# Povezivanje upravljačkog uređaja OpenDayligt i simulatora/emulatora IMUNES

## Tehnička dokumentacija

**Verzija 1.2**


**Studentski tim:** Matija Šantl

**Nastavnik:** prof. dr. sc. Maja Matijašević
**Voditelj:** dr. sc. Ognjen Dobrijević

# Sadržaj

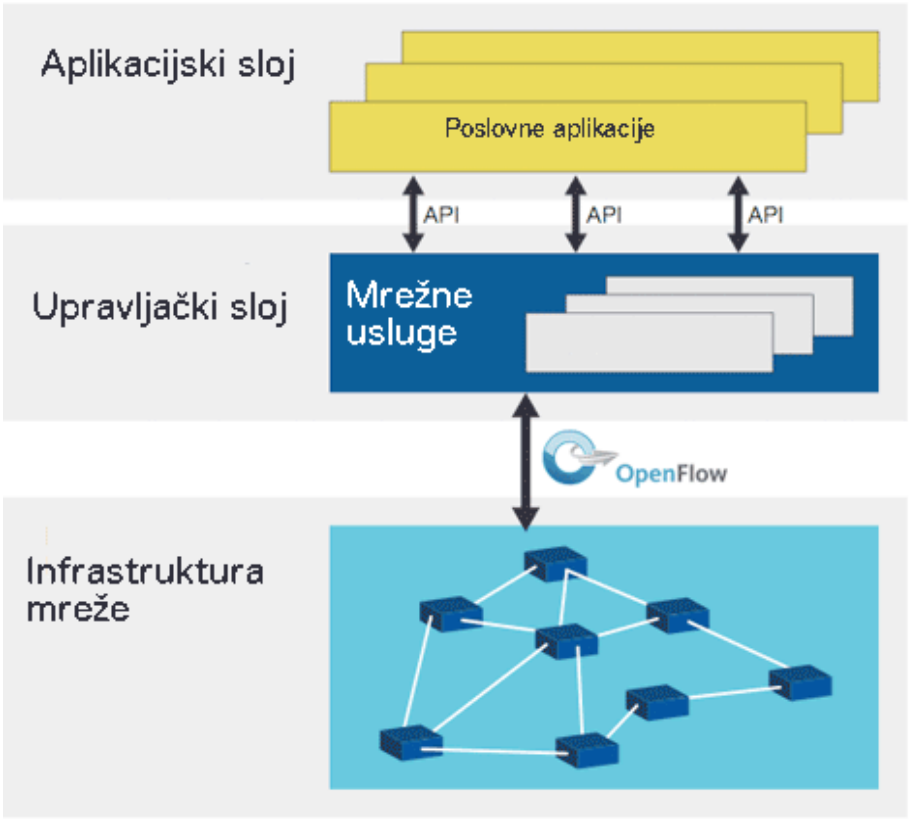
# 1. Uvod

Klasične komunikacijske mreže temeljene ne već programiranim uređajima čije karakteristike ovise od proizvođača do proizvođača onemogućuju provedbu ispitivanja novih komunikacijskih protokola u produkcijskim mrežama. Kao rješenje tog problema, predložena je specifikacija *OpenFlow* kao izvedba programski upravljanih komunikacijskih mreža. Osnovna ideja predloženoga pristupa je upravljanje prosljeđivanjem podataka u komutatorima pomoću programske podrške smještene na zasebnom računalu, tzv. upravljačkom uređaju.

Programski upravljane komunikacijske mreže (engl. *software-defined networks*, SDN) su komunikacijske mreže kod kojih je funkcija prosljeđivanja prometa u mrežnim uređajima odvojena od funkcije upravljanja prosljeđivanjem. Funkcija upravljanja prosljeđivanja prometa se izvodi na upravljačkom uređaju (engl. *controller*) koji definira pravila prosljeđivanja za pojedine tokove podataka.

Objasnimo najprije dva sučelja čiji se nazivi često pojavljuju u opisu komunikacijskih mreža. To su *northbound* i *southbound* sučelja. *Northbound* sučelje je sučelje koje određenoj komponenti programski upravljane komunikacijske mreže omogućuje komunikaciju s komponentama višeg sloja. Ono opisuje proctor protokolom podržane komunikacije između upravljačkog uređaja i aplikacije ili upravljačkih programa višeg sloja. *Southbound* sučelje je sučelje koje određenoj komponenti programski upravljane komunikacijske mreže omogućuje komunikaciju s komponentama nižeg sloja.

Specifikacija *OpenFlow* (OF) definira komunikacijski protokol između upravljačkog i mrežnih uređaja, kojeg onda ti uređaji koriste za međusobnu komunikaciju. Specifikacija *OpenFlow* predstavlja *southbound* sučelje programski upravljanih komunikacijskih mreža. Jedna od njenih glavnih zadaća je omogućiti ostvarivanje zahtjeva koristeći *northbound* sučelje [6].




**Slika 1. Arhitektura programski upravljanih komunikacijskih mreža [5]**

*OpenDaylight* je upravljački uređaj otvorenog koda koji je napisan u programskom jeziku *Java*. Kao takav, upravljački uređaj *OpenDaylight*, se može koristiti na svim uređajima i operacijskim sustavima koji podržavaju *Javu.* Dodatno, usluge orijentirane prema platformi i druga proširenja također se mogu ugraditi u upravljački uređaj za poboljšanu funkcionalnost programski upravljanih komunikacijskih mreža [4].

*REST* (engl. *Representational State Transfer*) je stil programske arhitekture za izgradnju distribuiranih sustava. *REST* nije tehnologija niti arhitektura, već skup kriterija za dizajn skalabilnih raspodijeljenih sustava.

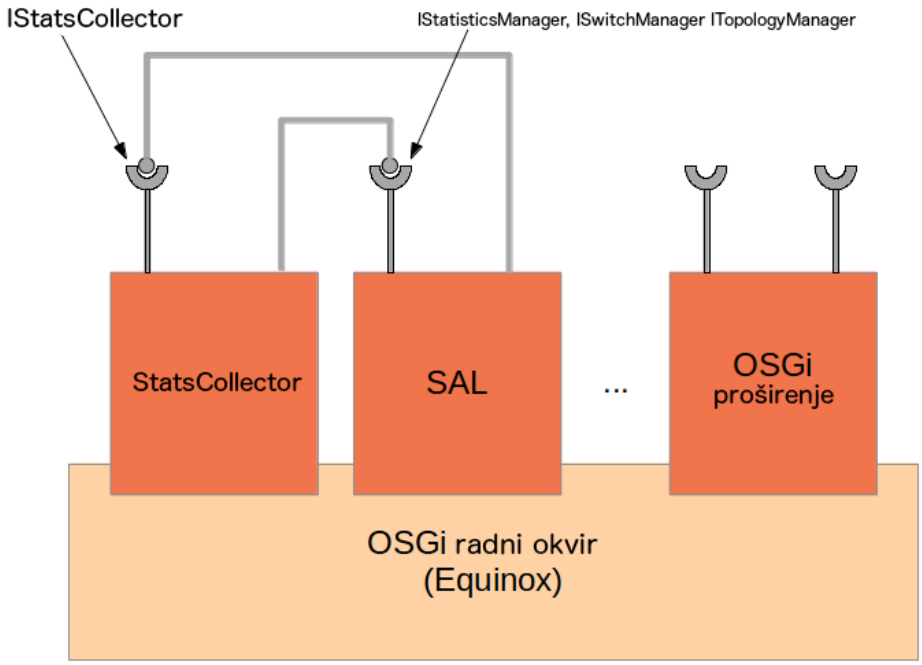
## 2. Opis razvijenog proizvoda

PODESI (***P****ovezivanje upravljačkog uređaja* ***O****pen****D****aylight i* ***e****mulatora/****s****imulatora* ***I****MUNES*), je skup od dva proširenja namijenjenih za upravljački uređaj *OpenDaylight* napisanih u programskom jeziku *Java*. Prvo proširenje, *StatsCollector*, periodički prikuplja statističke podatake od mreženih uređaja u mreži, dok se drugo proširenje, *MyStats*, koristi za obradu tih podataka koji će se kasnije koristiti prilikom optimalnih puteva između dva računala u mreži.

OSGi (*engl. Open Service Gateway initiative*) specifikacija opisuje modularan sustav i platformu usluge za programski jezik *Java* koja implementira potpun i dinamičan model komponenti. Aplikacije ili komponente, koje dolaze u obliku proširenja za razvijanje, se mogu udaljeno pokrenuti, zaustaviti, osvježiti te ukloniti bez potrebe ponovnog pokretanja cijelog sustava. Upravljački uređaj *OpenDaylight* koristi *OSGi* specifikaciju kako bi se jezgrene komponente i proširenja mogla jednostavno dodavati te time, bez potrebe ponovnog pokretanja cijelog sustava, lako promijeniti način rada upravljačkog uređaja. Prilikom razvijanja proširenja potrebno je implementirati metode *start, stop, init* te *destroy* u razredu proširenja.

Dodatno, prilikom izrade proširenja korištena je tzv. apstrakcija slojeva sustava vođena aplikacijskim programskim sučeljem (*engl. API*[1]*-driven Service Abstraction Layer, AD-SAL*).

Arhitektura sustava sastoji se od više *OSGi* proširenja ostvarenih kao *Java* razredi. Proširenja se pokreću u skolpu *OSGi* radnog okvira (unutar upravljačkog uređaja *OpenDaylight,* koristi se naziv *Equinox*).




**Slika 2. Arhitektura *OSGi* proširenja**

[1] API (*engl. Application Program Interface*)

### 2.1 Opis problema i funkcijski zahtjevi

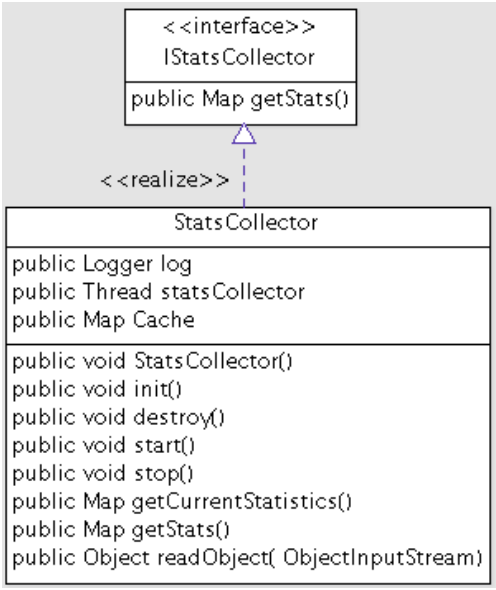
Programski upravljane komunikacijske mreže (*engl. SDN*) uvode dodatne stupnjeve slobode prilikom upravljanja usmjeravanjem prometa. Jedan način iskorištavanje te mogućnosti je *pametno* usmjeravanje tokova s obzirom na trenutno stanje mrežnih uređaja. Za takvo usmjeravanje potrebno je dohvatiti i obraditi podatke te definirati sam tok. Prilikom izrade proširenja, pazilo se na sljedeće funkcijske zatjeve.

1. Definiranje novog toka prilikom dolaska novog paketa. Kako bi se to ostvarilo, rješenje mora imati pristup *REST* sučelju upravljačkog uređaja ili pak biti izvedeno kao proširenje upravljačkog uređaja.
2. Uvid u trenutno stanje mrežnih uređaja. Opet su moguća dva načina, rješenje mora imati pristup *REST* sučelju ili može biti izvedeno kao proširenje.
3. Mogućnost izračuna statistika potrebnih za odabir *optimalnih* tokova. Budući da bi rješenje koje bi se periodički pokretalo i dohvaćalo takve podatke bilo ovisno o sustavu na kojem se razvija, rješenje koje bi koristilo *REST* sučelje je odbačeno. Rješenje koje bi bilo izvedeno kao proširenje upravljačkog uređaja može se pokrenuti unutar samo upravljačkog uređaja, te je time neovisno o sustavu na kojem se razvija, a pritom zadovoljava sve zahtjeve.

### 2.2 Opis programske izvedbe

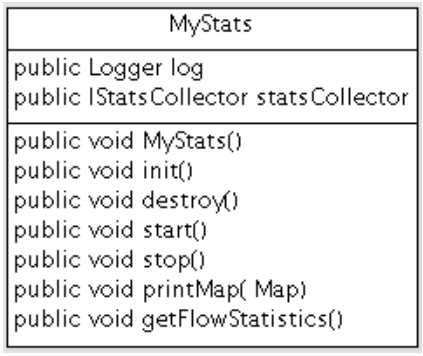
Iz Slika 2 se može primijetiti da *OSGi* proširenja izlažu usluge koje druga proširenja mogu koristiti. Proširenje *StatsCollector* koristi usluge koje izlažu jezgrene komponente *IStatisticsManager, ISwitchManager* i *ITopologyManager* te izlaže uslugu *IStatsCollector* koju koristi proširenje *MyStats.*

Proširenje *StatsCollector* je ostvareno pomoću istoimenog razreda. Dijagram razreda (*engl. Class diagram*) *StatsCollector* prikazan je u nastavku.




**Slika 3. Dijagram razreda *StatsCollector***

Iz dijagrama razreda je vidljivo da razred *StatsCollector* realizira sučelje *IStatsCollector* koje je potrebno za predati *OSGi* radnom okviru kako bi se omogućilo pozivanje metoda tog sučelja iz drugih proširenja unutar *OSGi* radnog okivra. Također možemo primijetiti metode *init, destroy, start* i *stop*, koje su potrebne kako bi se proširenje moglo pravilno učitati, te njime kasnije baratati iz *OSGi* konzole. Metoda *getCurrentStatistics* se koristi od strane dretve koja periodično prikuplja statističke podatke, dok je *getStats* metoda sama implementacija sučelja pomoću kojeg se dohvaćaju prikupljeni statistički podaci. Metoda *readObject* je potrebna kako bi se ostvarila deserijalizacija ulaznog toka iz *OSGi* radnog okvira.




**Slika 4. Dijagram razreda *MyStats***

Dijagram razreda proširenja *MyStats* je malo jednostavniji od prethodnog. Uz potrebne metode, *init, destroy, start* i *stop,* implementirana je metoda *getFlowStatistics* koja uspostavlja komunikaciju s proširenjem *IStatsCollectori* te dohvaća prikupljene podatke. Osim toga, u toj metodi se također dohvaćaju podaci o topologiji mreže kako bi se upotpunilo trenutno stanje mreže.


**Slika 5. Dijagram razreda *Data***

Razred *Data* središnja je struktura podataka koju koriste proširenja *MyStats* i *StatsCollector*. Struktura podataka je prilagođena spremanju i dohvatu statističkih podataka dohvaćenih od mrežnih uređaja te njihovih oznaka.

Statistički podaci koji su potrebni za pronalaženje optimalnih puteva između dva računala su postotak izgubljenih paketa (engl. Packet loss) i kašnjenje (engl. Delay). Upravljački uređaj OpenDaylight periodički dohvaća sve statističke podatke od mrežnih uređaja. Podaci koje mrežni uređaji šalju upravljačkom uređaju mogu se podijeliti u 14 skupina. To su *Individual Flow Statistics*, *Aggregate Flow Statistics*, *Flow Table Statistics*, *Port Statistics*, *Group Description*, *Group Statistics*, *Meter Configuration*, *Meter Statistics*, *Queue Statistics*, *Node Description*, *Flow Table Features*, *Port Description*, *Group Features* i *Meter Features*. Nama potrebni podaci, gubitak paketa (*engl. Packet loss)* i kašnjenje *(engl. Delay),* se računaju pomoću podataka koji se nalaze u skupini *Port Statistics*. Za periodičko prikupljanje statističkih podataka u upravljčkom uređaju *OpenDaylight* zadužen je *SDNStatsCollector* modul. Za dohvaćanje statističkih podataka od upravljačkog uređaja OpenDaylight koristi se *StatisticsManager* sučelje koje izlaže metode za dohvaćanje potrebnih podataka.

Za potrebe određivanja optimalnih puteva između dva računala u mreži potrebno je znati i topologiju mreže. Upravljački uređaj *OpenDaylight* izlaže metode za manipulaciju i dohvaćanje topologije preko sučelja *TopologyManager*. Postojeću strukturu podataka koju dobijemo od proširenja *StatsCollector* nadopunjujemo podacima o topologiji. Time dobijamo strukturu podataka koja sadrži sve potrebne podatke za računanje optimalnih putava u mreži.

Definirat ćemo dvije vrste optimalnih puteva između dva čvora. Optimalni put s obzirom na gubitak paketa (*engl. packet loss*) te optimalni put s obzirom kašnjenje (*engl. delay)*. Ako želimo uspostaviti audio sjednicu, tada nam je važnije da nam kašnjenje bude čim manje, dakle optimalan put je onaj koji ima minimalno kašnjenje, a ako želimo uspostaviti video sjednicu, tada name je važnije da izgubimo čim manje paketa, dakle optimalan put je onaj na kojem izgubimo najmanje paketa.

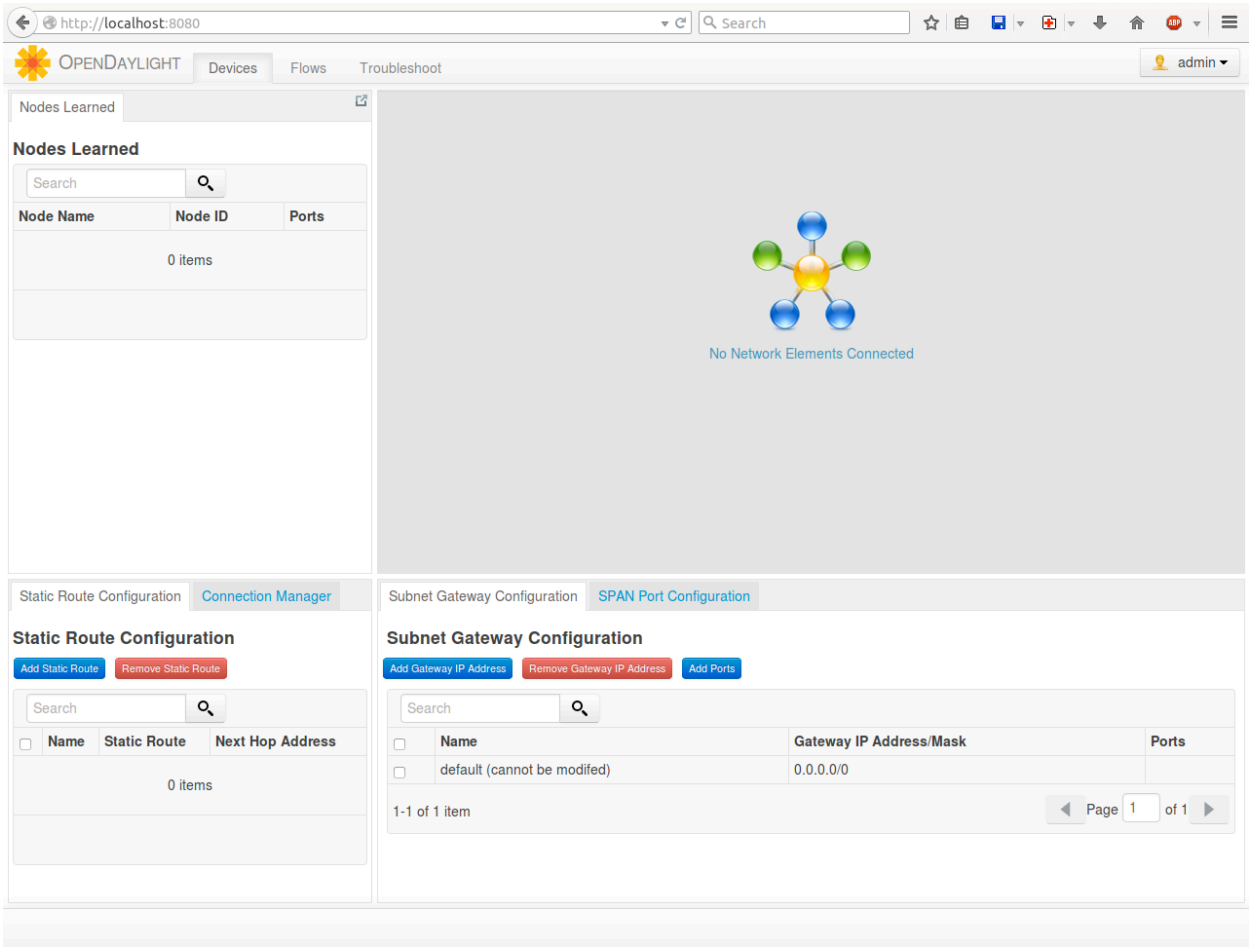
Pokretanjem upravljačkog uređaja OpenDaylight, te aktiviranjem proširenja *StatsCollector*, upravljački uređaj počinje prikupljati statističke podatke te ih sprema u privremenu memoriju od tisuću zapisa. Na taj način u memoriji upravljačkog uređaja imamo zapisano prošlo stanje iz kojeg možemo zaključiti kako će se određeni mrežni uređaji ponašati ako ih stavimo pod opterećenje. Aktiviranjem proširenja *MyStats*, dohvaćamo prikupljene podatke iz memorije i topologiju mreže, te na temelju tih podataka možemo računati optimalan put između dva računala u mreži.

## 3. Tehničke značajke

Ovaj projekt je ostvaren u programskom jeziku *Java* jer je i sam upravljački uređaj, OpenDaylight, ostvaren u tom programskom jeziku. Uz *Javu*, korišten je i *Maven*, alat za upravljanje i razumijevanje programskih projekata. *Maven* upravlja izgradnjom projekta tako da se definira POM (*engl. Project Object Model*) datoteka u kojoj je definirano na koji način je potrebno izgraditi projekt i koje sve biblioteke uključiti izvršnu datoteku.

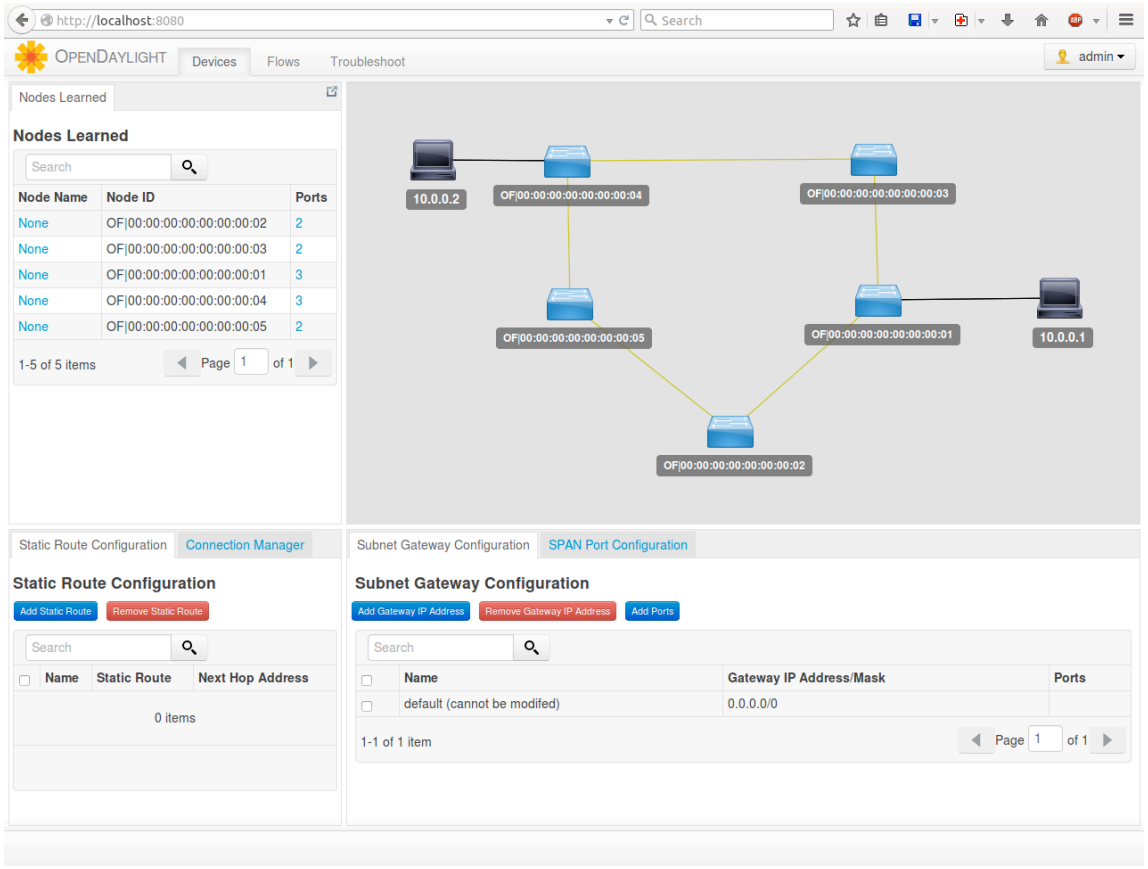
Sam upravljački uređaj OpenDaylight, pa tako i ostvarena proširenja, koristi *git* kao sustav za upravljanje verzijama. Prilikom izrade proširenja sve promjene su bile spremane u *git* repozitorij.

Prilikom izrade ovog projekta korišten je upravljački uređaj *OpenDaylight*, verzija *Helium*.




**Slika 6. Web sučelje upravljačkog uređaja *OpenDaylight***

Računalo na kojem je bio pokrenut upravljački uređaj je bilo fizički odvojeno od računala na kojem je bio pokrenut simulator/emulator IMUNES. Računala su umrežena prema primjerima iz [3].




**Slika 7. Web sučelje upravljačkog uređaja *OpenDaylight* nakon uključivanja mreže**

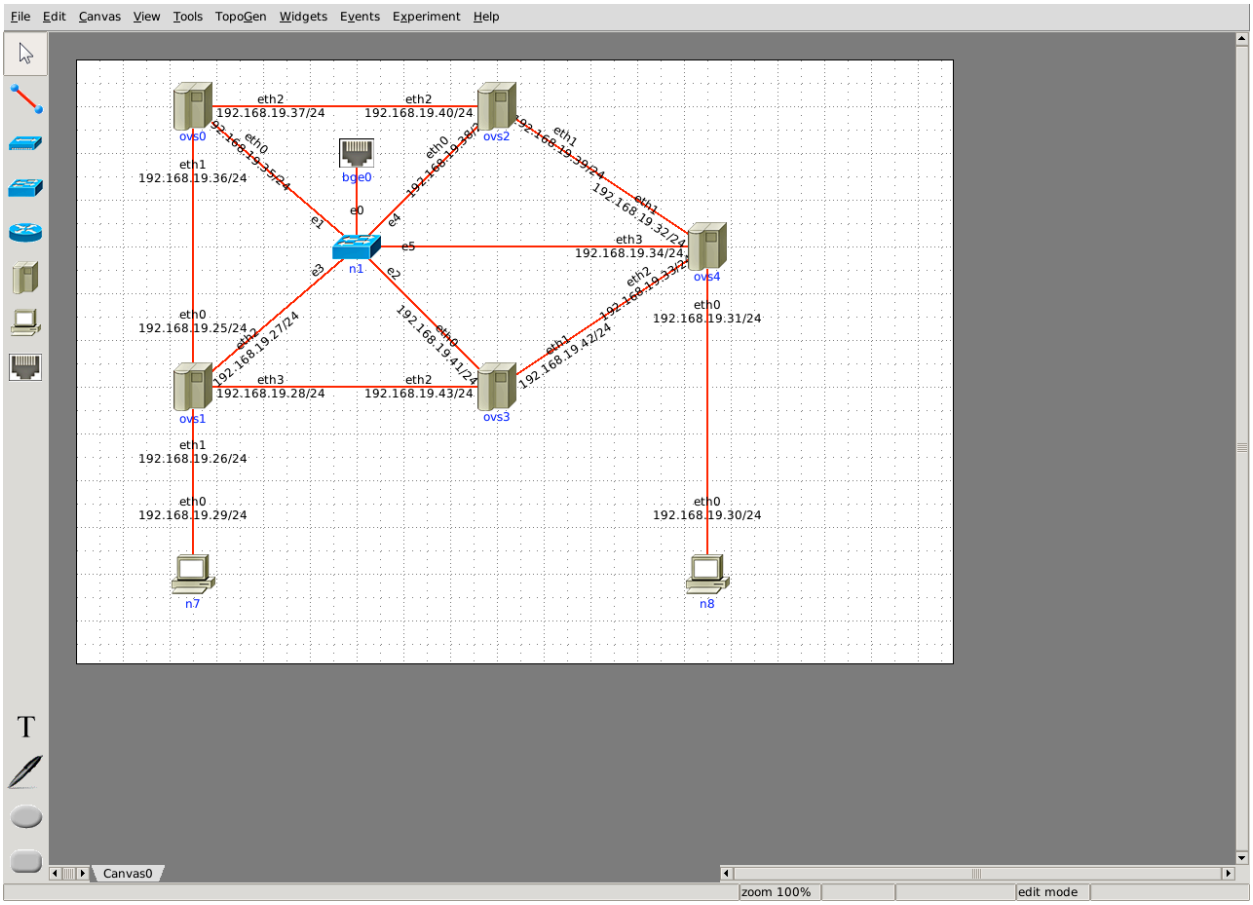
# 4. Upute za korištenje

## 4.1 Laboratorijsko okruženje

Za potrebe izrade ovog projekta korišten je prethodno podešeno računalo prema uputama koje salaze u [2], u četvrtom poglavlju. Računalo s tako podešenim simulatorom/emulatorom *IMUNES* je sposobno za rad s upravljačkim uređajem *OpenDaylight.*

Na jednom računalu je instaliran i podešen upravljački uređaj *OpenDaylight*. Drugo računalo, fizički odvojeno od prethodno spomenutog računala, koristi se za pokretanje simulatora/emulatora *IMUNES*. To računalo ima dva dodatna mrežna sučelja. Jedno od tih sučelja se koristi za spajanje komunikacijske mreže u simulatoru/emulatoru *IMUNES* s upravljačkim uređajom koji je pokrenut na drugom računalu.

Topologija mreže koja je korištena u simulatoru/emulatoru *IMUNES* je prikazana je na sljedećnoj slici.




**Slika 8. Topologija mreže**

Iz priložene slike topologije mreže možemo vidjeti da su svi komutatori spojeni na sučelje *bge0*. To sučelje je fizičko sučelje računala na kojem je pokrenut simulator/emulator *IMUNES*. Sučelje *bge0* je zatim spojeno na fizički komutator na koji je također spojeno računalo na kojem je pokrenut upravljački uređaj *OpenDaylight*. Na isti se fizički komutator spaja i lokalna mreža laboratorija na kojoj postoji *DHCP* poslužitelj kako bi računala dobila *IP* adrese koje su npr. potrebne prilikom postavljanja komutatora unutar *IMUNES-a* koji moraju znati *IP* adresu upravljačkog uređaja.

### 4.2 Povezivanje simulatora/emulatora IMUNES i upravljačkog uređaja OpenDaylight

Upravljački uređaj *OpenDaylight* i komunikacijska mreža unutar simulatora/emulatora *IMUNES*, prema OF specifikaciji, mogu komunicirati ukoliko se nakon pokretanja *IMUNES* eksperimenta, pokrene skripta *netinit2.py* iz [2]. Unutar te skripte se nalaze naredbe koje je potrebno pokrenuti na svakom mrežnom komutatoru unutar simulatora/emulatora *IMUNES* kako bi se podigao servis zadužen za razmjenu poruka s upravljačkim uređajem *OpenDaylight, Open vSwitch.* Nakon pokretanja te skripte, upravljački uređaj *OpenDaylight* i mrežni komutatori mogu razmjenjivati poruke definirane *OF* specifikacijom. Pokretanje skripte se radi na sljedeći način.

```
1.  $ python netinit2.py –c 192.168.19.153 –s 192.168.19.0/24
2.  $ python netinit2.py -kill
```

U gore navedenom primjeru, adresa kontrolera, te maska podmreže (engl. *Subnet Mask*) odgovaraju onima sa slike Slika 8. Prva se naredba koristi nakon što pokrenemo eksperiment unutar *IMUNES-a,* a ona pokreće servis *ovs-switch,* koji zapravo omogućuje komutatoru da komunicira s upravljačkim uređajem koristeći specifikaciju *OpenFlow,* na svim komutatorima. Druga se naredba koristi prije gašenja eksperimenta, a ona osigurava da se na svim komutatorima pravilo ugasi *ovs-switch* servis.

### 4.3 Primjene izvedenih modula u sklopu laborijatorijskog okruženja

Prije nego što možemo početi razvijati proširenja za upravljački uređaj *OpenDaylight* trebamo preuzeti primjerak upravljačkog uređaja na željeno računalo. To možemo napraviti na dva načina. Prvi je da preuzmemo kopiju upravljačkog uređaja koja se nalazi na [1]. Drugi je način da preuzmemo *git* repozitorij u kojem se nalaze sve datotke s izvornim tekstom upravljačkog uređaja. Preuzimanje izvornog teksta i stvaranje izvršne datoteke možemo napraviti izvršavanjem sljedećeg sljeda naredbi.

```
1.  $ git clone https://git.opendaylight.org/gerrit/p/controller.git
2.  $ cd ./controller/opendaylight/distribution/opendaylight/
3.  $ mvn clean install
```

Nakon što smo preuzeli izvorni tekst upravljačkog uređaja *OpenDaylight*, možemo početi s razvijanjem vlastitih proširenja. Najprije ćemo napraviti hijerarhijsku strukturu kazala pomoću alata *Maven*.

```
1.  $ cd ./controller/opendaylight/
2.  $ mkdir mystats
3.  $ mvn archetype:generate -DgroupId=com.example -DartifactId=mystats
    DarchetypeArtifactId=maven-archetype-quickstart -Dpackage=com.example.mystats -
    DinteractiveMode=false
```

Rezultat izvršavanja gornjeg sljeda naredbi dobijamo strukturu kazala kakva se koristi i kod samog upravljačkog uređaja *OpenDaylight*. Korijensko kazalo našeg proširenja je *./controller/opendaylight* unutar kojeg se nalazi *Maven* konfiguracijska datoteka *pom.xml*. U toj datoteci moramo navesti koja sve sučelja upravljačkog uređaja želimo koristiti kako bi ih alata *Maven* znao uključiti prilikom izgradnje.

Izvorni tekst proširenja pišemo unutar*./controller/opendaylight/mystats/main/src/com/example/mystats* kazala. Potrebno je dodati dvije datoteke, *Activator.java* i *MyStats.java*. Unutar datoteke *Activator.java* navodimo koje sve servise koristimo u proširenju i kako izlažemo sučelja proširenja. Unutar datoteke *MyStats.java* razvijamo proširenje.

Jednom kad smo napisali izvorni kod proširenja, vrijeme je da izgradimo izvršnu datoteku. Taj postupak prepuštamo alatu *Maven* tako da pokrenemo sljedeću naredbu.

```
1.  $ mvn clean install
```

Gornja će naredba najprije preuzeti sve pakete koji su potrebni za izgradnju izvršne datoteke s repozitorija koji je naveden u *pom.xml* datoteci korijenskog kazala proširenja. Ukoliko ne želimo preuzimati nove pakete već izgraditi izvršnu datoteku na temelju lokalno dostupnih paketa, izvršnu datoteku izgradimo pokretanjem sljedeće naredbe.

```
1.  $ mvn package
```

Rezultat obje naredbe će biti izvršna datoteka naziva *mystats_1.5.0-SNAPSHOT.jar* unutar kazala *./mystats/target/*.

Gore navedeni postupak, od stvaranja hijerarhijske structure kazala do stvaranja izgradnje izvršne datoteke, moramo ponoviti i za proširenje *StatsCollector* kao i za sučelje *IStatsCollector*, koje je definirano u zasebnom *Java* razredu. Sučelje *IStatsCollector* ne treba aktivacijsku datoteku *Activator.java* jer ono samo izlaže svoje usluge drugim proširenjima.

Nakon što smo izgradili izvršnu datoteku proširenja, možemo ispitati kako radi unutar upravljačkog uređaja *OpenDaylight*. Najprije pokrenemo sam upravljački uređaj sljedećom naredbom.

```
1. $ cd ./controller/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage/opendaylight
2. $ sudo ./run.sh
```

Nakon što se inicijaliziraju svi moduli upravljačkog uređaja, proširenje je najprije potrebno instalirati unutar *OSGi* konzole. To radimo pozivom naredbe *install* unutar *OSGi* konzole.

```
1. osgi> install file:/controller/opendaylight/mystats/target/mystats-1.5.0-SNAPSHOT.jar
2. osgi> start ID
```

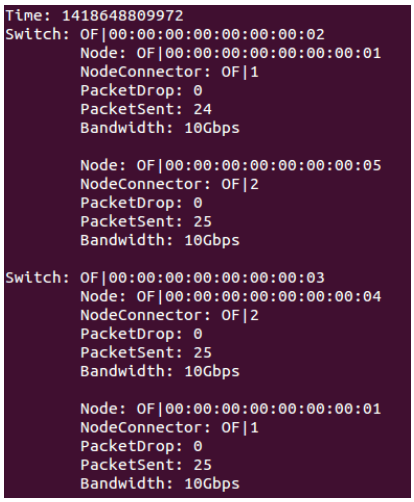
```
osgi> install file:/home/msantl/Desktop/git_opendaylight/controller/opendaylight/mystats/target/mystats-1.5.0-SNAPSHOT.jar
Bundle id is 280
RegisteredServices   null
ServicesInUse        null
LoaderProxy          com.example.mystats; bundle-version="1.5.0.SNAPSHOT"
Fragments            null
ClassLoader          null
LastModified         1418649032375
Headers               Bnd-LastModified = 1418648673246
 Build-Jdk = 1.7.0_65
 Built-By = msantl
 Bundle-Activator = com.example.mystats.Activator
 Bundle-ManifestVersion = 2
 Bundle-Name = mystats
 Bundle-SymbolicName = com.example.mystats
 Bundle-Version = 1.5.0.SNAPSHOT
 Created-By = Apache Maven Bundle Plugin
 Export-Package = com.example.mystats;uses:="org.opendaylight.controller.sal.core,org.apache.felix.dm,org.slf4j,org.opendaylight.controller.st
atisticsmanager,com.example.statscollector,org.opendaylight.controller.topologymanager,org.opendaylight.controller.switchmanager,org.opendayli
ght.controller.sal.utils";version="1.5.0.SNAPSHOT"
 Import-Package = com.example.statscollector;version="[1.5,2)",org.apache.felix.dm;version="[3.0,4)",org.opendaylight.controller.sal.core;vers
ion="[0.9,1)",org.opendaylight.controller.sal.flowprogrammer;version="[0.9,1)",org.opendaylight.controller.sal.match;version="[0.9,1)",org.ope
ndaylight.controller.sal.reader;version="[0.9,1)",org.opendaylight.controller.sal.utils;version="[0.9,1)",org.opendaylight.controller.statisti
csmanager;version="[0.6,1)",org.opendaylight.controller.switchmanager;version="[0.8,1)",org.opendaylight.controller.topologymanager;version="[
0.5,1)",org.slf4j;version="[1.7,2)"
 Manifest-Version = 1.0
 Tool = Bnd-1.50.0


Version              1.5.0.SNAPSHOT
BundleData           com.example.mystats_1.5.0.SNAPSHOT
BundleContext        null
BundleId             280
StartLevel           1
SymbolicName         com.example.mystats
KeyHashCode          280
StateChanging        null
BundleDescription    com.example.mystats_1.5.0.SNAPSHOT
Framework            org.eclipse.osgi.framework.internal.core.Framework@2ab3c8d4
ResolutionFailureException org.osgi.framework.BundleException: The bundle "com.example.mystats_1.5.0.SNAPSHOT [280]" could not be resolved
Revisions            [com.example.mystats_1.5.0.SNAPSHOT]
ProtectionDomain     null
Key                  280
Location             file:/home/msantl/Desktop/git_opendaylight/controller/opendaylight/mystats/target/mystats-1.5.0-SNAPSHOT.jar
State                2
Bundle                 280|Installed  |    1|com.example.mystats (1.5.0.SNAPSHOT)
```

**Slika 9. Instalacija proširenja unutar *OSGi* konzole**

Naredba *install* će ispisati podatke oko instaliranog proširenja između kojih se nalazi i identifikacijska oznaka. Nju ćemo korisitit kao refrentnu oznaku za naše proširenje. Naredbom *start ID* pokrećemo proširenje. Ako je sve u redu, unutar konzole bismo trebali vidjeti ispis proširenja. Proširenje zaustavljamo naredbom *stop ID*.

Nakon pokretanja proširenja, na izlaz *OSGi* konzole dobijamo ispis koji sadrži prikupljene statistike u sljedećem formatu.




**Slika 10. Format ispisa proširenja *MyStats***

Konačno, na dobivenom izlazu (Slika 10.) možemo vidjeti koje sve podatke proširenje *PODESI* dohvaća od upravljačkog uređaja *OpenDaylight.* U prvoj se liniji nalazi vremenska oznaka (engl. *timestamp*) koja označava vrijme u kojem su te statistike prikupljene. Nakon toga slijedi pet (broj komutatora u topologiji, a radi preglednosti prikazana samo jedna) grupa od kojih se svaka sastoji od sljedećih podataka. U prvom redu svake grupe stoji oznaka komutatora (*Switch: OF|00:00:00:00:00:00:00:02*) nakon koje slijedi onoliko podgrupa koliko taj komutator ima neposrednih susjeda. U topologiji na kojoj je testirano proširenje, komutator *OF|00:00:00:00:00:00:00:02* ima dva neposredna susjeda, *OF|00:00:00:00:00:00:00:01 i OF|00:00:00:00:00:00:00:05.* O svakom od tih komutatora znamo sljedeće informacije, koje također i ispisujemo. To su:

1. Oznaka susjednog komutatora kojim je trenutni komutator neposredno vezan (*OF|00:00:00:00:00:00:00:01*)
2. Oznaka vrata na kojima je ta veza ostvarena (*OF|1*)
3. Broj izgubljenih paketa (*0*)
4. Broj poslanih paketa (*24*)
5. Nazivna propusnost (*10Gbps*)

Imajući te informacije spremne, moguće je pronalaženje puteva vlastitim heuristikama koje koriste prikupljene statističke podatke. Pronalaženje optimalnih puteva, obavještavanje komutatora o novim putevima te opsežno ispitivanje kvalitete usluge takvog načina rada mreže ostalo je za budući rad.

## Literatura

[1] Upravljački uređaj *OpenDaylight*, OpenDaylight, [Online: http://www.opendaylight.org/software/downloads], datum zadnjeg pristupa: 21.12.2014.

[2] Ivan Vuk, "Programski upravljano prosljeđivanje medijskih tokova optimalnim mrežnim putevima", diplomski rad, FER, srpanj 2013.

[3] Daniel Romić, "Uspostava višemedijske sjednice kroz emuliranu programski upravljanu komunikacijsku mrežu", diplomski rad, FER, lipanj 2014.

[4] Matija Šantl, "Upravljački uređaji za programski upravljane komunikacijske mreže", seminarski rad, FER, svibanj 2014.

[5] Open Network Foundation, Software-defined networking: The new norm form networks, [Online: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf], Siječanj 2015., ONF White Paper

[6] Open Network Foundation, OpenFlow Switch Specifiacation (v.1.3.1), [Online: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf], Siječanj 2015., ONF White Paper

## APPENIX A – IZVORNI KÔD PROŠIRENJA MYSTATS (MYSTATS.JAVA)

```
1.  package com.example.mystats;
2.
3.  import java.util.Iterator;
4.  import java.util.Set;
5.  import java.util.List;
6.  import java.util.Map;
7.  import java.util.HashMap;
8.
9.  import com.example.statscollector.IStatsCollector;
10. import com.example.statscollector.Data;
11.
12. import org.opendaylight.controller.sal.utils.ServiceHelper;
13. import org.opendaylight.controller.sal.core.Node;
14. import org.opendaylight.controller.sal.core.NodeConnector;
15. import org.opendaylight.controller.sal.core.Edge;
16. import org.opendaylight.controller.sal.core.Host;
17. import org.opendaylight.controller.topologymanager.ITopologyManager;
18. import org.slf4j.Logger;
19. import org.slf4j.LoggerFactory;
20.
21. /*
22.  * MyStats
23.  *  statisticke podatke koje prikuplja StatsCollector te ih nadopunjuje
24.  *  podacima o topologiji mreze
25.  */
26. public class MyStats{
27.     /* log - za spremanje bitnih dogadaja u datoteku s izvjestajem */
28.     private static final Logger log = LoggerFactory.getLogger(MyStats.class);
29.
30.     public MyStats() { }
31.
32.     /* init - metoda koja se poziva nakon ucitavanja modula u OSGi radni okvir*/
33.     void init() {
34.         log.debug("INIT called!");
35.     }
36.
37.     /* destroy - metoda koja se poziva nakon micanja modula iz OSGi radnog okvira */
38.     void destroy() {
39.         log.debug("DESTROY called!");
40.     }
41.
42.     /* start - metoda koja se poziva nakon pokretanja modula u OSGi radnom okviru*/
43.     void start() {
44.         log.debug("START called!");
45.         getFlowStatistics();
46.     }
47.
48.     /* stop - metoda koja se poziva nakon zaustavljanja modula u OSGi radnom okviru*/
49.     void stop() {
50.         log.debug("STOP called!");
51.     }
52.
```

```
53.         /* pomocna metoda koja ispisuje sadrzaj Mape bez obzira na tipove podataka */
54.         void printMap(Map mp) {
55.             Iterator it = mp.entrySet().iterator();
56.             System.out.println("----------");
57.             while (it.hasNext()) {
58.                 Map.Entry pairs = (Map.Entry)it.next();
59.                 try {
60.                     System.out.println(pairs.getKey() + " = " + pairs.getValue());
61.                 } catch (Exception e) {
62.                     System.out.println("n/a");
63.                 }
64.             }
65.             System.out.println("----------");
66.         }
67.
68.         /* getFlowStatistics - glavna metoda ove klase. Koristi sucelje klase
69.          * StatsCollector kako bi dohvatila prikupljene podatke. Nakon toga te
70.          * podatke nadopunjuje informacijama o topologiji mreze. */
71.         void getFlowStatistics() {
72.             /* naziv grupe OF komutatora */
73.             String containerName = "default";
74.
75.             /* struktura podataka u koju spremamo sve podatke */
76.             Map<Node, List<Data> > edge = new HashMap();
77.             /* struktura podataka u koju spremamo statisticke podatke */
78.             Map<Long, Map<Node, List<Data> > > res;
79.
80.             /* dohvacamo instancu klase TopologyManager koja pruza sucelje za
81.              * dohvacanje trenutne topologije mreze */
82.             ITopologyManager topologyManager = (ITopologyManager) ServiceHelper
83.                 .getInstance(ITopologyManager.class, containerName, this);
84.
85.             /* dohvacamo instancu klase StatsCollector koja pruza sucelje za
86.              * dohvacanje prikupljenih statistickih podataka */
87.             IStatsCollector statsCollector = (IStatsCollector) ServiceHelper
88.                 .getInstance(IStatsCollector.class, containerName, this);
89.
90.             /* provjeri da je StatsCollector pokrenut*/
91.             if (statsCollector != null) {
92.                 /* dohvati prikupljene podatke */
93.                 res = statsCollector.getStats();
94.
95.                 /* uzmi najnoviji podatak */
96.                 Long latest = null;
97.                 for (Long timestamp : res.keySet()) {
98.                     if (latest == null || timestamp > latest) {
99.                         edge = res.get(timestamp);
100.                        latest = timestamp;
101.                    }
102.                }
103.
104.                System.out.println("Time: " + latest);
105.            } else {
106.                /* ispisi pogresku i izadi */
107.                System.out.println("StatsCollector not present!");
```

```
108.            return;
109.        }
110.
111.        /* dohvati trenutnu topologiju mreze */
112.        Map<Node,Set<Edge>> topology = topologyManager.getNodeEdges();
113.
114.        /* za svaki cvor za koji imamo prikupljene statisticke podatke */
115.        for (Node key : edge.keySet()) {
116.            /* za svakog neposrednog susjeda cvora*/
117.            for (Edge e : topology.get(key)) {
118.                // get topology information about that node
119.
120.                /* dohvati podatke njihovoj povezanosti */
121.                NodeConnector tail_nc = e.getTailNodeConnector();
122.                NodeConnector head_nc = e.getHeadNodeConnector();
123.
124.                Node tail = tail_nc.getNode();
125.                Node head = head_nc.getNode();
126.
127.                /* osvjezi strukturu podataka s informacijom o poveznasnoti */
128.                List<Data> neighbours = edge.get(head);
129.                for (Data n : neighbours) {
130.                    if (head_nc.equals(n.getNodeConnector())) {
131.                        n.setNode(tail);
132.                    }
133.                }
134.            }
135.        }
136.
137.        /* za svaki cvor za koji imamo prikupljene statisticke podatke */
138.        for (Node key : edge.keySet()) {
139.            /* za svaki zapis koji imamo za taj covr */
140.            for (Data n : edge.get(key)) {
141.                /* dohvati listu hostova koji su spojeni na neki od portova */
142.                List<Host> hosts = topologyManager
143.                    .getHostsAttachedToNodeConnector(n.getNodeConnector());
144.
145.                /* osvjezi strukturu podataka s informacijom o hostovima */
146.                if (hosts != null) {
147.                    n.setHosts(hosts);
148.                }
149.            }
150.        }
151.
152.        /* ispisi prikupljene informacije */
153.        for (Node key : edge.keySet()) {
154.            System.out.println("Switch: " + key.toString());
155.
156.            for (Data data : edge.get(key)) {
157.                if (data.getNode() != null) {
158.                    System.out.println("\tNode: " + data.getNode().toString());
159.                }
160.                if (data.getHosts() != null) {
161.                    System.out.print("\tHosts: ");
162.                    for (Host h : data.getHosts()) {
```

```
163.                         System.out.print(h.getNetworkAddressAsString() + ", ");
164.                     }
165.                     System.out.println("");
166.                 }
167.
168.                 System.out.println("\tNodeConnector: " +
169.                         data.getNodeConnector().getNodeConnectorIdAsString());
170.                 System.out.println("\tPacketDrop: " + data.getPacketDrop());
171.                 System.out.println("\tPacketSent: " + data.getPacketSent());
172.
173.                 System.out.println("\tTx count: " + data.getTxCount());
174.                 System.out.println("\tRx count: " + data.getRxCount());
175.                 System.out.println("\tBandwidth: " + data.getBandwidth());
176.                 System.out.println("");
177.             }
178.         }
179.
180.         /* TODO run A* on collected data */
181.         /* TODO install a new flow using flowManager */
182.
183.         return;
184.     }
185. }
```

## APPENIX B – IZVORNI KÔD AKTIVATORA PROŠIRENJA MYSTATS (ACTIVATOR.JAVA)

```
1.  package com.example.mystats;
2.
3.  import org.apache.felix.dm.Component;
4.  import org.opendaylight.controller.sal.core.ComponentActivatorAbstractBase;
5.  import org.slf4j.Logger;
6.  import org.slf4j.LoggerFactory;
7.
8.  public class Activator extends ComponentActivatorAbstractBase {
9.      protected static final Logger logger = LoggerFactory
10.             .getLogger(Activator.class);
11.
12.     public void init() {
13.     }
14.
15.     public void destroy() {
16.     }
17.
18.     public Object[] getImplementations() {
19.         Object[] res = { MyStats.class };
20.         return res;
21.     }
22.
23.     public void configureInstance(Component c, Object imp, String containerName) {
24.         if (imp.equals(MyStats.class)) {
25.             // export the service
26.             c.setInterface(new String[] { MyStats.class.getName() }, null);
27.         }
28.     }
29. }
```

## APPENIX C – IZVORNI KÔD KONFIGURACIJSKE DATOTEKE PROŠIRENJA MYSTATS (POM.XML)

```
1.  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
2.     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
3.     <modelVersion>4.0.0</modelVersion>
4.     <parent>
5.        <groupId>org.opendaylight.controller</groupId>
6.        <artifactId>commons.opendaylight</artifactId>
7.        <version>1.5.0-SNAPSHOT</version>
8.        <relativePath>../../../controller/opendaylight/commons/opendaylight</relativePath>
9.     </parent>
10.
11.    <groupId>com.example</groupId>
12.    <artifactId>mystats</artifactId>
13.    <name>mystats</name>
14.    <url>http://maven.apache.org</url>
15.    <packaging>bundle</packaging>
16.
17.    <build>
18.       <plugins>
19.          <plugin>
20.             <groupId>org.apache.felix</groupId>
21.             <artifactId>maven-bundle-plugin</artifactId>
22.             <version>2.3.6</version>
23.             <extensions>true</extensions>
24.             <configuration>
25.                <instructions>
26.                   <Import-Package>
27.                      org.opendaylight.controller.sal.core,
28.                      org.opendaylight.controller.topologymanager,
29.                      org.opendaylight.controller.sal.utils,
30.                      org.opendaylight.controller.sal.reader,
31.                      org.opendaylight.controller.sal.flowprogrammer,
32.                      org.opendaylight.controller.sal.match,
33.                      org.slf4j,
34.                      org.apache.felix.dm,
35.                      com.example.statscollector
36.                   </Import-Package>
37.                   <Bundle-Activator>
38.                      com.example.mystats.Activator
39.                   </Bundle-Activator>
40.                   <Export-Package>
41.                      com.example.mystats
42.                   </Export-Package>
43.                </instructions>
44.             </configuration>
45.          </plugin>
46.       </plugins>
47.    </build>
48.    <dependencies>
49.       <dependency>
50.          <groupId>junit</groupId>
```

```
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.opendaylight.controller</groupId>
            <artifactId>sal</artifactId>
        </dependency>
        <dependency>
            <groupId>org.opendaylight.controller</groupId>
            <artifactId>topologymanager</artifactId>
        </dependency>
        <dependency>
            <groupId>com.example</groupId>
            <artifactId>statscollector</artifactId>
            <version>1.5.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
</project>
```

## APPENIX D – IZVORNI KÔD SUČELJA ISTATSCOLLECTOR (ISTATSCOLLECTOR.JAVA)

```
1.  package com.example.statscollector;
2.
3.  import java.util.List;
4.  import java.util.Map;
5.
6.  import org.opendaylight.controller.sal.core.Node;
7.
8.  /* IStatsCollector - sucelje koje se pruza drugim paketima unutar
9.   * OpenDaylight-a*/
10. public interface IStatsCollector {
11.     /* getStats - metoda za dohvat prikupljenih statistickih podataka */
12.     public Map<Long, Map<Node, List<Data> > > getStats();
13.     /* getTest - metoda za ispitivanje ispravnosti sucelja */
14.     public List<String> getTest();
15. }
```

## APPENIX E – IZVORNI KÔD STRUKTURE PODATAKA DATA (DATA.JAVA)

```
1.  package com.example.statscollector;
2.
3.  import java.util.List;
4.  import java.util.ArrayList;
5.
6.  import org.opendaylight.controller.sal.core.Node;
7.  import org.opendaylight.controller.sal.core.NodeConnector;
8.  import org.opendaylight.controller.sal.core.Host;
9.
10. /* Data - struktura podataka koja je namijenjena za spremanje i dohvat
11.  * statistickih podataka te podataka o topologiji mreze. */
12. public class Data {
13.     /* node - oznaka susjednog cvora */
14.     private Node node;
15.     /* host - lista hostova spojenih na cvor */
16.     private List<Host> host;
17.     /* nc - oznaka porta na kojem je ostvarena veza */
18.     private NodeConnector nc;
19.     /* pkt_drop - broj izgubljenih paketa */
20.     private long pkt_drop;
21.     /* pkt_sent - broj poslanih paketa */
22.     private long pkt_sent;
23.     /* tx_count - broj poslanih okteta */
24.     private long tx_count;
25.     /* rx_count - broj primljenih okteta */
26.     private long rx_count;
27.     /* bandwidth - nazivna vrijednost propusnosti */
28.     private String bandwidth;
29.
30.     public Data() {
31.         this.node = null;
32.         this.host = null;
33.         this.nc = null;
34.         this.bandwidth = null;
35.     }
36.
37.     public void setNode(Node node) {this.node = node;}
38.     public void setNodeConnector(NodeConnector nc) {this.nc = nc;}
39.     public void setPacketDrop(long pkt_drop) {this.pkt_drop = pkt_drop;}
40.     public void setPacketSent(long pkt_sent) {this.pkt_sent = pkt_sent;}
41.     public void setTxCount(long tx_count) {this.tx_count = tx_count;}
42.     public void setRxCount(long rx_count) {this.rx_count = rx_count;}
43.     public void setBandwidth(String bandwidth) {this.bandwidth = bandwidth;}
44.     public void setHosts(List<Host> hosts) {
45.         this.host = new ArrayList();
46.         for (Host h : hosts) {
47.             this.host.add(h);
48.         }
49.     }
50.
51.     public Node getNode() {return this.node;}
52.     public NodeConnector getNodeConnector() {return this.nc;}
```

```
53.     public long getPacketDrop() {return this.pkt_drop;}
54.     public long getPacketSent() {return this.pkt_sent;}
55.     public long getTxCount() {return this.tx_count;}
56.     public long getRxCount() {return this.rx_count;}
57.     public String getBandwidth() {return this.bandwidth;}
58.     public List<Host> getHosts() {return this.host;}
59.
60.     @Override
61.     public String toString() {
62.         String ret = "Data";
63.         if (getNode() != null) {
64.             ret += "\tNode: " + getNode().toString();
65.         }
66.         if (getHosts() != null) {
67.             ret += "\tHosts: ";
68.             for (Host h : getHosts()) {
69.                 ret += h.getNetworkAddressAsString() + ", ";
70.             }
71.         }
72.
73.         if (getNodeConnector() != null) {
74.             ret += "\tNodeConnector: " + getNodeConnector().getNodeConnectorIdAsString();
75.         }
76.
77.         ret += "\tPacketDrop: " + getPacketDrop();
78.         ret += "\tPacketSent: " + getPacketSent();
79.
80.         ret += "\tTx count: " + getTxCount();
81.         ret += "\tRx count: " + getRxCount();
82.
83.         if (getBandwidth() != null) {
84.             ret += "\tBandwidth: " + getBandwidth();
85.         }
86.         return ret;
87.     }
88. }
```

## APPENIX F – IZVORNI KÔD KONFIGURACIJSKE DATOTEKE SUČELJA ISTATSCOLLECTOR (POM.XML)

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
3.    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
    4.0.0.xsd">
4.    <modelVersion>4.0.0</modelVersion>
5.    <parent>
6.      <groupId>org.opendaylight.controller</groupId>
7.      <artifactId>commons.opendaylight</artifactId>
8.      <version>1.5.0-SNAPSHOT</version>
9.      <relativePath>../../commons/opendaylight</relativePath>
10.   </parent>
11.
12.   <groupId>com.example</groupId>
13.   <artifactId>statscollector</artifactId>
14.   <url>http://maven.apache.org</url>
15.   <packaging>bundle</packaging>
16.
17.   <build>
18.     <plugins>
19.       <plugin>
20.         <groupId>org.apache.felix</groupId>
21.         <artifactId>maven-bundle-plugin</artifactId>
22.         <version>2.3.6</version>
23.         <extensions>true</extensions>
24.         <configuration>
25.           <instructions>
26.             <Import-Package>
27.               org.opendaylight.controller.sal.core,
28.               org.slf4j,
29.               org.apache.felix.dm
30.             </Import-Package>
31.             <Export-Package>
32.               com.example.statscollector
33.             </Export-Package>
34.           </instructions>
35.         </configuration>
36.       </plugin>
37.     </plugins>
38.   </build>
39.   <dependencies>
40.     <dependency>
41.       <groupId>org.opendaylight.controller</groupId>
42.       <artifactId>sal</artifactId>
43.     </dependency>
44. </dependencies>
45. </project>
```

## APPENIX G – IZVORNI KÔD PROŠIRENJA STATSCOLLECOTR (STATSCOLLECTORIMPL.JAVA)

```
1.  package com.example.statscollector.internal;
2.
3.  import java.util.List;
4.  import java.util.ArrayList;
5.
6.  import java.util.Map;
7.  import java.util.HashMap;
8.  import java.util.Collections;
9.
10. import java.util.LinkedHashMap;
11. import java.util.Map.Entry;
12. import java.util.Iterator;
13. import java.util.Date;
14.
15. import java.io.FileNotFoundException;
16. import java.io.IOException;
17. import java.io.ObjectInputStream;
18.
19. import com.example.statscollector.IStatsCollector;
20. import com.example.statscollector.Data;
21.
22. import org.opendaylight.controller.sal.utils.IObjectReader;
23. import org.opendaylight.controller.sal.utils.ServiceHelper;
24. import org.opendaylight.controller.sal.core.Node;
25. import org.opendaylight.controller.sal.core.Property;
26. import org.opendaylight.controller.sal.core.NodeConnector;
27. import org.opendaylight.controller.sal.reader.NodeConnectorStatistics;
28. import org.opendaylight.controller.statisticsmanager.IStatisticsManager;
29. import org.opendaylight.controller.switchmanager.ISwitchManager;
30. import org.opendaylight.controller.switchmanager.Switch;
31. import org.slf4j.Logger;
32. import org.slf4j.LoggerFactory;
33.
34. /* StatsCollectorImpl - ostvarenje sucelja IStatsCollector. StatsCollectorImpl
35.  * pokrece dretvu koja svake dvije sekunde sprema statisticke podatke koje
36.  * dohvaca od upravljackog uredaja preko sucelja IStatisticsManager u lokalni
37.  * LRU cache od 1000 zapisa. Implementira metodu getStats koju onda mogu
38.  * pozivati drugi paketi unutar OpenDaylight-a kako bi dohvatili te podatke */
39. public class StatsCollectorImpl implements IStatsCollector, IObjectReader{
40.     /* log - za spremanje bitnih dogadaja u datoteku s izvjestajem */
41.     private static final Logger log = LoggerFactory.getLogger(StatsCollectorImpl.class);
42.     /* statsCollector - dretva koja je zaduzena za periodicki dohvat
43.      * statistickih podataka od upravljckog uredaja*/
44.     private Thread statsCollector;
45.     /* Cache - lokalni LRU cache */
46.     private final Map<Long, Map<Node, List<Data> > > Cache = Collections
47.         .synchronizedMap(new LRUCache<Long, Map<Node, List<Data> > >(1000));
48.
49.     /* LRUCache - ostvarenje LRU prirucne memorije */
50.     private class LRUCache <K, V> extends LinkedHashMap <K, V> {
51.         /* capacity - kapacitet prirucne memorije */
52.         private final int capacity;
```

```
53.         public LRUCache(int capacity) {
54.             super(capacity+1, 1.0f, true);
55.             this.capacity = capacity;
56.         }
57.
58.         @Override
59.         protected boolean removeEldestEntry(final Map.Entry<K, V> eldest) {
60.             return (super.size() > this.capacity);
61.         }
62.     }
63.
64.     /* pomocna metoda koja ispisuje sadrzaj Mape bez obzira na tipove podataka */
65.     void printMap(Map mp) {
66.         Iterator it = mp.entrySet().iterator();
67.         System.out.println("----------------------");
68.         while (it.hasNext()) {
69.             Map.Entry pairs = (Map.Entry)it.next();
70.             System.out.println(pairs.getKey() + " = " + pairs.getValue());
71.         }
72.         System.out.println("----------------------");
73.     }
74.
75.     public StatsCollectorImpl() {}
76.
77.     /* init - metoda koja se poziva nakon ucitavanja modula u OSGi radni okvir*/
78.     void init() {
79.         log.debug("INIT called!");
80.         /* inicijaliziraj dretvu statsCollector */
81.         statsCollector = new Thread(new StatsCollectorThread());
82.     }
83.
84.     /* destroy - metoda koja se poziva nakon micanja modula iz OSGi radnog okvira */
85.     void destroy() {
86.         log.debug("DESTROY called!");
87.     }
88.
89.     /* start - metoda koja se poziva nakon pokretanja modula u OSGi radnom okviru*/
90.     void start() {
91.         log.debug("START called!");
92.         /* pokreni dretvu statsCollector */
93.         statsCollector.start();
94.     }
95.
96.     /* stop - metoda koja se poziva nakon zaustavljanja modula u OSGi radnom okviru*/
97.     void stop() {
98.         log.debug("STOP called!");
99.         /* zaustavi dretvu statsCollector */
100.        statsCollector.interrupt();
101.    }
102.
103.    /* getCurrentStatistics - glavna metoda ove klase. Koristi se za dohvat
104.     * trenutnih statistickih podataka od upravljackog uredaja. Koristi sucelje
105.     * klasa StatsManager i SwitchManager kako bi prikupila podatke oko
106.     * postojecih cvorova i njihovih brojaca */
107.    Map<Node, List<Data> > getCurrentStatistics() {
```

```
108.    /* naziv grupe OF komutatora */
109.    String containerName = "default";
110.    /* naziv polja u kojem je spremljen podatak o propusnosti*/
111.    String propertyName = "bandwidth";
112.
113.    /* struktura podataka u koju spremamo rezultat */
114.    Map<Node, List<Data> > edge = new HashMap();
115.
116.    /* dohvacamo instancu klase StatisticsManager koja pruza sucelje za
117.     * dohvat statistickih podataka */
118.    IStatisticsManager statsManager = (IStatisticsManager) ServiceHelper
119.       .getInstance(IStatisticsManager.class, containerName, this);
120.
121.    /* dohvacamo instancu klase SwitchManager  koja pruza sucelje za dohvat
122.     * informacija o postojecim komutatorima */
123.    ISwitchManager switchManager = (ISwitchManager) ServiceHelper
124.       .getInstance(ISwitchManager.class, containerName, this);
125.
126.    /* za svaki komutator u mrezi */
127.    for (Switch swc : switchManager.getNetworkDevices()) {
128.       /* dohvati statisticke podatke o njemu */
129.       Node node = swc.getNode();
130.       // get stats about every nodeconnector from that node
131.       List<NodeConnectorStatistics> stat = statsManager
132.          .getNodeConnectorStatistics(node);
133.
134.       Map<NodeConnector, Data> node_data = new HashMap();
135.
136.       /* za svaki port mreznog komutatora */
137.       for (NodeConnector nc : swc.getNodeConnectors()) {
138.          /* dohvati podatke o nazivnim vrijednostima */
139.          Map<String,Property> mp =
140.             switchManager.getNodeConnectorProps(nc);
141.
142.          /* spremi rezultat u strukturu podataka Data */
143.          Data data = new Data();
144.
145.          // update node connector entry
146.          data.setNodeConnector(nc);
147.          // update bandwidth entry
148.          data.setBandwidth(mp.get(propertyName).getStringValue());
149.
150.          node_data.put(nc, data);
151.       }
152.
153.       /* za svaki port i pripadajuce statisticke podatke */
154.       for (NodeConnectorStatistics ncs : stat) {
155.          /* dohvati postojeci zapis */
156.          Data data = node_data.get(ncs.getNodeConnector());
157.
158.          if (data == null) {
159.             /* ignoriraj port na kojem je upravljacki uredaj */
160.             continue;
161.          }
162.
```

```
163.             /* dohvati i spremi statisticke podatke */
164.             long sent = ncs.getReceivePacketCount() + ncs.getTransmitPacketCount();
165.             long drop = ncs.getReceiveDropCount() + ncs.getTransmitDropCount();
166.
167.             // update packet drop entry
168.             data.setPacketDrop(drop);
169.             // update packet sent entry
170.             data.setPacketSent(sent);
171.
172.             // update transmit byte count entry
173.             data.setTxCount(ncs.getTransmitByteCount());
174.             // update receive byte count entry
175.             data.setRxCount(ncs.getReceiveByteCount());
176.
177.             node_data.put(ncs.getNodeConnector(), data);
178.          }
179.
180.          /* pohrani prikupljene podatke o mreznom komutatoru u rezultat */
181.          List<Data> list_data = new ArrayList();
182.
183.          for (NodeConnector key : node_data.keySet()) {
184.             // move full stats info in a list
185.             list_data.add(node_data.get(key));
186.          }
187.          // update result map
188.          edge.put(node, list_data);
189.       }
190.       return edge;
191.    }
192.
193.    /* StatsCollectorThread - dretva zaduzena za periodicko prikupljanje
194.     * statistickih podataka */
195.    private class StatsCollectorThread implements Runnable {
196.       @Override
197.       public void run() {
198.          while (true) {
199.             try {
200.                Thread.sleep(2000);
201.                /* dohvati trenutnu vremensku oznaku */
202.                Date date = new java.util.Date();
203.                /* dohvati statisticke podatke */
204.                Map<Node, List<Data> > res = getCurrentStatistics();
205.                /* pohrani rezultat u prirucnu memoriju */
206.                Cache.put(date.getTime(), res);
207.             } catch (InterruptedException e) {
208.                break;
209.             } catch (Exception e) {
210.                System.out.println("Something went wrong");
211.             }
212.          }
213.       }
214.    }
215.
216.    /* getStats - sucelje za dohvat statistickih podataka */
217.    @Override
```

```
218.    public Map<Long, Map<Node, List<Data> > > getStats() {
219.        return new HashMap<Long, Map<Node, List<Data> > >(Cache);
220.    }
221.
222.    /* getTest - sucelje za ispitivanje ispravnosti */
223.    @Override
224.    public List<String> getTest() {
225.        List<String> ret = new ArrayList();
226.
227.        ret.add("test1");
228.        ret.add("test2");
229.        ret.add("test3");
230.
231.        return ret;
232.    }
233.
234.    /* readObject - sucelje koje omogucuje prijenost podataka prema drugima
235.     * klasama */
236.    @Override
237.    public Object readObject(ObjectInputStream ois)
238.        throws FileNotFoundException, IOException, ClassNotFoundException {
239.        return ois.readObject();
240.    }
241.}
```

## APPENIX H – IZVORNI KÔD AKTIVATORA PROŠIRENJA STATSCOLLECTOR (ACTIVATOR.JAVA)

```
1.  package com.example.statscollector.internal;
2.
3.  import org.apache.felix.dm.Component;
4.  import org.opendaylight.controller.sal.core.ComponentActivatorAbstractBase;
5.  import org.slf4j.Logger;
6.  import org.slf4j.LoggerFactory;
7.
8.  import com.example.statscollector.IStatsCollector;
9.
10. public class Activator extends ComponentActivatorAbstractBase {
11.     protected static final Logger logger = LoggerFactory
12.             .getLogger(Activator.class);
13.
14.     public void init() {
15.     }
16.
17.     public void destroy() {
18.     }
19.
20.     public Object[] getImplementations() {
21.         Object[] res = { StatsCollectorImpl.class };
22.         return res;
23.     }
24.
25.     public void configureInstance(Component c, Object imp, String containerName) {
26.         if (imp.equals(StatsCollectorImpl.class)) {
27.             // export the service
28.             c.setInterface(new String[] { IStatsCollector.class.getName() }, null);
29.
30.         }
31.     }
32. }
```

## APPENIX I – IZVORNI KÔD KONFIGURACIJSKE DATOTEKE PROŠIRENJA STATSCOLLECTOR (POM.XML)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>org.opendaylight.controller</groupId>
      <artifactId>commons.opendaylight</artifactId>
      <version>1.5.0-SNAPSHOT</version>
      <relativePath>../../commons/opendaylight</relativePath>
   </parent>

   <groupId>com.example</groupId>
   <artifactId>statscollector.implementation</artifactId>
   <url>http://maven.apache.org</url>
   <packaging>bundle</packaging>

   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.felix</groupId>
            <artifactId>maven-bundle-plugin</artifactId>
            <version>2.3.6</version>
            <extensions>true</extensions>
            <configuration>
               <instructions>
                  <Import-Package>
                     org.opendaylight.controller.sal.core,
                     org.opendaylight.controller.statisticsmanager,
                     org.opendaylight.controller.switchmanager,
                     org.opendaylight.controller.sal.utils,
                     org.opendaylight.controller.sal.reader,
                     org.slf4j,
                     org.apache.felix.dm,
                     com.example.statscollector
                  </Import-Package>
                  <Bundle-Activator>
                     com.example.statscollector.internal.Activator
                  </Bundle-Activator>
               </instructions>
            </configuration>
         </plugin>
      </plugins>
   </build>
   <dependencies>
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>3.8.1</version>
         <scope>test</scope>
      </dependency>
      <dependency>
```

```
            <groupId>org.opendaylight.controller</groupId>
            <artifactId>sal</artifactId>
        </dependency>
         <dependency>
            <groupId>org.opendaylight.controller</groupId>
            <artifactId>statistics.northbound</artifactId>
        </dependency>
        <dependency>
            <groupId>org.opendaylight.controller</groupId>
            <artifactId>statisticsmanager</artifactId>
        </dependency>
        <dependency>
            <groupId>com.example</groupId>
            <artifactId>statscollector</artifactId>
            <version>1.5.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
</project>
```