public class HelloWorldImpl implements HelloService {
    @Override
    public ListenableFuture<RpcResult<HelloWorldOutput>> helloWorld(HelloWorldInput input) {
        HelloWorldOutputBuilder helloBuilder = new HelloWorldOutputBuilder();
        helloBuilder.setGreeting("Hello " + input.getName() + " " + input.getNumber1());
        return RpcResultBuilder.success(helloBuilder.build()).buildFuture();
    }
}
```

KUVA 8. Hello world -RPC

Tämän jälkeen samassa kansiossa olevaan HelloProvider.java-tiedostoon rekisteröidään hello.yang-tiedostoon luotu RPC (liite 1). Tämän jälkeen voidaan rakentaa edellä luodut Java-luokat. Näin voidaan testata niihin tehdyt muutokset. Kun alla oleva käsky ajetaan impl-kansiossa, rakennetaan pelkästään impl-kansiossa olevat tiedostot. Java-luokkien kääntämiseen käytetään jo tuttua komentoa "mvn clean install". Rakennetaan koko projekti, jonka jälkeen projekti on valmis ajettavaksi komennolla "./karaf/target/assembly/bin/karaf" projektin juurikansiossa oltaessa.

### 3.6 Hello world RPC:n testaaminen RESTin kautta

RPC:n testaamisessa voidaan hyödyntää esimerkiksi Postmania, jonka avulla HTTP-kutsujen luominen on yksinkertaista. Alasvetovalikosta voidaan valita kutsun tyyppi POST, jonka jälkeen lisätään osoite, johon kutsu lähetetään (kuva 9).



Kuva 9. RPC-kutsun tyyppi ja osoite

Otsikot-välilehdelle lisätään Accept, Content-Type sekä Authorization -otsikot (kuva 10).

▼ Headers (3)		
	KEY	VALUE
<input checked="" type="checkbox"/>	Accept	application/json
<input checked="" type="checkbox"/>	Content-Type	application/json
<input checked="" type="checkbox"/>	Authorization	--user admin:admin

Kuva 10. RPC-kutsun otsikot

Kutsun vartaloksi annetaan nimi ja numero (kuva 11). Vartalossa syntaksin oikeanlaisuus on tärkeää. Vääränlaisella syntaksilla kutsua ei saada lähetettyä onnistuneesti.

```
1 {  
2   "input": {  
3     "name": "Teemu",  
4     "number1": 1  
5   }  
6 }
```

Kuva 11. RPC-kutsun vartalo

Edellä mainitulla kutsulla saadaan kuvan 12 mukainen vastaus. Vastaus sisältää siis oksan greeting, joka taas sisältää sanan "Hello" sekä Hello world RPC:ssä asetetun nimen ja numeron.



Kuva 12. RPC-kutsun vastaus

### 3.7 Ilmoitusten lisääminen

YANG-malliin lisätään ilmoitus luomalla notification-niminen oksa. Ilmoitukselle voidaan antaa kuvaus, jonka tarkoituksena on nimenomaan kuvata ilmoituksen tehtävä. Kuvauksen lisäksi ilmoitukselle annettiin oksiksi nimi ja numero, jotka ovat tyyppiä string ja uint32 (kuva 13).

```
notification helloDone {
    description "Done";
    leaf name {
        type string;
    }
    leaf number1 {
        type uint32;
    }
}
```

Kuva 13. Ilmoitus YANG-mallissa

Ilmoituksille luotiin ajaja, jonka avulla ilmoitusten testaamista saadaan helpotettua huomattavasti (kuva 14). Ajaja voidaan asettaa ajamaan koodia esimerkiksi 5 sekunnin välein, jolloin ilmoituksia saadaan myös välitetyksi käyttöliittymälle viiden sekunnin välein. Ilmoituksen sisältö muuttuu jokaisella ajokierroksella, joten datan muutokset voidaan huomata käyttöliittymässä tai selaimen kehittäjän työkaluissa.

Toinen vaihtoehto olisi toteuttaa huomautus itse "Hello world" -RPC:n sisällä. Tämä tarkoittaa sitä, että kun RPC suoritetaan, lähetetään myös samalla ilmoitus käyttöliittymälle. Tällaisessa testitapauksessa ilmoituksen lähettäminen RPC:n yhteydessä on hyödyllistä. Yhdellä kutsulla voidaan todentaa sekä ilmoituksen että RPC:n toimivuus.

```

private static long count = 0;
Runnable helloDone = () -> {
    count++;
    HelloDoneBuilder notifyBuilder = new HelloDoneBuilder();
    notifyBuilder.setName("TESTNOTIFICATION" + count).setNumber1(count);
    notificationProvider.offerNotification(notifyBuilder.build());
    LOG.info("Received HelloDone notification" + notifyBuilder.getName() + notifyBuilder.getNumber1());
};

ScheduledExecutorService ses = Executors.newScheduledThreadPool(1);
ScheduledFuture scheduledFuture = ses.scheduleAtFixedRate(helloDone, 5, 5, TimeUnit.SECONDS);

```

Kuva 14. HelloDone-ilmoituksen ajaminen

### 3.8 Ilmoitusten käsittely virtojen avulla käyttöliittymässä

Ilmoituksia voidaan välittää esimerkiksi käyttöliittymälle niin sanottujen virtojen avulla. Virta avataan yksinkertaisella POST-kutsulla, joka palauttaa paluuarvona virran nimen (kuva 15). Virran nimen avulla voidaan avata WebSocket-yhteys, joka mahdollistaa reaaliaikaisen datan siirtämisen. Ilmoituksia voidaan hyödyntää monella tapaa. Ilmoituksissa voidaan suoraan välittää laite- tai tiladataa esimerkiksi käyttöliittymälle. Ilmoituksia voidaan käyttää myös niin sanottuina laukaisimina käyttöliittymässä. Niiden avulla voidaan laukaista jokin tietty kutsu tai käyttäjälle näkyvillä olevaa dataa voidaan muokata halutunlaiseksi.

OpenDaylightissa autentikoitumiseen tarvitaan vakiona käyttäjätunnukseksi ja salasanaaksi admin. Tunnukset syötetään jokaisen HTTP-kutsun otsikossa, jotta autentikointi saadaan onnistumaan.

```

openNotificationStream() {
  axios({
    method: 'post',
    url: 'http://localhost:8181/restconf/operations/sal-remote:create-notification-stream',
    data: {
      "input": {
        "notifications": [
          "(urn:opendaylight:params:xml:ns:yang:hello?revision=2019-11-27)helloDone"
        ],
        "notification-output-type": "JSON"
      },
      headers: {
        'Authorization': 'Basic ' + btoa('admin:admin'),
        'Content-Type': 'application/json',
        'Accept': 'application/json'
      }
    })
    .then(res => {
      console.log(res);
    })
    .catch(err => {
      console.log(err);
    })
  })
}

```

Kuva 15. Funktio ilmoitusvirran avaamiseksi

Tämän jälkeen voidaan tilata ilmoitusvirta edellisen kutsun palauttaman [URL:n](#) avulla (kuva 16). Tilauskutsu onnistuessaan palauttaa WebSocketin [URL:n](#).

```

subscribeNotificationStream() {
  axios({
    method: 'get',
    url: 'http://localhost:8181/restconf/streams/stream/create-notification-stream/hello:helloDone',
    headers: {
      'Authorization': 'Basic ' + btoa('admin:admin'),
      'Content-Type': 'application/json',
      'Accept': 'application/json'
    }
  })
  .then(res => {
    console.log(res);
  })
  .catch(err => {
    console.log(err);
  })
}

```

Kuva 16. Funktio virran tilaamiseksi

Avataan WebSocket-yhteys osoitteeseen, minkä edellinen tilauskutsu on palauttanut (kuva 17). Tässä testitapauksessa on käytetty staattista osoitetta, koska osoite on ollut jo tiedossa etukäteen eikä siihen ole ollut tulossa muutoksia. Tuotantotapauksessa osoite voitaisiin syöttää parametrina WebSocketin avaavaan funktioon.

```

openWebSocket() {
  try {
    var socketLocation = "ws://localhost:8185/create-notification-stream/hello:helloDone";
    var notificationSocket = new WebSocket(socketLocation);

    notificationSocket.onmessage = (event) => {
      console.log('Received HelloDone NOTIFICATION.');
```

Kuva 17. Funktio WebSocket-yhteyden avaamiseksi

Chromen kehittäjän työkalujen konsoliin saatu ilmoitus sisältää YANG-mallissa määritellyt oksat (kuva 18). Niiden arvot voidaan esimerkiksi lukea käyttöliittymään tai ilmoituksella voidaan käskyttää käyttöliittymää. Alla olevasta kuvasta huomataan, että kyseessä on kolmas ilmoitus, jonka palvelin on luonut käynnistyttyään. Tämä huomataan number1:n arvosta 3. Arvo kasvaa joka kerta, kun ilmoitus ajetaan.

```

Received HelloDone NOTIFICATION.
▼ {ietf-restconf:notification: {...}, event-time: "2020-02-13T14:44:36.164+02:00"} ⓘ
  event-time: "2020-02-13T14:44:36.164+02:00"
  ▼ ietf-restconf:notification:
    ▼ hello:helloDone:
      name: "TESTNOTIFICATION3"
      number1: 3
      ► __proto__: Object
      ► __proto__: Object
      ► __proto__: Object
```

Kuva 18. Ilmoitus Chromen kehittäjän työkaluissa

## 4 PALVELINOHJELMISTO

### 4.1 Ohjelmistokehityksen valinta

Alkuperäisessä toteutuksessa hyödynnetään Lightya ja myöskin tässä työssä oli tarkoitus hyödyntää sitä. WebSocket-yhteyttä muodostaessa Lightyssa huomattiin ohjelmointivirhe, jonka takia alusta jouduttiin vaihtamaan OpenDaylightiin. Lightyssa virran luominen onnistuu samaan tapaan kuin OpenDaylightissa, mutta virran tilaaminen ei onnistu. Virta ikään kuin katoaa ja sitä kutsuessa saadaan 404 virhe.

Edellä mainittua ohjelmointivirhettä koitettiin selvittää, mutta ratkaisua ei suoraan löytynyt. Lightyn kehittäjät selvittävät ohjelmointivirhettä, mutta tietoa selvityksen valmistumisen ajankohdasta ei osattu sanoa. Toisin sanoen kyseisen WebSocket-yhteyden luominen tällä hetkellä Lightylla on mahdotonta. Vaikka aikaisemmin on mainita siitä, että Lighty on kevyempi sovellus kuin OpenDaylight, löytyy OpenDaylightistakin hyviä puolia.

Lightyn kanssa tarvitaan proxy-palvelin, joka tässä tapauksessa oli NGINX. OpenDaylight sisältää oman proxy-palvelimensa, joka helpottaa tässä tapauksessa. Käyttöliittymästä HTTP-kutsut voidaan suoraan ohjata porttiin 8181 ja WebSocket-yhteys voidaan avata portissa 8185. Edellä mainitut portit ovat OpenDaylightin vakioportit näille operaatioille.

### 4.2 Toiminta

Palvelinohjelmiston pääasiallisena tehtävänä on ylläpitää ja säilöä laitedataa sekä ilmoittaa laitedataan liittyvistä muutoksista. Laitedatan muutoksilla tarkoitetaan datan lisäämistä, muokkaamista tai poistamista.

Palvelinohjelmistolle tehtiin samankaltaisia muutoksia useaan eri paikkaan. Palvelinohjelmisto lähettää ilmoituksia aina, kun dataa lisätään, muokataan tai poistetaan. Näin ollen data saadaan liikkumaan käyttöliittymälle lähes reaaliajassa.



Jos laitedatan lisäämisessä, muokkaamisessa tai poistamisessa kohdataan virhe, palvelinohjelmisto ilmoittaa siitä ja käyttäjä saa ilmoituksen suoraan käyttöliittymässä.

Tuotantoympäristössä palvelimen on tarkoitus keskustella eri verkkolaitteiden kanssa SNMP-protokollan avulla. Sovelluskehitysvaiheessa kuitenkin hyödynnetään fyysisten laitteiden sijasta SNMP-simulaattoria. SNMP-simulaattorilla voidaan simuloida tuhansia erilaisia laitteita, jotka hyödyntävät SNMP-protokollaa. Simulaattoria hyödynnetään lähinnä testaus- ja sovelluskehitysvaiheissa sen yksinkertaisuuden vuoksi (kuva 20).

```
snmpsimd.py --v2c-arch --logging-method=stdout --data-dir={DATA_DIRECTORY}  
--agent-udp4-endpoint=127.0.0.1.2001 --variation-modules-dir=/usr/local/snmpsim/variation
```

Kuva 20. Komento SNMP-simulaattorin käynnistämiseksi

Edellä mainitussa komennossa kolmantena parametrina on datan tiedostosijainti. Tarvittaessa datatiedoston voi hakea Githubista (<https://github.com/etingof/snmpsim/blob/master/data/public.snmprec>). Tällöin datasijaintiparametri voidaan laittaa osoittamaan ladattuun public.snmprec-tiedostoon. SNMP-simulaattorin avulla datalle voidaan antaa myös halutut vaihteluvälit, jolloin simulaattori simuloi datan muutosta.

## 5 KÄYTTÖLIITTYMÄ

### 5.1 Toiminta

Aikaisempi käyttöliittymä on kysellyt palvelimelta laitetietoja tietyn väliajan välein käyttäjän valinnasta riippuen. Nyt käyttöliittymää muutettiin siten, että se kysyy laitetietoja saatuaan palvelimelta huomautuksen. Tämän avulla vähennetään huomattavasti turhia kyselyitä.

Käyttöliittymään tehtiin WebSocket-yhteys, joka saa palvelimelta huomautuksen, kun dataan on tullut muutos. Huomautuksen saatuaan, käyttöliittymä kysyy palvelimelta muutoksessa tulleet tiedot. WebSocket-yhteys tehtiin samalla tavalla kuin kappaleessa 4 käsitellyssä esimerkkisovelluksessa.

Toteutettu WebSocket-yhteys mahdollistaa verkkolaitteiden datan dynaamisen ja lähes reaaliaikaisen päivityksen selaimessa. Tämän ansiosta verkkolaitteiden tilaa voidaan seurata tarkemmin käyttöliittymän kautta.

### 5.2 Kutsut

Luvussa 4.1. mainitaan, että OpenDaylightin kanssa ei tarvita erillistä proxy-palvelinta. OpenDaylight kuuntelee HTTP-kutsuja vakiona portissa 8181. Kutsut voidaan ohjata suoraan kyseiseen porttiin eikä CORS:in kanssa tarvitse taistella. Tämä tekee HTTP-kutsujen toteuttamisesta todella suoraviivaista.

## 6 POHDINTA

Itse toiminnallisuuden tekemiseen ei kulunut paljoa aikaa. Suurin osa ajasta käytettiin OpenDaylightiin, Lighttyyn sekä SDN:iin tutustumisessa. Voidaan tietenkin olettaa, että tutustuminen on auttanut huomattavasti itse toteutuksen tekemisessä. Opinnäytetyön tekemisen alussa tietämys SDN:sta oli lähellä nollaa. Nyt opinnäytetyön valmistuessa voidaan sanoa tietämyksen sekä ymmärtämyksen kehittyneen huomattavasti. Opinnäytetyö on ollut todella hyödyllinen sekä teki jälle että työpaikalle.

SDN parantaa ja helpottaa huomattavasti verkkojen ylläpitämistä sekä niiden ohjelmoitavuutta. Tämän ansiosta verkkojen älykkyys kasvaa. Edellä mainittu taas johtaa siihen, että verkkokustannuksia voidaan pienentää ja palvelun laatua parantaa. SDN mahdollistaa myös verkon liikenteen monitoroinnin ja ohjauksen sovellustasolla, esimerkiksi verkkokäyttöliittymässä.

SDN voi tulevaisuudessa mahdollisesti heikentää laitevalmistajien markkinatilannetta. SDN:n ansiosta laitteistoa ei tarvitse ostaa enää samalta valmistajalta, vaan kaikkien valmistajien laitteet tukevat samoja SDN-sovelluksia. Toisaalta tämä mahdollistaa pienempien laitevalmistajien kasvun ja myynnin nousun.

Työ parantaa huomattavasti käyttäjäkokemusta sekä sovelluksen luotettavuutta. Lähes reaaliaikaisten ilmoitusten avulla myös palvelinohjelmiston muutoksia voidaan tarkkailla huomattavasti paremmin kuin aikaisemmin. Ilman WebSocket-yhteyttä huomattava määrä palvelinohjelmiston muutoksista voi jäädä käyttäjältä huomaamatta käyttöliittymästä pitkän päivitysajan takia.

## LÄHTEET

Ciena. n.d. What is SDN?. Luettu 11.2.2020. <https://www.ciena.com/insights/what-is/What-Is-SDN.html>

HowToDoInJava. Lokesh Gupta. 2016. What is Java programming language? Luettu 10.3.2020. <https://howtodoinjava.com/java/basics/what-is-java-programming-language/>

Ilya Etingof. 2019. SNMP Agent Simulator. Luettu 2.3.2020. <http://snmplabs.com/snmpsim/>

Internet Engineering Task Force (IETF). 10/2010. YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF). Luettu 6.2.2020. <https://tools.ietf.org/html/rfc6020>

Internet Engineering Task Force (IETF). 12/2011. The WebSocket Protocol. Luettu 18.3.2020. <https://tools.ietf.org/html/rfc6455>

Internet Engineering Task Force (IETF). 1/2017. RESTCONF Protocol. Luettu 6.2.2020. <https://tools.ietf.org/html/rfc8040>

Internet Engineering Task Force (IETF). 6/2011. Network Configuration Protocol (NETCONF). Luettu 6.2.2020. <https://tools.ietf.org/html/rfc6241>

MDN. 3.9.2019. WebSocket. Luettu 3.3.2020. <https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>

Mozilla. 9.12.2019. JavaScript. Luettu 7.2.2020. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

OpenDaylight. n.d. Current release. Luettu 3.3.2020. <https://www.opendaylight.org/what-we-do/current-release/sodium>

OpenDaylight. n.d. Developing Apps on the OpenDaylight controller. Luettu 13.2.2020. <https://docs.opendaylight.org/en/stable-sodium/developer-guide/developing-apps-on-the-.opendaylight-controller.html>

OpenDaylight. n.d. Platform Overview. Luettu 18.3.2020. <https://www.opendaylight.org/what-we-do/odl-platform-overview>

PANTHEON.tech. 2019. FAQ. Luettu 6.2.2020. <https://lighty.io/>

**LIITTEET****Liite 1. hello.yang**

---

```
module hello {
  yang-version 1;
  namespace "urn:opendaylight:params:xml:ns:yang:hello";
  prefix "hello";
  revision "2019-11-27" {
    description "Initial revision of hello model";
  }
  rpc hello-world {
    input {
      leaf name {
        type string;
      }
      leaf number1 {
        type uint32;
      }
    }
    output {
      leaf greeting {
        type string;
      }
    }
  }
  notification helloDone {
    description "Done";
    leaf name {
      type string;
    }
    leaf number1 {
      type uint32;
    }
  }
}
```

## Liite 2. HelloProvider.java

---

```

public class HelloProvider {

    private static final Logger LOG = LoggerFactory.getLogger(HelloProvider.class);

    private final DataBroker dataBroker;
    private ObjectRegistration<HelloService> helloService;
    private RpcProviderService rpcProviderService;

    private NotificationPublishService notificationProvider;

    private static final InstanceIdentifier<Hello> HELLO_IID = InstanceIdentifier.builder(Hello.class).build();
    private static final ExecutorService EXECUTOR = Executors.newFixedThreadPool(2);

    public HelloProvider(final DataBroker dataBroker, final RpcProviderService rpcProviderService,
        NotificationPublishService notificationPublishService) {
        this.dataBroker = dataBroker;
        this.rpcProviderService = rpcProviderService;
        this.notificationProvider = notificationPublishService;
    }

    public void init() {
        LOG.info("HelloProvider Session Initiated");
        helloService = rpcProviderService.registerRpcImplementation(HelloService.class,
            new HelloWorldImpl(this.notificationProvider, this.dataBroker));
    }

    public void close() {
        LOG.info("HelloProvider Closed");
        if (helloService != null) {
            helloService.close();
        }
    }
}

```

### Liite 3. HelloWorldImpl.java

```

public class HelloWorldImpl implements HelloService {

    private static final Logger LOG = LoggerFactory.getLogger(HelloWorldImpl.class);
    private final DataBroker dataBroker;
    private static final InstanceIdentifier<Hello> HELLO_IID = InstanceIdentifier.builder(Hello.class).build();
    private static final ExecutorService EXECUTOR = Executors.newFixedThreadPool(2);

    private NotificationPublishService notificationProvider;

    public HelloWorldImpl(NotificationPublishService notificationPublishService, final DataBroker dataBroker) {
        this.notificationProvider = notificationPublishService;
        this.dataBroker = dataBroker;
    }

    @Override
    public ListenableFuture<RpcResult<HelloWorldOutput>> helloWorld(HelloWorldInput input) {
        HelloWorldOutputBuilder helloBuilder = new HelloWorldOutputBuilder();
        helloBuilder.setGreeting("Hello " + input.getName() + " " + input.getNumber1());

        notificationProvider.offerNotification(new HelloDoneBuilder().setName(input.getName())
            .setNumber1(input.getNumber1()).build());
        LOG.info("\n Sent the notification, done by the hello package.");

        Hello hello = new HelloBuilder().setName(input.getName()).setNumber1(input.getNumber1()).build();

        WriteTransaction tx = dataBroker.newWriteOnlyTransaction();
        tx.put(CONFIGURATION, HELLO_IID, hello);
        tx.commit().addCallback(new FutureCallback<CommitInfo>() {

            @Override
            public void onSuccess(CommitInfo arg0) {
                LOG.info("Added a Hello from HelloWorldImpl");
            }

            @Override
            public void onFailure(Throwable throwable) {
                LOG.error("Failed to write a Hello {} from HelloWorldImpl");
            }
        }, EXECUTOR);

        return RpcResultBuilder.success(helloBuilder.build()).buildFuture();
    }

    private static long count = 0;
    Runnable helloDone = () -> {
        count++;
        HelloDoneBuilder notifyBuilder = new HelloDoneBuilder();
        notifyBuilder.setName("TESTNOTIFICATION" + count).setNumber1(count);
        notificationProvider.offerNotification(notifyBuilder.build());
        LOG.info("Received HelloDone notification" + notifyBuilder.getName() + notifyBuilder.getNumber1());
    };

    ScheduledExecutorService ses = Executors.newScheduledThreadPool(1);
    ScheduledFuture scheduledFuture = ses.scheduleAtFixedRate(helloDone, 5, 5, TimeUnit.SECONDS);
}

```