Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Megyeri Csaba

HIBRID SDN HÁLÓZATI MEGOLDÁSOK VIZSGÁLATA

Diplomaterv

KONZULENS

Dr. Zsóka Zoltán

BUDAPEST, 2017

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott Megyeri Csaba, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2017. 12. 17.

...…….

Megyeri Csaba

Kivonat

A dolgozatom során a Software Defined Networking fejlődésével kapcsolatos paradigma, a hibrid SDN hálózatok vizsgálatával foglalkoztam. Ez a témakör a hálózati fejlődés szempontjából egy igen fontos fordulópont, ugyanis egyre több olyan kutatás van, amely ebben az irányban nyit meg új területeket, illetve a nagyobb szolgáltatók is egyre inkább foglalkoznak azzal, hogy a jelenlegi hálózatukat tovább fejleszthessék, ezáltal jobb teljesítményt érhessenek el akár a jelenlegi infrastruktúrával, akár új elemek használatával.

A kutatások jelenlegi állását vizsgálva még nyitott téma, hogy milyen módszerekkel, eszközökkel lehet költséghatékonyan a lehető legjobb eredményt elérni. Feladatom volt, hogy megvizsgáljak néhány jelenleg ajánlásként létező módszert, illetve közelebbről is megismerkedjek az SDN paradigma által kínált lehetőségekkel, ezeket egy olyan hálózaton teszteljem, amely a lehető legközelebb áll egy olyan elrendezéshez, ahol ténylegesen fizikailag összekötött eszközök dolgoznak együtt. Ebben az esetben ugyanis nem triviális, hogy hogyan oldjuk meg a hálózati eszközök közötti kommunikációt, milyen protokollokat használjunk, illetve milyen módon lehet ezeket tesztelni. A célkitűzések során olyan jellegű feladatok merültek fel, melyek a témakör minél nagyobb részét lefedték, így egyrészt a már említett hálózati megvalósítás mellett az esetleges újításokhoz tartozó implementációs lépések vizsgálatát is megejtettem. A legfőbb cél az volt, hogy ne egy olyan virtualizációt valósítsak meg amikor mindent egy számítógépen virtualizálunk, hanem minél közelebb kerüljek a fizikailag összekötött hálózatokhoz.

A dolgozat első felében felvázolok néhány esetet a hibrid csomópontos megoldások közül, majd áttérek arra az esetre, amikor úgy nevezett legacy eszközökkel való együttműködést vizsgálunk. A munkám legnagyobb része a hálózat lépésről lépésre való kialakítása, illetve a virtuális világtól való elszakítása. A dolgozat végén egy specifikációt közlök, melynek segítségével legacy eszközök és SDN-ben található eszközök egymás között routing információt oszthatnak meg.

Abstract

During my Thesis I was engaged in experimenting with hybrid SDN, which is a part of the evolution of Software Defined Networking. This topic is a really important turning point in the evolution of networking, more and more researches are made which open up new part of this topic and the main providers are also more and more involved because they will always want to upgrade their networking capabilities, thus improving their performance with the actual infrastucture or with using newer technologies.

Examining the current researches, it's still an open question what methods and tools are the best way to achieve a cost-effective result. It was my duty to look into some of these researches and to get more knowledge on the oppurtunities offered by this SDN paradigm and to try out some of them on a network that is as close to a physically connected network as possible. In a scenario like that it is not trivial how we can get our networking devices to communicate with each other, what protocolos we should use, or how can we test these layouts. During establishing the goals of this project, I found tasks which covered broadly the topic, so beside making a working network I had to make steps toward implementing a brand new solution. The main goal of this project was to create a network as close to a physically connected network as possible, so I could not use a method like virtualize everything on a single host.

In the first part of my Thesis I present some cases of hybrid networks where the hybrid word means thath hybrid nodes are used, and then I continue with a case where our SDN devices communicate with so-called legacy devices. The majority of my work consisted of creating the planned network step-by-step and to make it less virtualized. In the last section I present a specification for a system in which legacy devices and SDN devices can use routing protocols together and so share information about the whole network.

Rövidítések

SDN	Software Defined Networking
IP	Internet Protocol
QoS	Quality of Service
L2, L3	Layer 2, Layer 3
sw, hw	software, hardware
VLAN	Virtual LAN
SAL	Service Abstraction Layer
ARP	Address Resolution Protocol
ONOS	Open Network Operating System
API	Application programming interface
MAC	Media Access Control
MPLS	Multiprotocol Label Switching
OFCS	Openflow capable switch
OSHI	Open Source Hybrid IP/SDN
ISP	IP Service Provider
OSPF	Open Shortest Path First
CR, PE, CE	Core Router, Provider Edge, Customer Edge
VPN	Virtual Private Network
WAN	Wide Area Network
IP VLL	IP Virtual Leased Line
L2 PW	L2 Pseudo Wire
Mantoo	Management tools for SDN
JSON	JavaScript Object Notation

SSH	Secure Shell
TTL	Time-To-Live
ICMP	Internet Control Message Protocol
NAT	Network Address Translation
VM	Virtual Machine
OVS	Openvswitch
BR	Bridge
ODL	Opendaylight
HTTP	HyperText Transfer Protocol
REST	Representational State Transfer
URL	Uniform Resource Locator
ESXi	Elastic Sky X
XML	Extensible Markup Language
DD	Database Description
LSA	Link-state advertisement

1 Bevezetés

Az utóbbi idők technológiai fejlődése magával hozza azt is, hogy elkerülhetetlen lett bizonyos területek további fejlesztése is. Ez a helyzet a hálózatok esetében is, ahol habár önmagában is folynak a különböző kutatások és újítások, mégis sok olyan alkalmazás lett az utóbbi években, amelyek megkövetelik a specifikus irányok fejlesztését is.

A Software Defined Networking -továbbiakban SDN- egy olyan kutatási terület, mely során a jelenlegi hálózati képet kell átalakítanunk mind fizikailag, mind logikailag. A hálózatok kezdeti kialakítása rengeteg újítás felé nyitotta meg a világot a mérnökök számára, és az akkori hálózati kép elég volt nagyon hosszú időre a különböző alkalmazásokhoz. Az utóbbi időkben sokkal nagyobb hangsúly került a közösségi médiára, cloud alapú rendszerekre, melyek azt eredményezik, hogy a használt hálózati erőforrások is lényegesen megnőnek. Ha a média területét nézzük, látható, hogy egyre jobb minőségű felvételek megosztására van igény, mely az adatközpontoktól induló kapcsolatok sávszélességének növelését kívánja meg. A különböző úgynevezett big data feldolgozására is egyre nagyobb az igény, mely szintén nagy terhelést tud a hálózati eszközökre gyakorolni. Végül, de nem utolsó sorban a robbanásszerűen nőtt azon eszközök száma, melyeket röviden csak mobil eszközöknek nevezhetünk. Ezek is csatlakozni kívánnak a már kialakított hálózatokra, mellyel újabb terhelő tényező alakult ki.

Az SDN megjelenése magával vonja, hogy a jelenleg használt módszert egyszer akár le is válthatja teljes mértékben, azonban ez időben egy nagyon hosszú folyamat lehet, így egyre több olyan tanulmány született arról az esetről, amikor az SDN alapú és az IP alapú hálózatok együtt vannak jelen bizonyos szegmensekben. A megvalósítás sok akadályt vethet fel, ugyanis nem egyértelmű, hogy hogyan lehet ezt a két különböző hálózati megoldást egy hálózatban észrevétlenül együtt működtetni, illetve hogyan lehet ezeket a rendszereket felügyelni, illetve tovább fejleszteni.

A diplomatervemben több ilyen megoldásnak járok utána, kialakítok egy ilyen jellegű rendszert és egy lehetséges implementációra adok ajánlást.

2 Software Defined Networking

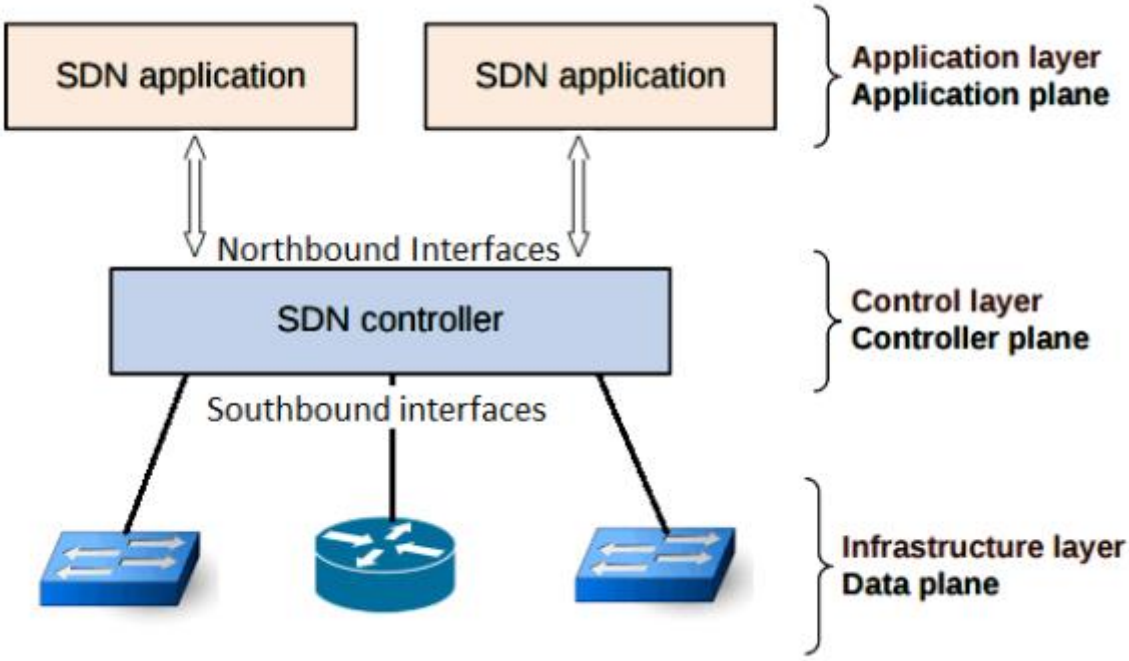
2.1 Software Defined Networking alapgondolata

A Software Defined Networking (SDN) [1] egy olyan paradigma, mely egyre inkább jellemző a hálózatok világára. Az SDN alapja az, hogy a vezérlési síkot és az adatkapcsolati réteget elkülönítsük egymástól, ezáltal levesszük a terhelést a csomóponti elemekről és át tudjuk adni a teljes útvonalválasztás, csomagkezelés feladatokat egy vezérlőnek. Ezt a szétválasztási gondolatmenetet ábrázolja az 1. ábra. Ennek köszönhetően szoftveresen tudjuk kezelni a hálózati feladatokat, ami sokban megkönnyíti a különböző feladatok ellátását. Habár manapság is szoftverek végzik az egyes csomópontokban a hálózati eszközök vezérlését, nagy szakértelem szükséges, hogy minden egyes eszköz megfelelően legyen beállítva és kialakuljon a tervezés során felvázolt hálózati megoldás. Az SDN ezen a téren is nagy segítséget tud nyújtani, hiszen sokkal rugalmasabban kezeli a hálózatkonfigurációt. Ezt a vezérlési és adatkapcsolati sík elválasztásával, a hálózat programozhatóságával, illetve azzal a ténnyel éri el, hogy van egy központi egység, aminek a hálózat egészéről van képe. Az elosztott hálózatokhoz hasonlóan egy SDN hálózatnak is vannak olyan alapkövetelményei, amelyeknek meg kell felelnie, attól függően, hogy milyen alkalmazásban is szeretnénk alkalmazni. Ezek az alábbiak:

- Második és harmadik rétegbeli kapcsolat biztosítása
- Hálózat virtualizáció
- Többfelhasználós környezet támogatása
- Skálázhatóság
- A logikai és fizikai kapcsolatok konfigurálásának szétválasztása
- Címek szétválasztása
- Elszigetelhető forgalom
- Az eszközök mozgathatóságának támogatása, akár virtuális gépek mozgatása
- Szolgáltatás minőségének biztosítása (Quality of Service=QoS)

- Auto-provisioning és szolgálatás felderítés
- Operations, Administration and Maintenance (OAM)

A hosztok, szerverek, hálózati eszközök általában vagy Internet Protokoll (IP) vagy Ethernet kapcsolaton keresztül érhetőek el, ezért a Layer 2-es (L2) és Layer 3-as (L3) kapcsolatok megléte alapvető követelménynek kell lennie.

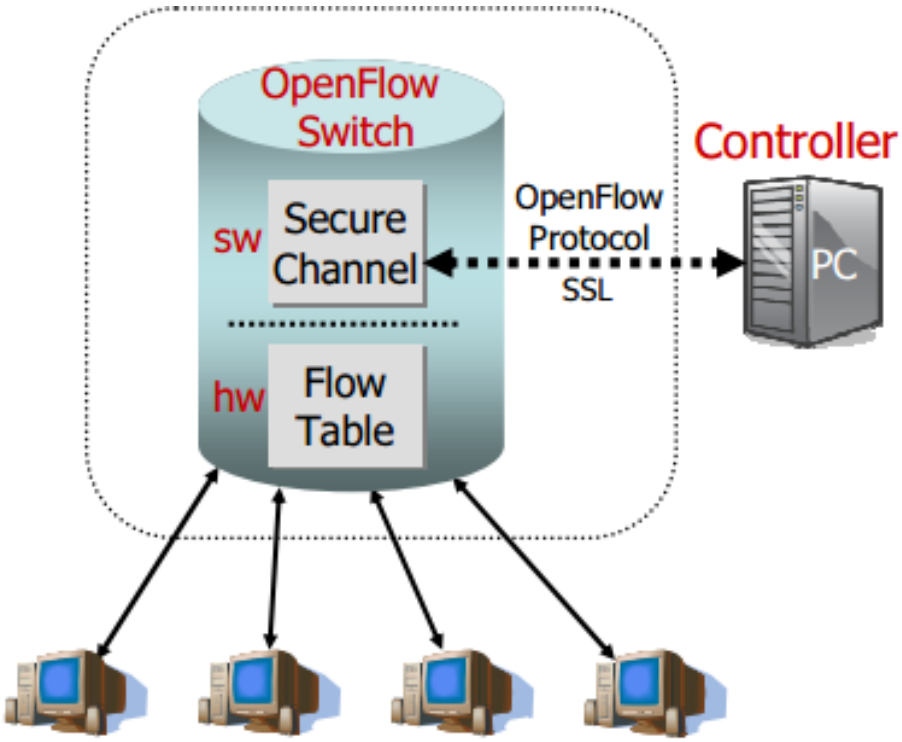


1. ábra Különböző rétegek szétválasztása

Az SDN megvalósításához látható, hogy több dologra is szükségünk van viszont nem szükséges, hogy mindent az alapoktól kezdve felépítsünk, hiszen rengeteg olyan nyílt forráskódú megoldást is találhatunk, amely a segítségünkre lehet egy tesztkörnyezet kialakításakor.

2.2 Openflow

Az OpenFlow egy olyan eszköz, amit gyakran használnak kutatások eredményeinek ellenőrzésére, illetve ennek segítségével akár új protokollokat is kipróbálhatunk olyan hálózatokban, amelyeket folyamatosan használunk. Az OpenFlow az Ethernet kapcsolókon alapul, egy belső forgalmi táblával. Az elgondolás lényege, hogy olyan platformot biztosítson, ami megfelel a kísérleti alkalmazásoknak, illetve a gyártók is tudjanak egy olyan felületet szolgáltatni, ami szabadon használható anélkül, hogy bővebb információt szolgáltatnának az eszközök működéséről.



2. ábra OpenFlow felépítés

Mivel sok olyan gyártó van aki hálózati eszközöket gyárt ezért elég nehéz egy olyan megoldást találni ami minden esetben azonosan lefedi a működést. Habár minden gyártó igyekszik saját megoldást használni, vannak bizonyos működési elvek, amelyeket nem lehet változtatni. Ezen funkciók meglétére, hasonlóságára épül az OpenFlow és ezt használja ki. Egy olyan protokollt biztosít amivel könnyedén tudjuk módosítani, irányítani a forgalmat. Egy hálózati üzemeltető könnyen ketté tudja választani a kísérletezésre és a hétköznapi használatra alkalmas hálózatot, ezáltal külön tud működni a két fajta folyamat, a másik megzavarása nélkül. Az ilyen eszközöket nevezzük OpenFlow switch-eknek. Ennek három részből kell állnia, amit a 2. ábra szemléltet is:

1. Flow Table: minden egyes folyamatra külön szabállyal, hogy tudja az eszköz, hogy mit kell tennie
2. Biztonságos csatorna: egy olyan csatorna, amellyel össze tudjuk kötni a switch-et egy távoli vezérlő egységgel

3. OpenFlow protokoll: ennek segítségével tud kommunikálni a vezérlő a switch

Az OpenFlow switch használatával elkerülhető az, hogy a felhasználónak a fizikai eszközt kelljen programoznia. Megkülönböztethetünk csak OpenFlow switch-eket, amelyeknél nincs támogatva a második és harmadik rétegbeli feldolgozás és olyan eszközöket, amelyek hagyományos Ethernet elveken működnek csak engedélyezve van rajtuk az OpenFlow protokoll használata és megfelelő interfészek is hozzá lettek adva. Az első esetben az eszköz feladata az, hogy a vezérlőtől kapott utasítások alapján útbaigazítsa a csomagokat és a megfelelő portokon továbbítsa azt. Ezen kívül végezhet még csomageldobást, vagy elküldheti a vezérlőegységnek megfelelő keretezéssel, hogy a későbbi csomagokra szabályt alakítsunk ki, vagy akár a kontrollerben végezzünk el bizonyos csomagfeldolgozási eljárásokat. A második esetben fontos újból kiemelni a korábban már leírt követelményt, vagyis valahogyan szét kell választani a hétköznapi, működő forgalmat és a tesztkörnyezetet.[2]

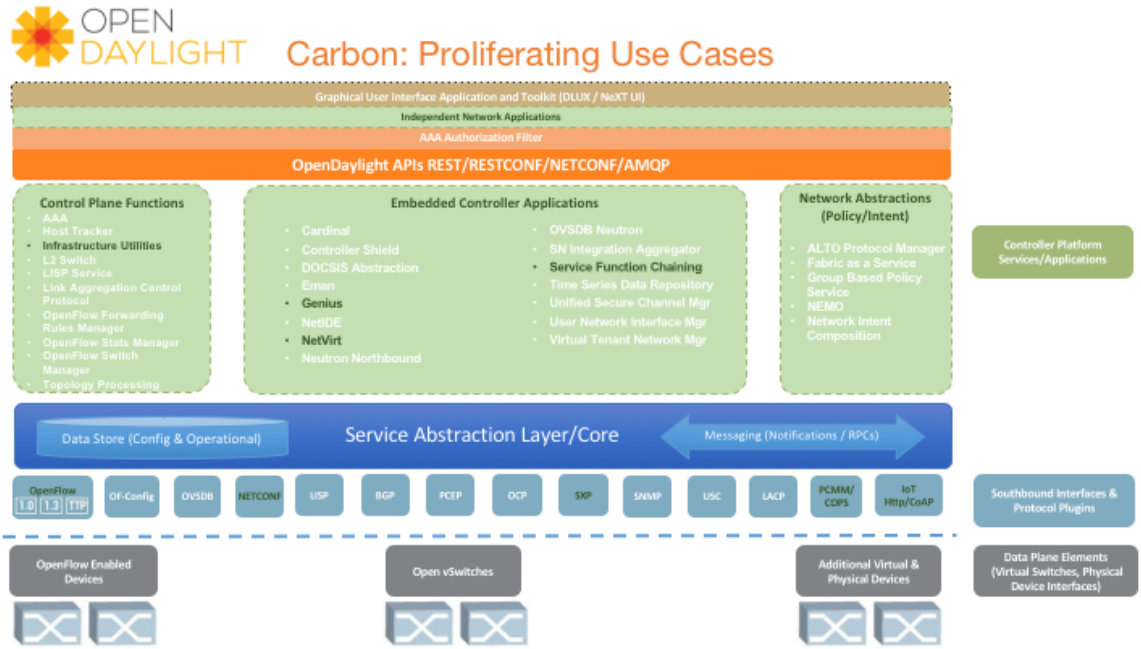
2.2.1 Kontrollerek

Az OpenFlow környezetben fontos feladata van a vezérlő eszközöknek, hiszen ezek azok a hálózati elemek, amik megoldják azt, hogy különválaszthassuk a különböző forgalmakat. Ethernet esetében egy elterjedt megoldás a VLAN-ok kialakítása és az OpenFlow esetében is ezzel a megoldással találkozhatunk a legtöbb esetben.

2.2.1.1 OpenDaylight

Az OpenDaylight [9] egy Java alapú moduláris megoldás, melyet nagyon sok másik kontroller vesz alapul. Ebben a keretrendszerben történő fejlesztések során a model-view-controller sémát kell használnunk, ezáltal egy nagyon rugalmas környezetet kapunk.

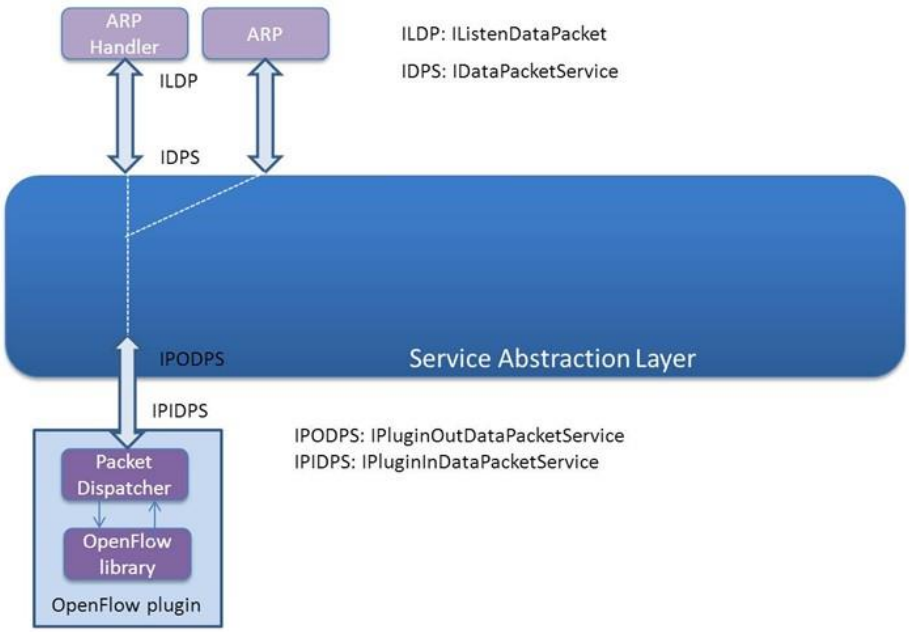
Ez a kontroller széles körben kedvelt, ezért mélyebb szinten szerettem volna vele megismerkedni a munkám során. Ahogy a 3. ábra is szemlélteti, a kontrollernek igen sok részegysége van, ami nagyban megsegíti a fejlesztések, kutatások előrehaladását. Az úgynevezett southbound interfészen rengeteg olyan protokollt láthatunk, amely már jól ismert akár SDN hálózatok, akár legacy rendszerek esetében.



3. ábra Az OpenDaylight Carbon felépítése

A southbound interfészen találhatunk olyan protokollokat, amelyek ahhoz szükségesek, hogy a kontroller az adatkapcsolati réteggel kommunikálni tudjon. Ezeket általában plugin-ként kezeli a kontroller és például itt található meg az Openflow is. Ezek a modulok dinamikusan csatlakoznak az úgynevezett Service Abstraction Layer-hez (továbbiakban SAL). Ez a réteg felel azért, hogy a felsőbb (északi) részen megírt szolgáltatások ki legyen kiszolgálva, elfedve azt, hogy milyen módon, protokollal kapcsolódik a kontroller az alsóbb szintekhez. Ez nagyban segíti azt a tényt, hogy a felsőbb rétegen megírt alkalmazások és az alsóbb szinten használt protokollok külön is tudjanak fejlődni.

Egy ilyen SAL szolgáltatásra példa a Data Packet Service, amelynek segítségével a kontroller az adatcsomagok vizsgálatát, továbbítását, kiküldését tudja megtenni. Ez a szolgáltatás hasznos lehet azokban az esetekben, ha egy olyan jellegű alkalmazást szeretnénk fejleszteni, amelyhez ilyen jellegű feladatokat szeretnénk rendelni.



4. ábra Service Abstraction Layer példával

A 4. ábra által bemutatott felépítéshez tartozó magyarázat[9]:

- IListenDataPacket: Egy olyan szolgáltatás, amelyet egy felső rétegbeli modul vagy alkalmazás készít azért, hogy csomagokat fogadhasson
- IDataPacketService: Egy olyan interfész, amelyet az SAL implementál azért, hogy csomagok küldésének és fogadásának lehetőségét biztosíthassa.
- IPluginOutDataPacketService: Ezt az interfészt akkor hozza létre az SAL, ha egy protokoll plugin csomagot szeretne eljuttatni az alkalmazásrétegbe
- IPluginInDataPacketService: Az interfész a protokoll plugin által van létrehozva azért, hogy az SAL-en keresztül csomagot küldhessünk a hálózati eszközökre.

A 4. ábra északi részén példaként az ARP handler szolgáltatást van feltüntetve. Ezen keresztül bemutatható, hogyan is működik a rendszer[9].

1. Az OpenFlow plugin kap egy ARP csomagot, amelyet el kell juttatni az ARP Handler alkalmazáshoz
2. Az OpenFlow plugin meghívja az IPluginOutDataPacketService-t hogy eljuttassa a csomagot a SAL-hez

3. Ha az ARP Handler alkalmazás beregisztrált az IListenDataPacketService szolgáltatásra, akkor amint megkapja az SAL a csomagot, továbbítja az alkalmazásnak
4. Az alkalmazás feldolgozza a csomagot

A visszairányba hasonló módon történik a kapcsolat felépítése:

1. Az alkalmazás készít egy csomagot, majd meghívja a SAL által biztosított IDataPacketService-t, hogy kiküldje a csomagot. A csomag célját az API-ban adjuk meg.
2. Az SAL meghívja az IPLuginInDataPacketService interfészt egy adott protokoll pluginnal attól függően, hogy mit használ a cél eszköz
3. A plugin elvégzi az összes feldolgozással járó feladatot, majd kézbesíti a csomagot a megfelelő eszköznek.

2.2.1.2 Ryu

Mivel ez a vezérlő egy Python alapú megoldás, ezért igen egyszerűen a *pip* szolgáltatás segítségével lehet telepíteni. A kontroller alapelve, hogy nem akar egy nagy monolitikus eszköz lenni, egyszerűen csak egy keretrendszert szeretne biztosítani az SDN alapú alkalmazások fejlesztéséhez. Ha mi szeretnénk megírni nagyon sok alkalmazást és így kialakítani a kontrollerünket, akkor jó választás lehet, hiszen nagyon erősen támogatja az alkalmazások létrehozását. [6]

2.2.1.3 Open Network Operating System

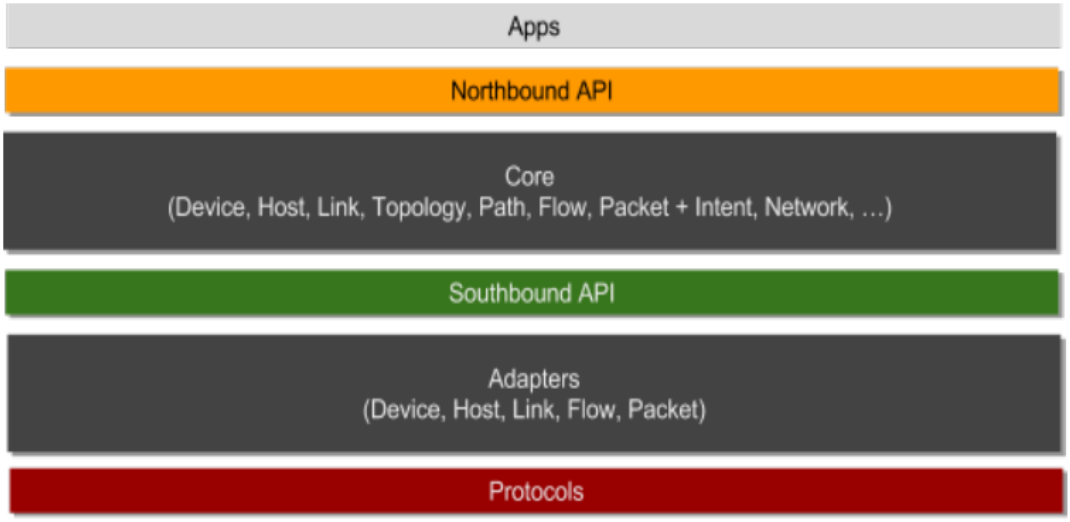
Az ONOS az Open Network Operating System rövidítéséből adódik. Ez a kontroller is egy Java alapú megoldás, amely SDN alkalmazások fejlesztéséhez nyújt megfelelő platformot. [7]

A kontroller architektúrájával kapcsolatos tulajdonságok nagyon hasonlítanak az Opendaylight esetében tapasztalt tulajdonságokhoz:

- Elosztott központi felépítés, ami növeli az elérhetőséget, valamint skálázhatóságot, és jobb teljesítményt biztosít.
- Northbound API-k amik megkönnyítik a kontroll, menedzsment és konfigurációs szolgáltatások fejlesztését.

- Southbound API-k amik segítenek abban, hogy vezérelni tudjuk mind az Openflow, mind a legacy eszközöket.
- A szoftvermodularitás biztosításával könnyen lehet fejleszteni, karbantartani az ONOS-t akár a közösség fejlesztői által is.

Az 5. ábra bemutatja az ONOS rétegeit, amely nagyon hasonlít az Opendaylight felépítéséhez.



5. ábra Az ONOS réteges felépítése

3 Hibrid hálózatok

A hibrid hálózatokat alapvetően két oldalról érdemes megközelíteni. Az első felfogásban, a hálózatunkban olyan hagyományos és nem hagyományos csomópontokat használunk, amelyek képesek az Openflow protokoll értelmezésére és használatára. Ebben az esetben magukat az eszközöket is hibridnek tekintjük.

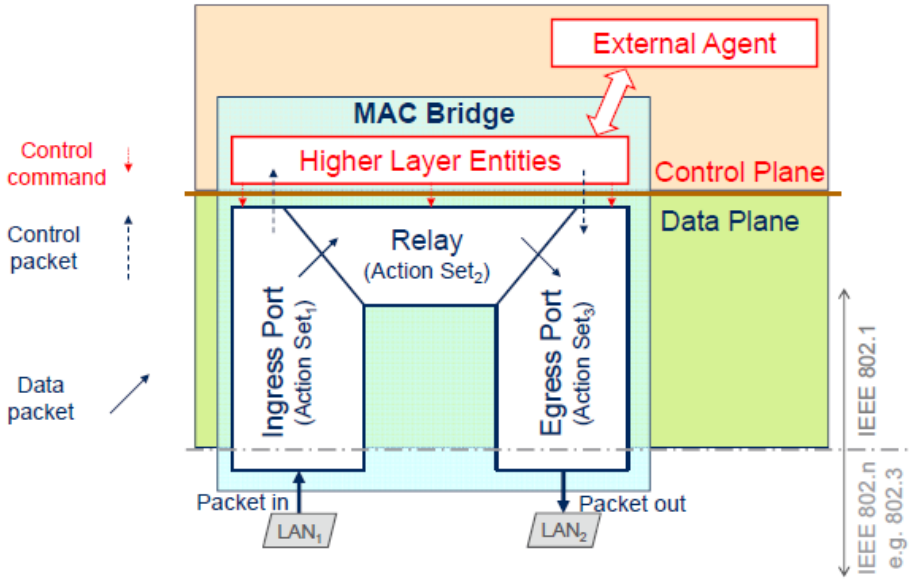
A másik felfogásban hibridnek tekintjük a hálózatot, hogyha van egy SDN alapokon működő rendszerünk és el szeretnénk érni, hogy valahogy ki tudjunk küldeni olyan üzeneteket is a kontrollerből, amiket értelmezni tud egy olyan eszköz is, ami nem ismeri az Openflow protokollt. Habár a legtöbb gyártó folyamatosan arra törekszik, hogy a régi eszközeiket kompatibilissé tegyék az SDN-es világgal, megeshet, hogy olyan eszközök működnek a hálózatban, amelyeket már nem támogat ilyen szinten a gyártó, ezért szükség lehet valamilyen olyan megoldásra, amivel el tudjuk halasztani a rendszer fejlesztésével járó költségeket.

Jelen esetben még kérdéses, hogy költséghatékonyság szempontjából melyik éri meg jobban a felhasználók, vállalatok számára, ezért mind a két lehetőséget megvizsgáltam és igyekeztem jobb belelátást nyerni a megoldásokba.

3.1 Bridge architektúra

A 802.1 szabvány egy olyan modellen alapul ahol a vezérlő és az adatréteg jól elkülöníthető. A 6. ábra bemutatja ezt a MAC Bridge architektúrát, illetve felépítését. A vezérlő rétegben találhatunk meg olyan entitásokat, amelyek az elosztott vezérlési protokollok segítségével irányítani tudják a hálózati forgalmat. Ezen felül a szabvány megengedi külső vezérlő entitások bevonását is, ami lehet akár egy SDN kontroller is, és ezek egymás mellett is megfelelően tudnak működni. Az adatrétegben van két port és egy ezeket összekötő modul. Az ellátott feladat alapján a két portot megkülönböztetjük, az egyik a bejövő, míg a másik a kimenő forgalom számára van fenntartva. Az érkező csomag a bejövő porton halad át, ahol, ha úgy van megoldva a rendszer, akkor végezhetünk valamilyen módosítást a csomagon. Ezután átkerül a középső modulba, majd a kimenő portra, ahol mind a két helyen lehet módosításokat végezni még a csomagon. Ezek a változtatások vagy a külső vezérlőtől vagy az elosztott vezérlőtől függenek. A portokon és az összekötő modulon lehet megvalósítani az úgynevezett

Virtual LAN (VLAN) alapú forgalomszétválasztáshoz tartozó feladatok ellátását, mint például a címkék megvizsgálása, átcímkézés. [5]



6. ábra Bridge architektúra

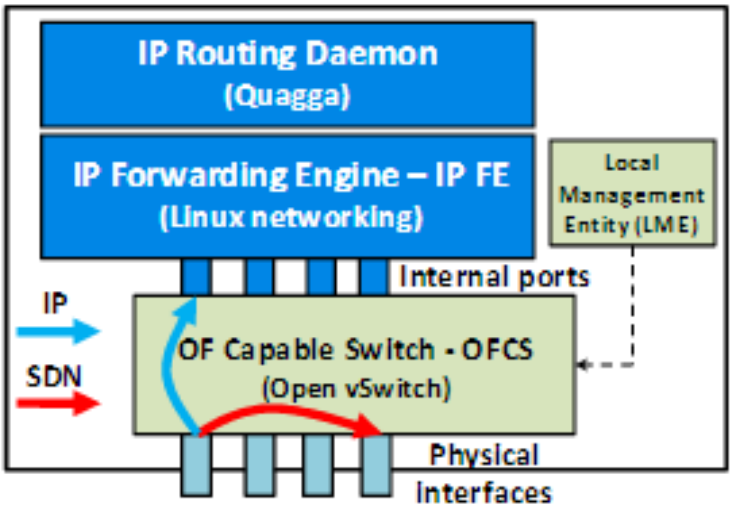
Az Ethernet hálózatoknak több topológiai rétege van, amelyekből az összeset lehet SDN vagy egy elosztott vezérlő protokoll által programozni.

A VLAN legnagyobb előnye, hogy ki tudunk alakítani virtuális hálózatokat és a csomagok magukkal viszik azt az azonosítót ami alapján meg tudjuk határozni, hogy melyik forgalomhoz tartozik. Ennek köszönhetően az eszközök tábláiban képet kaphatunk a hálózatunkban található virtuális topológiákról.

3.2 MPLS alapú megoldás

3.2.1 Hibrid csomópont

A hibrid csomópont egy olyan eszköz, mely magába foglal egy OpenFlow-ra képes switch-et (OFCS) egy IP forwarding engine-t és egy IP routing daemon-t. Ebben az esetben az OFCS egy Open vSwitch, a forwarding engine maga a Linux kernel IP hálózati része, míg a Quagga játssza a routing daemon szerepét.

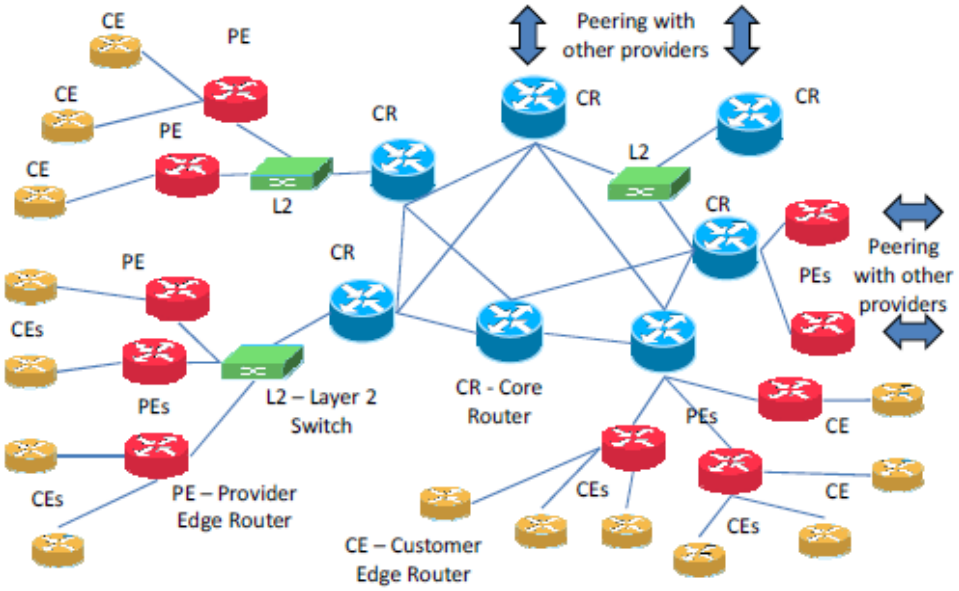


7. ábra Hibrid node

Látható, hogy szükség van arra, hogy virtuális portokat használjuk az OFCS és IP továbbító között, ehhez az Open vSwitch által biztosított Internal Port lett alkalmazva. Az 7. ábra mutatja be, hogy a fizikai interfészen beérkező csomagot IP forgalom esetén továbbítja az OFCS a belső portokra. Ezzel gyakorlatilag valamekkora teljesítménycsökkenést kapunk, hiszen a csomag kétszer halad át az eszközön, viszont cserébe nincs szükségünk arra, hogy az IP forwarding táblát SDN szabályokra fordítsuk.

3.2.2 Open Source Hybrid IP/SDN

A nagyméretű IP hálózatokban megjelenő SDN-ről több tanulmány is készült már, ezekből a [3] -t emelném ki és mutatnám be egy kicsit részletesebben, mert megfontolandó a szerzők által ajánlott módszer. A megoldást Open Source Hybrid IP/SDN-nek vagy röviden OSHI-nak nevezik. Mint ahogy a nevében is benne van open source programokkal oldották meg az IP és SDN hálózatokban lévő forgalomirányítást. Ez a megoldás azért jelentős, mert a biztosított keretrendszer és egyéb eszközök mellett lehetőség van új szolgáltatások bevezetésére és különböző teszteket futtatni az SDN vezérlővel. A tanulmányban bemutatott hálózat egy IP Service Provider (ISP) eset volt, amelyre egy példa topológiát mutat be a 8. ábra. Maga szolgáltató hálózat csatlakozhat másik ISP-khez, mondjuk BGP protokoll segítségével, míg egy ISP hálózaton belül valamilyen routing protokollt használunk, ami lehet akár OSPF is. A példa hálózatban jól láthatóak olyan elemek, melyek jellemzően IP/MPLS hálózatokban találhatóak meg, így van Core Router (CR), Provider Edge (PE), Customer Edge (CE).



8. ábra Egy lehetséges hálózati kép az OSHI megvalósításkor

Egy ilyen ISP esetében nagyon gyakran alkalmaznak MPLS technológiát, amikor egy-egy alagút képződik a csomópontok között és ezen keresztül zajlik a kommunikáció. Ez egy előnyös megoldás, hiszen egyrészt segít feljavítani a forgalomkezelés minőségét, másrészt tudunk VPN, vagy második rétegbeli kapcsolódást biztosítani a csomópontoknak. Az IP és MPLS alapú megoldások jól tudnak egymás mellett létezni, ezért az IP/SDN kapcsolatot is ennek megfelelően fogjuk tekinteni.

A hibrid csomópontokon olyan eszközöket értelmezünk, amelyek képesek IP alapú, illetve SDN alapú forgalomirányításra is, melyhez egy vezérlő segítségét használja. Alapvetően a hálózatban ezeken a csomópontokon kívül lehetnek hagyományos L2-es switchek, illetve IP routerek is, ezért is lesz a hálózatunk hibrid. Az MPLS esetében említettük, hogy egy adott alagútat tudunk kiépíteni két csomópont között. Az SDN esetében is tudunk ilyen útvonalakat kiépíteni, viszont több típust is meg lehet különböztetni, mint például MAC, IP címek, VLAN, MPLS címkék. Ezekre az útvonalakra gondolhatunk úgy is mint egy-egy virtuális áramkörre, amelyen a csomagokat tudjuk folyamatosan küldeni. A forrásban ismertetett megoldás tervezési fázisában felmerült a VLAN alapú címkézés módszere, azonban ezt korlátozza a címkék mennyisége, vagyis jobban kell figyelni az SDN alapú útvonalak számára, ezért inkább az MPLS alapú címkézést javasolják a tesztelésekhez, kutatásokhoz. A nagyméretű

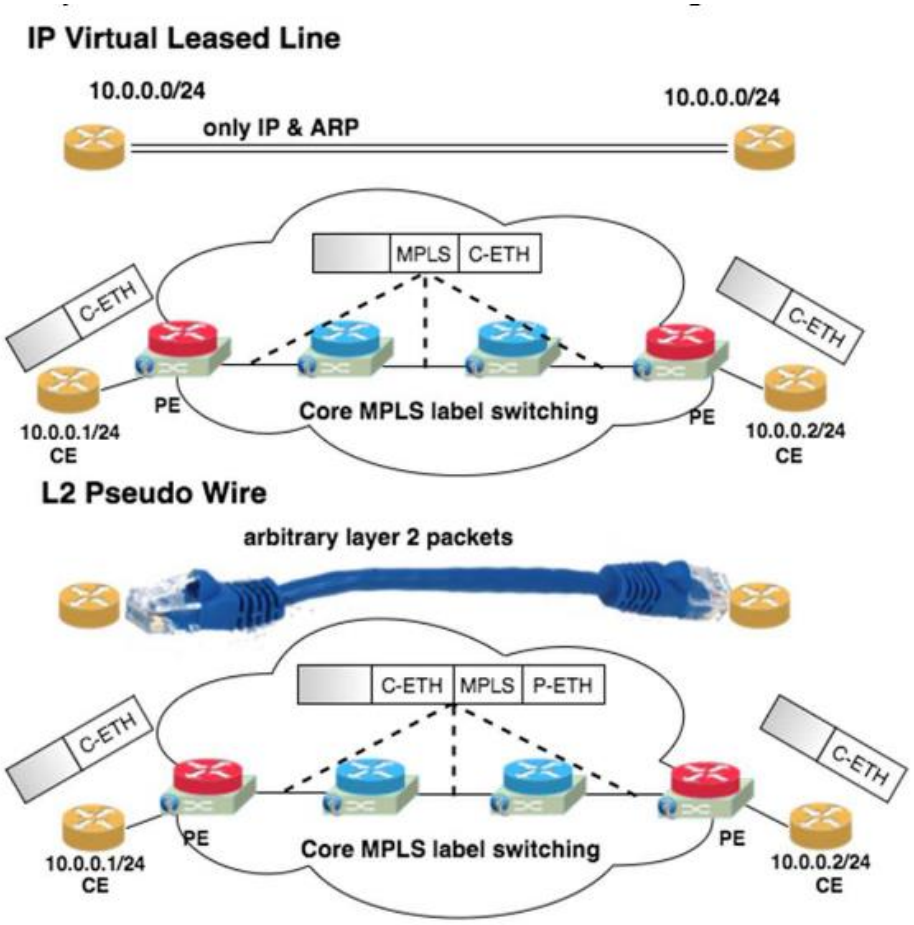
Wide Area Network (WAN) esetben az SDN kontroller és a megfelelő switch-ek közötti kommunikációval gond van, amit általában out-of-band módszerrel oldanak meg. Az OSHI általi legnagyobb előny a hagyományos IP routing és forwarding alkalmazása az SDN vezérlő és switch-ek között. A hibrid hálózatok által három főbb szolgáltatást tudunk kiemelni:

- virtuális magánhálózatok (L2 és L3)
- hálózati forgalom menedzsmentje
- gyors javítási mechanizmusok

A hálózaton áthaladó forgalom vizsgálata során el kell tudni dönteni, hogy adott esetben hagyományos IP alapú forgalomirányításra van szükségünk, vagy egy SDN-es útvonalhoz tartozik a csomag.

3.2.3 Főbb szolgáltatások

Az alapvető szolgáltatások közé tartozik az úgy nevezett IP Virtual Leased Line (IP VLL) és a Layer 2-es „Pseudo-wire” (L2 PW vagy PW). Ezek a szolgáltatások egy-egy dedikált csatornát biztosítanak vagy harmadik, vagy második rétegben, így olyan sávszélesség garanciát igénylő forgalmat tudunk rajta szállítani, mint például a valós időben történő kommunikáció, vagy akár olyan szolgáltatások, mint például a VPN. Általában a végpontok lehetnek virtuális vagy fizikai portok és a kapcsolatok közöttük MPLS címkékkel, vagyis MPLS alagúttal vannak kialakítva.



9. ábra PW és VLL

Az IP VLL szolgáltatás a működés során megőrzi az eredeti forrás és cél MAC címet, ami problémát tud okozni, ha hagyományos L2-es switch-eken is áthalad a forgalom, ezért ez a szolgáltatás csak akkor tud megfelelően működik ha minden edge és core csomópont OSHI kompatibilis és direkt kapcsolat van közöttük, tehát nincs a köztük lévő útvonalon L2-es switch. Habár ez az akadály megoldható címek átírásával, ez egy elég összetett és bonyolult feladat, ezért ha a hálózatban ilyen kapcsolók is léteznek, a PW szolgáltatást kell használnunk. Ahogy a 9. ábra is szemlélteti, ilyenkor a kezdeti Ethernet csomagot belerakjuk egy másik Ethernet csomagba, majd ezt az egészet egy MPLS fejléccel is ellátjuk.

3.2.4 Mantoo (Management Tools for SDN)

Ahogy a nevében is benne van a Mantoo egy olyan eszköztár, amellyel a hibrid hálózatunk felügyeletét, menedzselését tudjuk végrehajtani. Ebben nyílt forráskódú eszközök vannak összefogva annak érdekében, hogy könnyebben tudjuk az SDN-es kísérleteinket véghezvinni mind Mininetes környezetben mind elosztott

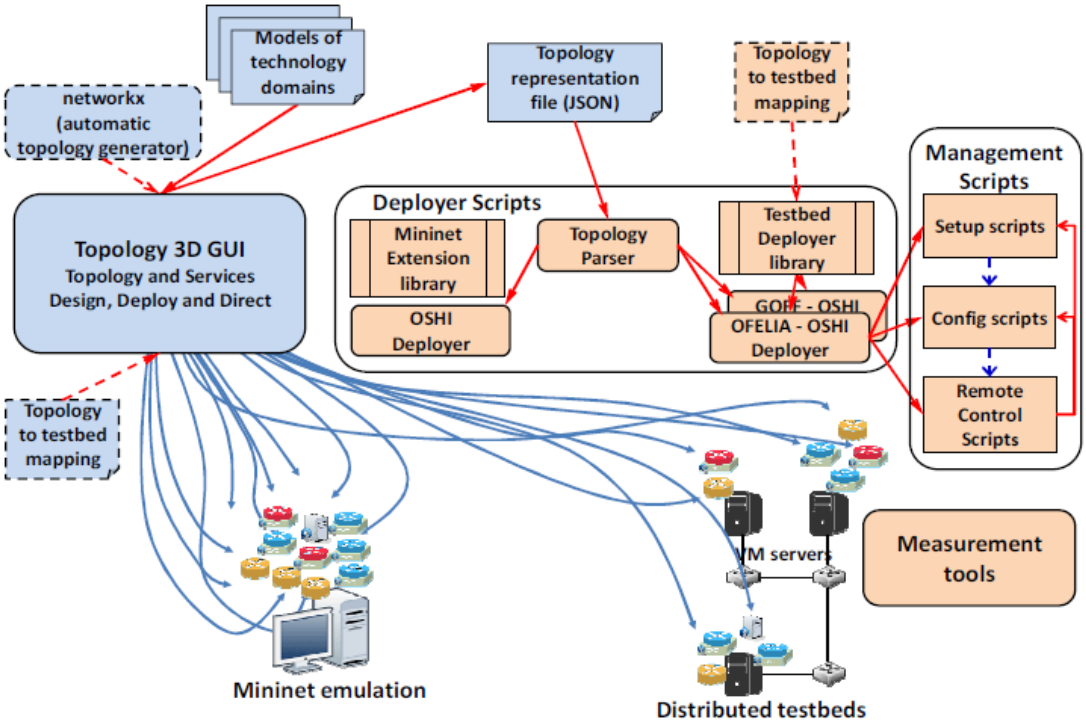
tesztkörnyezetben. Ennek az eszköznek a segítségével a tesztek különböző fázisain tudunk egyszerűen végig haladni, így a tervezési, megvalósítási, vezérlési és mérési fázisokon.

A tervezési fázishoz a Mantoo egy Topology3D nevezetű eszközt biztosít, amely egy webes felület. Ez egy JavaScript kliens és egy Python alapú back-end, ahol egy kényelmes felületen tudunk számunkra megfelelő topológiát kialakítani és itt akár még az eszközök konfigurálást is megtehetjük, parancsokat tudunk kiadni a megfelelő nézet kiválasztásával. A hálózatunk kialakításához egy szöveges környezetben tudjuk a megfelelő megszorításokat megadni, ezáltal sokkal pontosabban megkaphatjuk az általunk elképzelt hálózati képet. A felület ezután JSON formátumban kerül exportálásra, amelyet utána a megvalósítási fázisban használ fel a Mantoo.

A megvalósítási fázisban történik meg a megtervezett hálózat átfordítása olyan parancsokra, amelyek utána a csomópontok felkonfigurálásához szükségesek. Ebbe beletartoznak az eszközökhöz tartozó parancsok, illetve a különböző implementálni kívánt protokollok, szabályok leírása. Ez a fázis különböző platformokra vonatkozhat, az idézett irodalomban [3] a Mininet nevezetű emulátort célozták meg. A megvalósítás valójában számos Python script alapján van megoldva, amelyek feldolgozzák a tervezési fázisból nyert JSON forrásokat és több másik scriptet generálnak, amelyek leginkább shell scriptek.

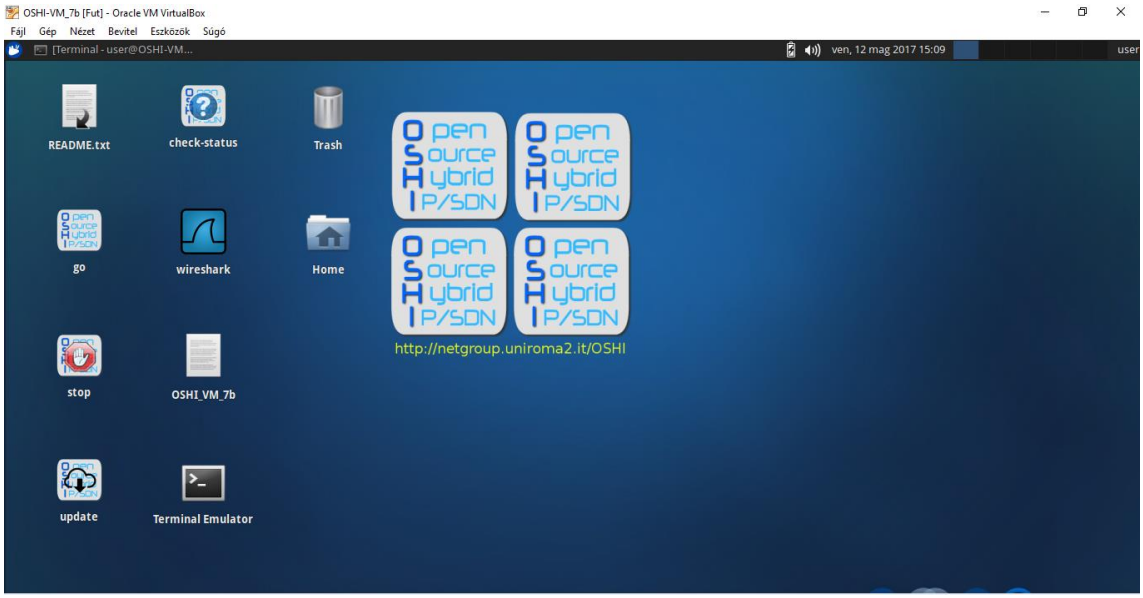
A mérési, vezérlési fázisban a Mininet sajátosságaiból adódóan terminál ablakokat nyithatunk a csomóponti elemeken és itt tudjuk követni a különböző folyamatok kimeneteit. Az automatizálás megoldásához egy Python könyvtár áll rendelkezésünkre, ami egy API biztosításával megadja számunkra azt a lehetőséget, hogy a saját teszteseteinket leprogramozhassuk. SSH segítségével tudunk forgalmat generálni, illetve akár az eszközök CPU terheltségét is meg tudjuk figyelni. Erre azért van szükség, mert a virtuális gépek általában nem hiteles információkkal látnak el minket, így a virtuális környezet segítségével a valósághoz sokkal jobban közelítő adatokat kaphatunk, amelyek bizonyos mérések esetén döntő tényezők is lehetnek.

A Mantoo részletesebb működéséről a 10. ábra tanulmányozásával tudhatunk meg többet, én csak a fontosabb részeket emeltem ki.



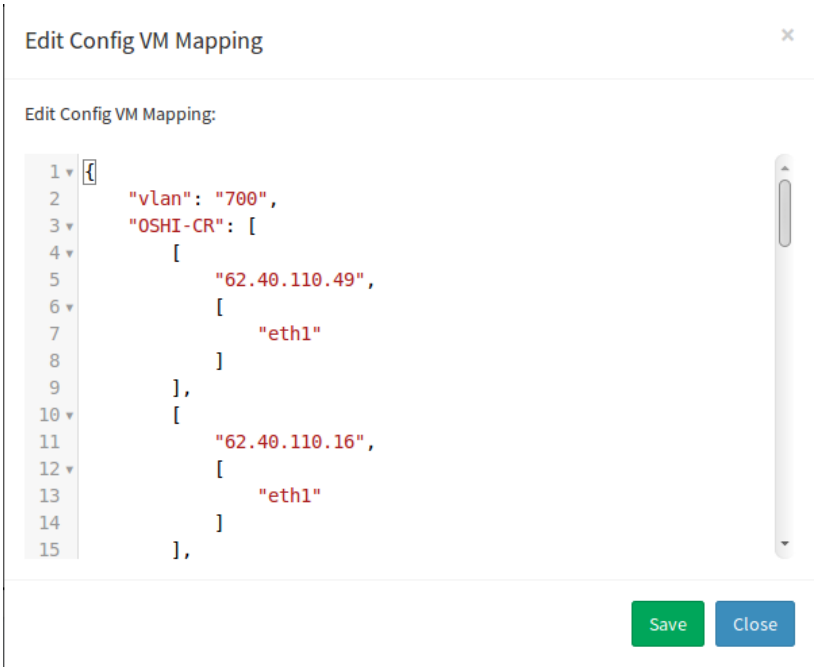
10. ábra A Mantoo működése

Az OSHI rendszerre rákeresve, megtalálható a hivatalos weboldaluk, ahonnan komplett virtuális gép is letölthető. A Topology3D-hez kapcsolódóan megnéztem pár működési mechanizmust, amelyekről a képernyőképek alább láthatóak.



11. ábra Virtuális gép a Mantoo eszköztár kipróbálására

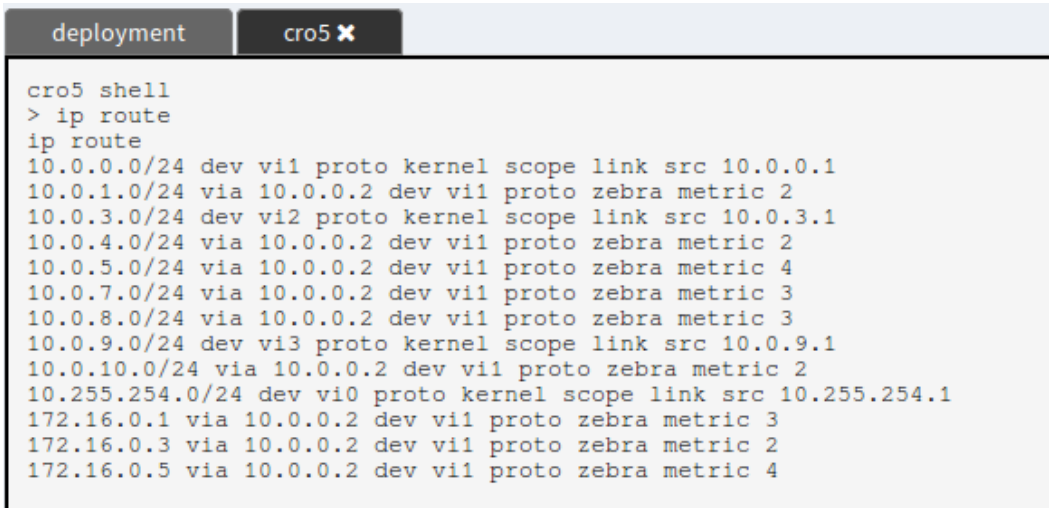
A virtuális gép működőképes és egyszerűen egy importálás után használható is. Az előbb látott képen látható, hogy különböző indító, leállító scriptek mellett van Wireshark is, illetve egy rövidebb használati útmutató, gyakorlási lehetőség.



12. ábra A virtuális gépek elhelyezkedésének módosítása

Ha a hálózati topológián egyes elemekre a control billentyű lenyomása mellett kattintunk, akkor kapjuk meg a terminált, ahol olyan parancsokat tudunk futtatni mint például:

- ip route
- ip addr
- ifconfig
- ping 10.0.0.1
- traceroute 10.0.4.1



13. ábra Routing tábla ellenőrzése

Ez a jellegű felépítés mindenképpen érdekes, hiszen úgy néz ki, hogy egyre több ilyen eszköz van és szükség van olyan jellegű szimulációs rendszerekre, amelyek segítenek abban, hogy tesztelni tudjunk minél több megoldási lehetőséget.

3.3 Hybridtrace

A hálózatok esetében egy elvárt követelmény, hogy legyen megfelelő eszközünk arra, hogy kideríthessük a hibákat. A hibrid környezetben nem egyértelmű, hogy milyen módszerekkel tudjuk megoldani ezt a vizsgálatot, hiszen külön módszert ismerünk a régi (legacy) hálózati megoldásokra és az SDN hálózatok vizsgálatára. Legacy hálózatokban a legjobban alkalmazható hibakeresési eszköz a traceroute, mely különböző platformokon különböző parancs segítségével érhető el. A traceroute során, a hálózaton áthaladó csomag útvonalát láthatjuk, de csakis harmadik rétegbeli IP útvonalat kaphatunk ilyenkor, hiszen az IP csomagokban lévő Time to Live (TTL) mezőn alapul ez a módszer. Az útvonalon lévő csomópontok ez alapján küldenek vissza ICMP hibaüzeneteket és így lehet végig követni egy csomag útját a hálózatban. [4]

Az SDN hálózatokban nem használható a traceroute, mert a switch-ek nem minden esetben működnek úgy, mint egy L3-as router, sokszor egyszerűen L2-es eszközökként kell tekintenünk rájuk. Éppen ezért értelemszerűen az IP alapú útvonalkövetés nem fog működni. Megoldásként az SDN traceroute-ot használják, melynek lényege, hogy az SDN switch-ekben bizonyos magas prioritású szabályok vannak elhelyezve, melyek segítségével az SDN vezérlő megtudja, hogy az

útvonalfelderítő csomag melyik csomópontokon haladt át. Érezhető, hogy ezt az eszközt nem használhatjuk a hagyományos hálózatunkban, hiszen ott nincs vezérlő egység.

A hibrid hálózatokban jól el tudunk különíteni külön SDN és külön hagyományos hálózati részeket, amelyek közötti átjárások miatt bonyolultabb eszközökhöz kell fordulnunk. Erre jelent megoldást a Hybridtrace nevezetű megvalósítás, aminek segítségével fel tudunk térképezni két csomópont közötti útvonalat ebben az új hálózati kialakításban. A módszer gyakorlatilag ötvözi a két különböző megoldást, azonban egy kicsit változtat is rajta, főleg az SDN-es részen. A csomag a hagyományos hálózati részekben a TTL mező növelésével hajtja végre az útvonal felderítését. Az SDN vezérlő egy olyan szabályt hoz létre a switch-ekben, ami illeszkedik arra az esetre, amikor a TTL értéke 1. Ekkor a csomópont elküldi a csomagot a vezérlőnek és megkezdődhet az SDN részhálózaton belüli útvonalkövetés. Ezen felül a Hybridtrace megvalósítása olyan, hogy az SDN hálózaton belül bizonyos OpenFlow-ra jellemző üzenetek segítségével képesek vagyunk megmondani a csomópontok közötti kapcsolatok késleltetését is, melynek segítségével újabb értékes adatot kaphatunk a hálózatról.

4 SDN hálózat megvalósítása

A hálózati megoldás során több lépcsőfokban terveztem a munkámat, kiindulva a teljesen virtuális megközelítésből, egészen odáig, hogy az egyes eszközök tényleges fizikai összeköttetésben álljanak egymással, mintha azt a való életben kábelek segítségével tennénk meg. Ezt az utolsó lépcsőfokot eszközök hiányában virtuális gépekkel és hypervisor segítségével szimuláltam, így még nem a tényleges hálózatot alakítottam ki, de egy jó közelítést adtam.

4.1 Virtualbox

A hálózati eszközök szimulálásához virtuálisan valósítottam meg az eszközöket, amelyre Virtualboxot használtam. A Virtualboxon belül különböző módon lehet a virtuális gépeinkhez hálózati csatlakozót rendelni.[10]

4.1.1 NAT mód

Ez a legegyszerűbb módja annak, hogy külső hálózatot elérjünk a virtuális gépünkről. Mivel semmilyen egyéb konfiguráció nem szükséges a hálózaton, ezért ez az alapbeállítás a Virtualboxban, tehát ha létrehozunk egy új virtuális gépet, akkor hozzáad egy ilyen adaptert. A virtuális gépről ezáltal elérjük az internetet is, viszont más gépekkel nem tud kommunikálni a hálózatunkon.

4.1.2 Bridged mód

Ebben a módban a Virtualbox közvetlenül hozzáfér a Virtualboxot futtató számítógép, vagyis a hoszt hálózati vezérlőjéhez. Ha ilyen beállítást választunk, akkor a hoszt gép számára úgy látszik a virtuális gép, mintha fizikailag, hálózati kábellel lenne csatlakoztatva. Ezáltal a hálózatunkon lévő fizikai és virtuális gépek látják egymást, tehát akár egy fizikai hálózattal is kommunikálhat a virtuális hálózatunk. A legtöbb esetben ilyen beállítási módot használtam a hálózatban.

4.1.3 Host-only mód

A host-only hálózati beállítás esetén a hoszt gépen egy szoftveres interfész jön létre, melyet tekinthetünk egy új loopback interfésznek is. Ezt az interfészt használjuk arra, hogy a virtuális gépet csatlakoztassuk a hoszt géphez. Ennek köszönhetően a

virtuális gépek látni fogják a hoszt gépet, viszont azon túl a többi fizikai eszközt nem láthatjuk.

4.1.4 Internal mód

Ez a beállítási mód a bridged módhoz hasonlít, viszont ebben az esetben a virtuális gépek csak egymással tudnak kommunikálni.

4.2 Kontroller Mininet hálózattal

4.2.1 OpenDaylight telepítése

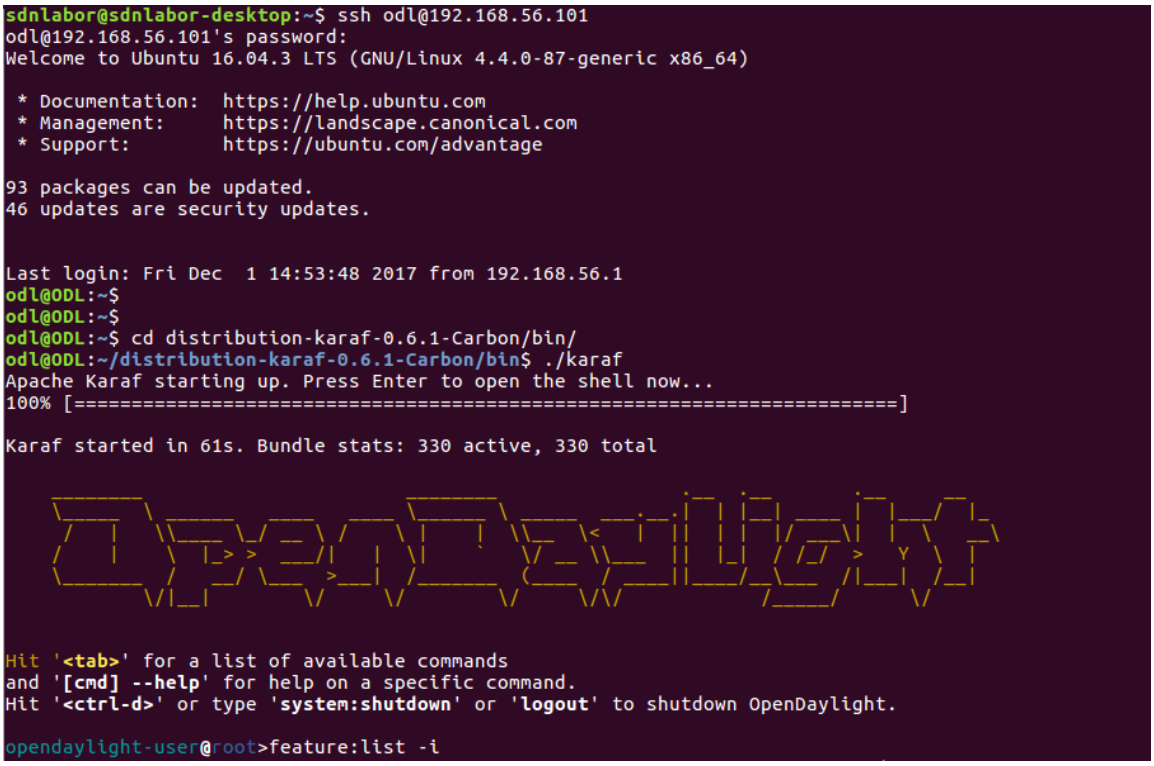
Az OpenDaylight kontrollert egy VirtualBox-ban létrehozott virtuális gépre telepítettem. Erre a virtuális gépre egy Ubuntu Server 16.04 operációs rendszert tettem fel, ugyanis nincs szükség grafikus felületre ahhoz, hogy az OpenDaylight-tal dolgozni tudjunk. A virtuális gép beállításánál figyelnem kellett arra, hogy elegendő erőforrást adjak a virtuális gépnek. Ezen felül fontos volt, hogy egy olyan hálózati interfészt is beállítsak, amelyet host-only networking módba kapcsoltam. Ezáltal a többi virtuális géppel is tud kommunikálni, illetve a hoszt gépről is elértem a kontrollert. Ez ahhoz volt szükséges, hogy a hoszt gép böngészőjében megnyitva a kontroller IP címét, kiegészítve a megfelelő porttal, láthassam az adminisztrációs felületet, amin a csomópontok, hálózati topológia, flow-k jelentek meg. A menedzselés megvalósításhoz SSH-t állítottam be, ezáltal a Virtualboxot futtató gépről is könnyebben elérhettem a kontroller gépet.

Miután feltelepítettem az operációs rendszert, beállítottam a kontrollert is. Ehhez először szükségem volt a Java telepítésére is.

```
$ sudo apt-get install openjdk-8-jdk
```

Ezután letöltöttem az OpenDaylight weboldaláról a Carbon nevezetű disztribúciót, melyet kicsomagolva már készen is vagyunk a kontroller telepítésével. A kontrollert úgy tudjuk elindítani, hogy a kicsomagolt könyvtárban a *bin* mappába navigálunk, majd ott kiadjuk az alábbi parancsot:

```
./karaf
```

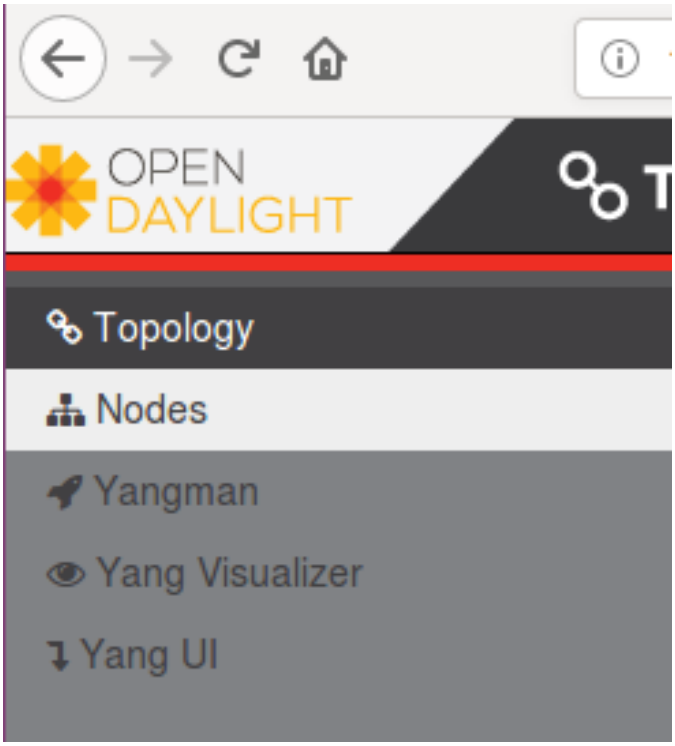


14. ábra OpenDaylight indulása

Ekkor még csak az alap csomagot telepítettem. Ahhoz, hogy a kontroller fel tudja deríteni a topológiát, illetve én is több információt tudhassak meg róla, szükség van egyéb modulok telepítésére is. Ezeket az alábbi parancsokkal tettem meg:

- `feature:install odl-restconf` : Ez a feature ahhoz szükséges, hogy elérhessük a RESTCONF API-t.
- `feature:install odl-l2switch-switch` : Ennek a lényege, hogy egy valós switch-hez hasonló működést tudjuk reprodukálni. Ez azért szükséges még, hogy a felhasználói felületen részletes információkat láthassunk az egyes csomóponti elemekről.
- `feature:install odl-dlux-core` : A felhasználói felülethez tartozó alapcsomagokat telepíthetjük így. Ha csak ezt telepítjük, akkor a felhasználó felületen csak a topológia alkalmazás érhető el.
- `feature:install odl-dlux-nodes` : A felhasználói felülethez hozzáadjuk a csomópontokhoz tartozó csomagot.
- `feature:install odl-dlux-topology` : Ahhoz szükséges, hogy a felhasználói felületen láthassuk az általunk kialakított hálózat képét.

- `feature:install odl-dlux-yangui odl-dlux-yangman odl-dlux-yangvisualizer` : A YANG modulhoz való hozzáférést segíti. A telepítés után a felhasználói felületen bal oldalon megjelennek a YANG-hoz tartozó menüpontok is.



15. ábra OpenDaylight felület, menüpontok

Az általunk, illetve alapból feltelepített feature-ök listáját a `feature:list -i` paranccsal ellenőrizhetjük.

A hálózatban ez a kontroller jelen, illetve a többi teszteset során is a `192.168.56.101` IP címen érhető el. A virtuális gépet futtató gépről is erre a címre tudtam SSH kapcsolatot kiépíteni, ezáltal a menedzselés könnyebben ment. A kontrollerhez tartozó felhasználói felületet a hoszt gépen megnyitott böngészőből értem el, méghozzá a `http://192.168.56.101:8181/index.html` címen. Itt először egy login képernyő fogadott, ahol a megfelelő autentikáció adatok megadására volt szükség. Az Opendaylight esetén az alapbeállítás az `admin` felhasználónév és `admin` jelszó. Ezek megadása után a kezdőfelület a Topology menüpont volt, vagyis első

belépés után egyből láthatjuk az általunk készített hálózatot, ha a kontroller felismerte. Abban az esetben, ha le szeretnénk állítani a kontrollert, elegendő egy `ctrl+D` billentyűkombinációval, `system:shutdown` vagy `logout` parancs kiadásával jeleznünk.

4.2.2 Mininet telepítése

A Mininet-es hálózat létrehozására több lehetőség is van. Az első esetben egyszerűen a telepített Linux alapú operációs rendszerünkre a `sudo apt-get install mininet` segítségével letölthetjük a Mininetet. Ezen felül szükségünk lesz egy virtuális switch-re ami jelen esetben egy openvswitch, amit a `sudo apt-get install openvswitch-switch` parancs kiadásával telepíthetünk. A csomag települése után a terminálban kiadott parancsokkal tudjuk létrehozni az általunk elképzelt hálózatot.

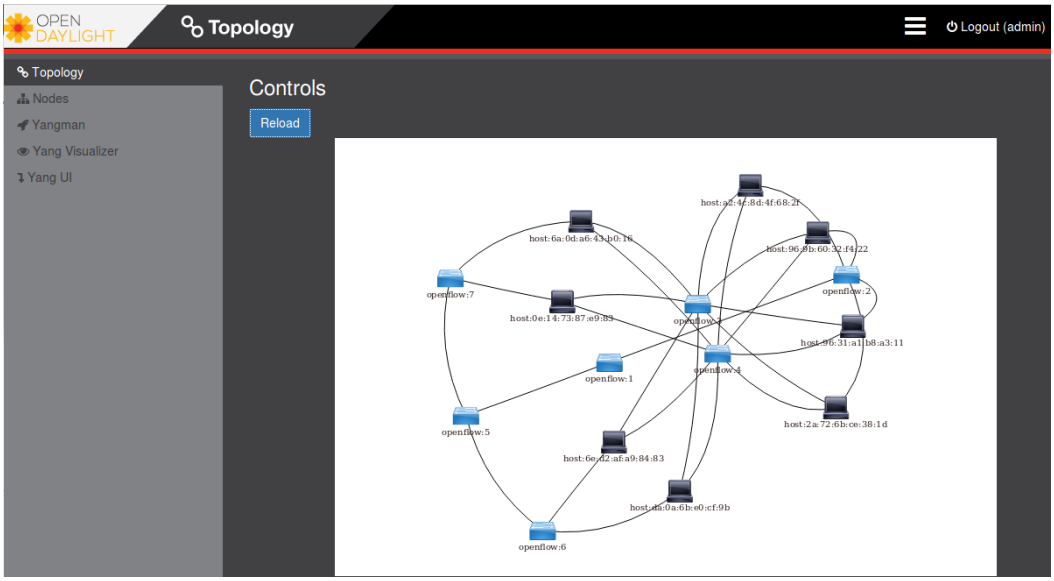
A második lehetőség, hogy egy külön erre a célra kialakított virtuális gépet töltünk le a Mininet honlapjáról. Ezt a virtuális gépet ezután beimportáljuk a Virtualbox felületén és már használhatjuk is a Mininetet.

Egy példa parancs:

```
$sudo mn –controller=remote,ip=192.168.56.101, --topo tree,3
```

```
mininet@mininet-vm: ~
mininet@mininet-vm:~$ sudo mn --controller=remote,ip=192.168.56.101 --topo=tree,
3
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.56.101:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6)
 (s5, s7) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet>
```

16. ábra Mininet beállítása és ping



17. ábra Mininet által létrehozott hálózat

Habár ez a megoldás roppant mód kényelmes tud lenni és igazán nagy hálózatokat is könnyedén tudunk vele szimulálni, ezáltal elég sokrétűen lehet használni, van egy nagy hátránya. Ennél a megvalósításnál a hálózat teljesen virtuális és egy

virtuális gépen fut minden, nagyon nehéz olyan következtetéseket levonni az esetleges mérések adataiból, amit utána a fizikai világban releváns mértékben tudunk hasznosítani, ezért számomra ez a megoldás nem felelt meg, így másik módszert kerestem az SDN alapú hálózat megvalósításához.

4.3 Kontroller és namespace

Ahhoz, hogy a megfelelő hálózati elrendezést meg tudjam valósítani, az egyes komponenseket is be kellett állítani. A kontrolleren a megfelelő modulok telepítése után nem volt egyéb teendő, így a switch-eket kellett konfigurálni.

4.3.1 Openvswitch beállítás

A hálózatomban Openvswitch [12] példányokat használtam, amelyekhez először szükség van egy operációs rendszerre, amire jelen esetben egy felhasználói felülettel rendelkező Ubuntut használtam. A virtuális gép létrehozásakor csináltam egy host-only adaptert, ami a kontroller felé néz, illetve ez által a Virtualboxot futtató gépről is be tudtam SSH-zni a switch-ekbe, amely megkönnyítette a menedzsment folyamatokat. Az openvswitch működéshez szükséges csomagokat egyszerűen *apt-get* parancs kiadásával telepítettem:

```
$sudo apt-get install openvswitch-switch
```

A munka során több parancsot is használtam a switchek beállításához:[13]

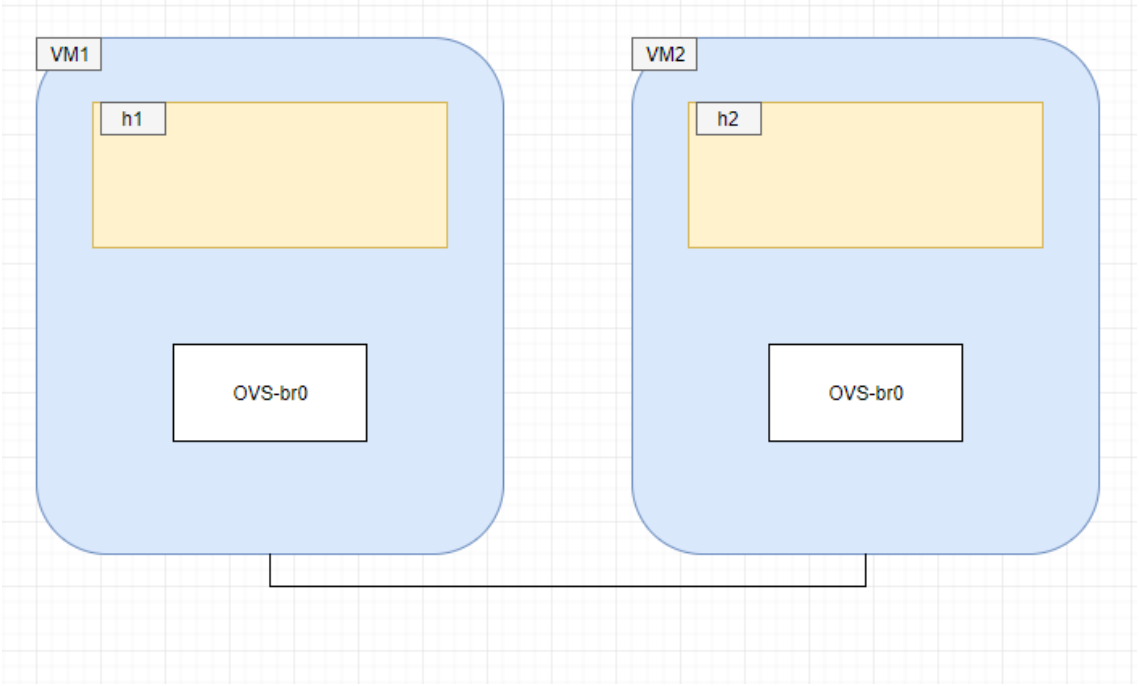
- `$ovs-vsctl add-br <név>:` Ezzel a paranccsal hozhatjuk létre a bridge-et amihez később portokat tudunk felvenni. Ez a bridge az operációs rendszerben egy interfészként jelenik meg, de a kontroller és az SDN szemléletéből valójában ezt tekintjük a switch entitásnak.
- `$ovs-vsctl add-port <bridge> <interfész>:` Arra szolgál, hogy az általunk választott bridge-hez/switch-hez hozzáadjunk egy választott portot.
- `$ovs-vsctl set-controller <bridge> <controller címe>:` Ennek segítségével tudjuk beállítani, hogy az eszközünk irányítását egy külső kontroller végezze el, ezáltal elegendő a kontrollerben beállítani a megfelelő csomagtovábbítási szabályokat.

- `$ovs-ofctl <bridge> <flow>:` Ennek a parancsnak használatával felvehetünk a switch-hez általános csomagtovábbítási szabályokat. Akkor használjuk, ha nincs kontroller, így az én elrendezésemben teszt jelleggel használtam ezt a parancsot.

4.3.2 Namespace-ek kialakítása, hálózat összekapcsolása

A hálózat kialakításához szükségem volt két virtuális gépre, amelyekhez Virtualboxban felvettem egy-egy olyan interfészt, amit a hoszt gép egy hálózati kártyájára kapcsoltam bridge-elt üzemmódban. A két virtuális gépre feltelepítettem egy-egy Virtualbox-ot is, majd a két virtuális gépen futtattam további Ubuntu Server virtuális gépeket, tehát egy egymásba ágyazott mérési elrendezést építettem ki.

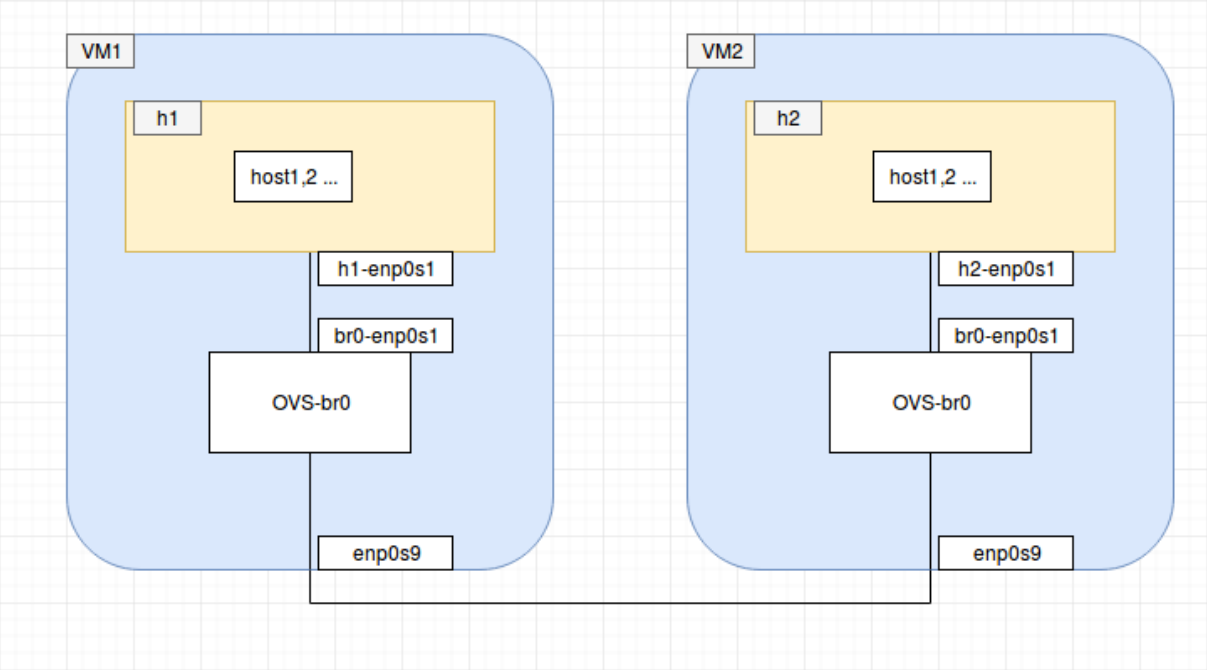
Egy hálózati namespace segítségével kialakíthatunk egy virtuális hálózatrészt, amely saját interfészekkel, IP címekkel, routing táblával és egyéb hálózati világból jól ismert lehetőségekkel rendelkezik. A megoldás során szükségem volt a namespace hozzáadásához, majd létrehoztam egy switch-et a virtuális gépen. Ezt mutatja a 18. ábra.



18. ábra Namespace és switch

Ezután a namespace-t és a switch-et összekötöttem egy patch kábellel, így kialakulhatott egy olyan jellegű kapcsolat, mintha fizikai eszközöket kötöttem volna össze. Ezután hozzáadtam a kontrollert, illetve a megfelelő portokat a switch-hez. Ahhoz, hogy harmadik rétegbeli kommunikációt is meg tudjak valósítani, IP címet

rendeltem a namespace-ekhez, majd a végső lépésként a hoszt gép "fizikai" portját is felvettem a switch-en.



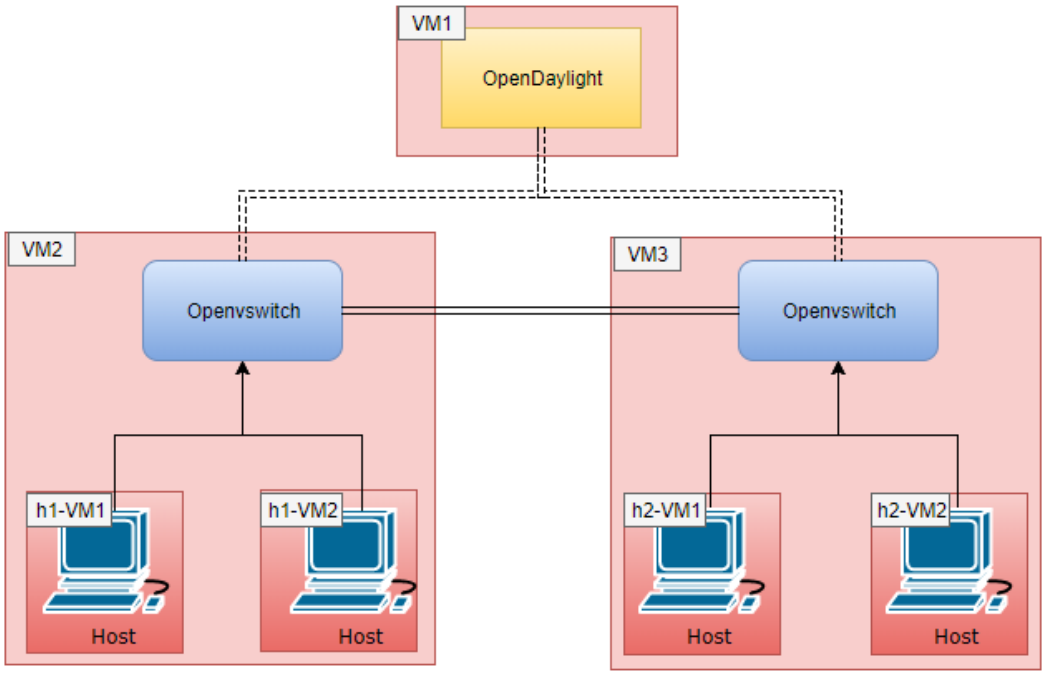
19. ábra A namespace megoldás terve, interfészekkel

Az openvswitcheket futtató virtuális gépeken a terminálon keresztül kiadott parancsokkal ejtettem meg a szükséges beállításokat:

Első gépen kiadott parancsok:

```
$ip netns add h1

$ovs-vsctl add-br br0

$ip link add h1-enp0s1 netns h1

$ip link set h1-enp0s1 type veth peer name br0-enp0s1

$ip link set h1-enp0s1 netns h1

$ovs-vsctl add-port br0 br0-enp0s1

$ip netns exec h1 ifconfig h1-enp0s1 192.168.1.1

$ovs-vsctl set-controller br0 tcp:192.168.56.101

$ifconfig enp0s9 0.0.0.0

$ovs-vsctl add-port br0 enp0s9
```

A másik virtuális gépen hasonló parancsokat használtam, néhány helyen módosítottam, hogy ne legyenek ütközések úgy, mint például IP címek, namespace nevek. Végül pingelés segítségével ellenőriztem a kapcsolatot.



20. ábra A virtuális gépek elrendezése namespace esetben

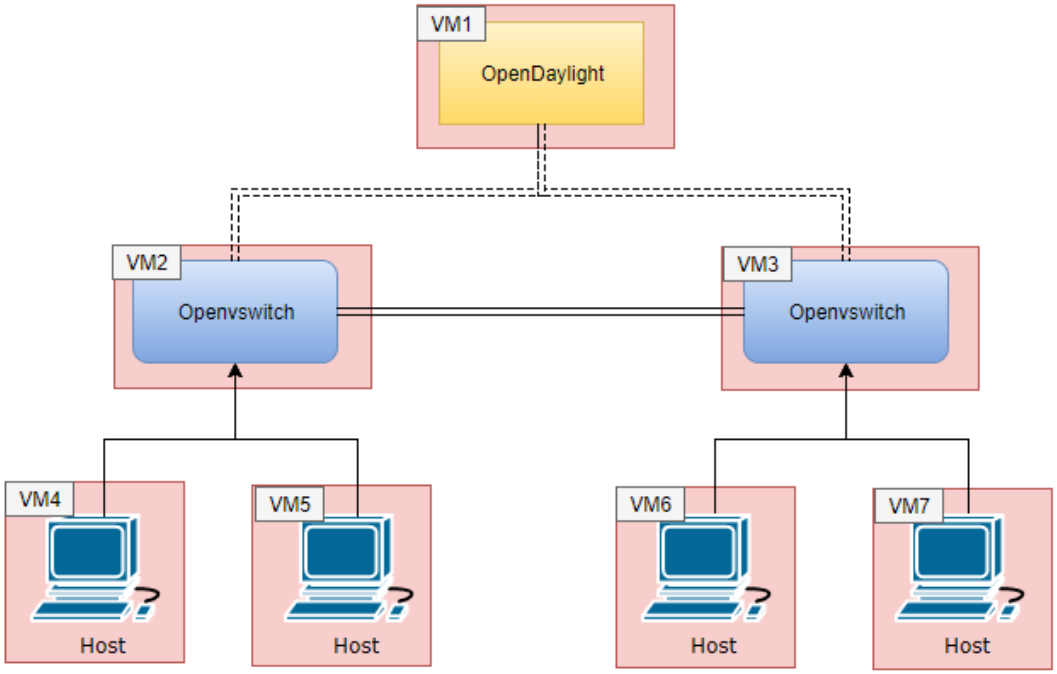
4.4 Külön virtuális gép minden eszköznek

Ebben az esetben újabb lépést teszünk afelé, hogy fizikailag összekapcsolt eszközök között alakítsunk ki egy SDN hálózatot. A 20. ábra jól mutatja az előző fejezetben tárgyalt namespace megoldásban használt virtuális gépek viszonyát. Ahhoz, hogy az egyes hálózati eszközök még jobb teljesítmény elérésére legyenek képesek a következő lépés, hogy minden eszköz külön virtuális gépre kerül.

A hoszt számítógépen egy Virtualbox-ot futtattam, amin minden egyes gépnek egy külön virtuális gépet kreáltam. Ebben az elrendezésben egy ODL kontroller, 2 Openvswitch és 4 hoszt gépet használtam. A 21. ábra mutatja azt az elrendezést, amit kialakítottam. A Virtualbox esetén lehetőségünk van különböző módon működő hálózati interfészeket hozzáadni a virtuális gépekhez. Ebben a hálózati elrendezésben úgy alakítottam ki a megfelelő részeket, hogy azok a valódi életbeli, fizikai elkülönülést

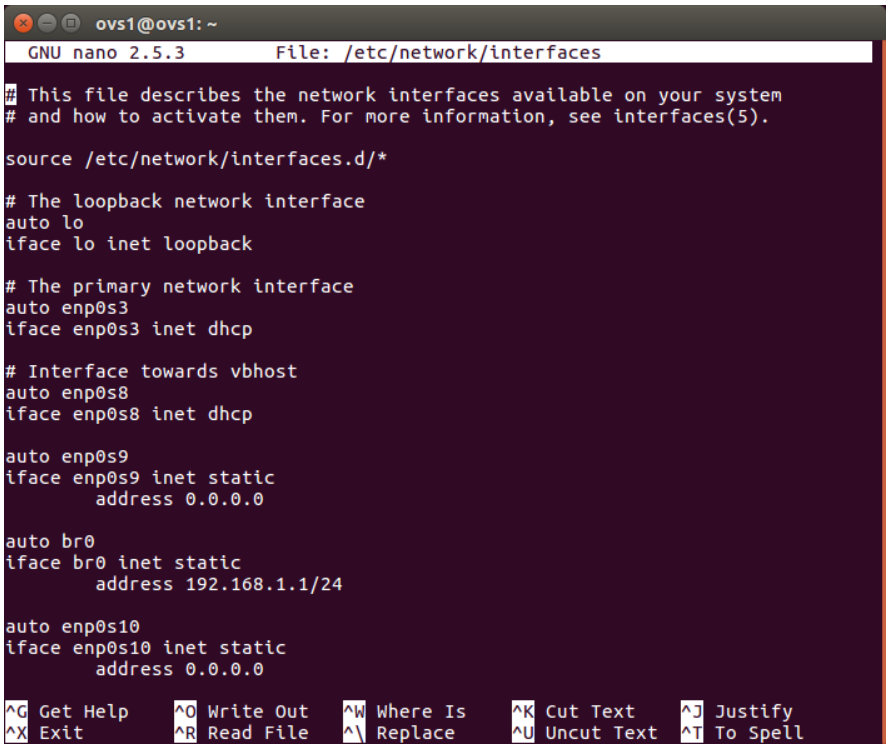
is megvalósítsák. Az egyik switchből és 2 kliens gépből kialakítottam egy „szigetet” amelyet a virtuális gépeket futtató hoszt gép egyik hálózati kártyájára csatoltam úgynevezett bridged üzemmódban, majd ugyanezt megtettem a másik gépekkel is egy újabb hálózati kártyával. Szerencsére a fizikai gépben volt három hálózati kártya is, ezáltal egyszerűbben tudtam szimulálni ezt a fizikailag elhatárolt szituációt.

A kontroller virtuális gépét a már korábban bemutatott módon alakítottam ki, ebben az esetben se volt semmilyen újabb beállításra szükség. Az összes többi gépre Ubuntu Server-t telepítettem és azokon dolgoztam tovább. A switchek esetében az előző részben bemutatott módon az *apt-get* parancs segítségével telepítettem a szükséges csomagot.



21. ábra Külön virtuális gép minden eszköznek

A kialakítás során szükséges volt az egyes eszközöknek saját IP címet adnom, amelyet a `/etc/network/interfaces` fájl módosításával tettem meg, ezáltal a beállított címeket megőrzi a virtuális gép és induláskor beállítja a megfelelő interfészeket. A 22. ábra bemutatja, hogyan állíthatunk be különböző interfészeket.



22. ábra Interfész beállításokra példa

A Virtualbox-ban beállított interfészek listáját az alábbi táblázat mutatja:

VM	Interfész	Megjegyzés
ODL	enp0s8	Host-only 192.168.56.101
OVS1	enp0s8	Host-only 192.168.56.102
	enp0s9	Bridged (hoszt enp9s0 hálózati kártyája) A másik switch felé néz
	enp0s10	Bridged (hoszt enp3s0 hálózati kártyája) A 192.168.1.11-12 gépek felé néző interfész
OVS2	enp0s8	Host-only 192.168.56.103
	enp0s9	Bridged (hoszt enp9s0 hálózati kártyája) A másik switch felé néz

	enp0s10	Bridged (hoszt enp12s0 hálózati kártyája) A 192.168.1.13-14 gépek felé néző interfész
host1	enp0s8	Bridged (hoszt enp3s0 hálózati kártyája) 192.168.1.11
host11	enp0s8	Bridged (hoszt enp3s0 hálózati kártyája) 192.168.1.12
host2	enp0s8	Bridged (hoszt enp12s0 hálózati kártyája) 192.168.1.13
host22	enp0s8	Bridged (hoszt enp12s0 hálózati kártyája) 192.168.1.14

1. táblázat Interfészek beállításai

A hálózatban használt eszközöket a már előző fejezetben is láthatott parancsokkal konfiguráltam. Az OVS1 nevezetű virtuális gépen kiadott parancsok:

$sudo ovs-vsctl add-br br0

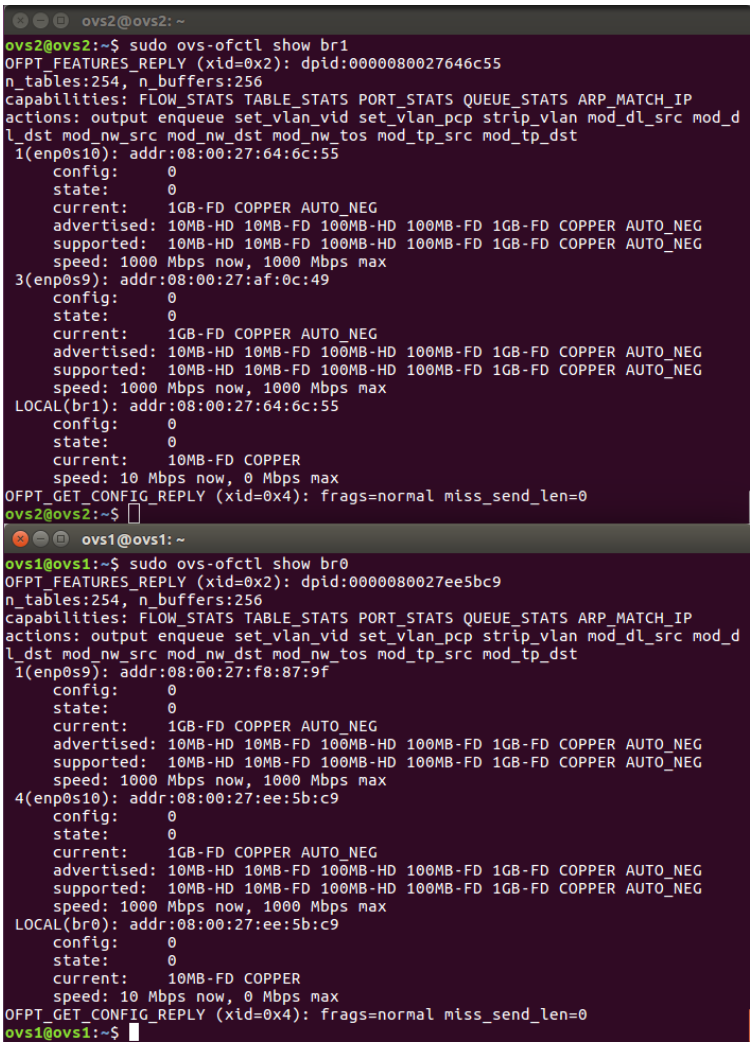
$sudo ovs-vsctl set-controller br0 tcp:192.168.56.101:6633

$sudo ovs-vsctl add-port br0 enp0s9

$sudo ovs-vsctl add-port br0 enp0s10

A tesztelések során eleinte a switch oldalán adtam meg azokat a flow-kat, amelyeknek segítségével a hálózatban megfelelően eljuthatnak a csomagok az egyik pontból a másikba. Ahhoz, hogy a flow-kat megfelelően lehessen irányítani, meg kellett határoznom a portok OVS által használt számozását.

$sudo ovs-ofctl show br0



23. ábra A portok ellenőrzése

Ennek a parancsnak a hatására megjelennek különböző interfészek, amelyek a switch-hez hozzá lettek adva és emellett láthatunk egy-egy számot is, ahogy a 23. ábra is mutatja. Ezután felvettem különböző flow-kat is:

$sudo ovs-ofctl add-flow br0 priority=100,in_port=1,actions:4,LOCAL

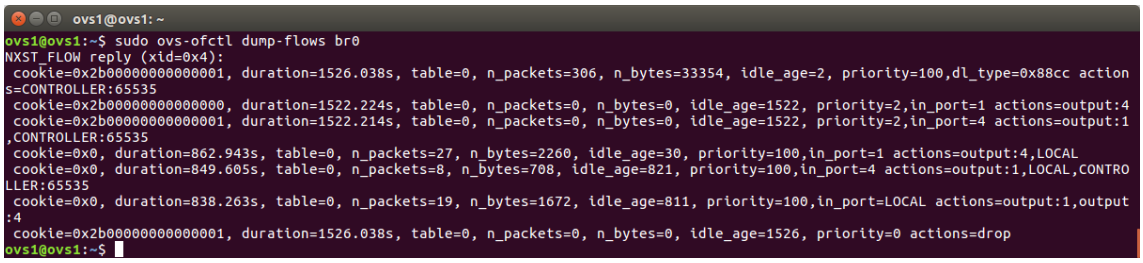
$sudo ovs-ofctl add-flow br0 priority=100,in_port=4,actions:1,LOCAL,CONTROLLER:65565

$sudo ovs-ofctl add-flow br0 priority=100,in_port=LOCAL,actions:1,4

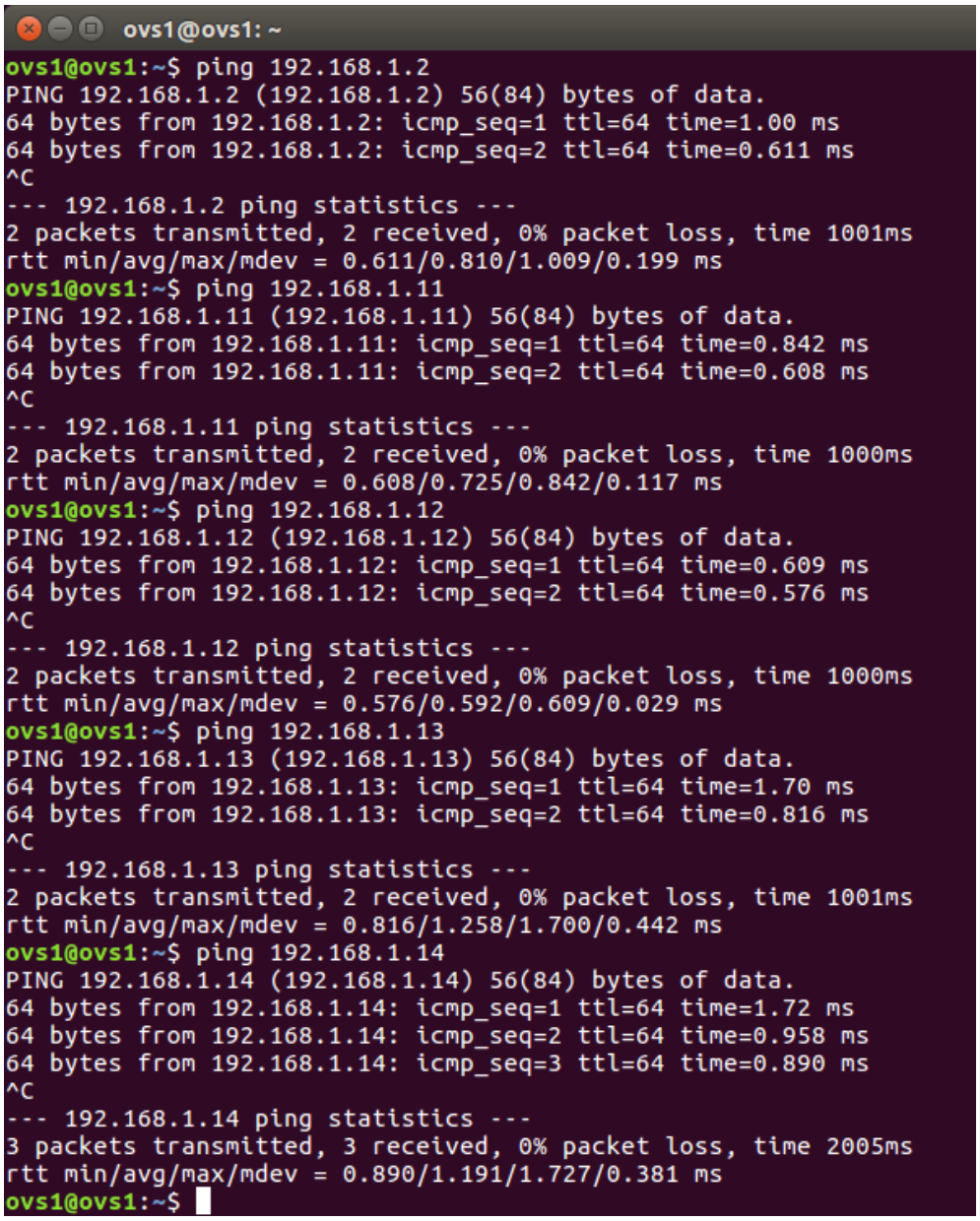
A flow-knál a LOCAL port felvételére azért volt szükség, hogy el tudjuk érni a .1-es címet is, melyet a bridge interfésznek vagyis a swtich-nek adtunk meg. Ez azért fontos még mert, így egy default gateway-t is megadhatunk a hálózatban lévő gépeknek,

így más hálózatba is át tudunk küldeni csomagokat. A flow készítése során lehetőségünk van "nw_dst=<IP cím>" kifejezést is írni, amellyel megmondhatjuk, hogy egy adott hálózatba küldött csomag esetében mit csináljunk a csomaggal, hova küldjük stb.

A hálózat kialakítása során minden gépet a 192.168.1.0/24-es hálózatra konfiguráltam fel. Ekkor eleinte a két switch tudta pingelni a szomszédos eszközöket, viszont azokat, amik már egynél nagyobb távolságra helyezkedtek el nem tudtam elérni. Ekkor módosítottam a switch-ekben lévő flow-kat méghozzá úgy, hogy azokat a csomagokat, amik bejönnek a portra, nem csak a LOCAL-ra, vagyis a switch-hez tartozó „*br*” interfészre továbbítottam, hanem a többi portra is. Ezt a 24. ábra mutatja be. Ekkor már tudtam bármelyik eszközről pingelni bármelyik másikat, az egyik eszközön el is végeztem ezt az ellenőrzést, ezt bizonyítja a 25. ábra.

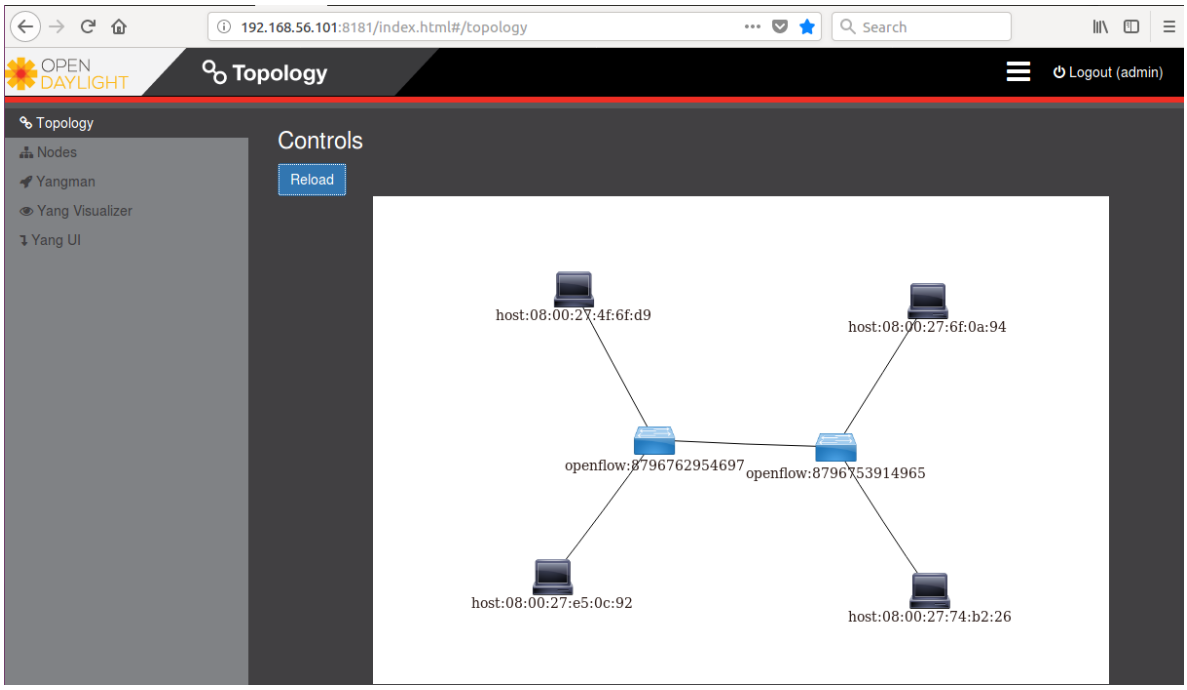


24. ábra Flow tábla az első switch-ben



25. ábra Minden eszköz eléri a másikat

A kontroller oldaláról is megvizsgálhatjuk a hálózati elrendezést a böngészőben, amelyet a 26. ábra mutat be.



26. ábra A hálózat megjelenítése OpenDaylight-ban

4.4.1 Flow bejegyzések küldése kontrollertől

Általánosan egy-egy switch-ben a megfelelő parancsok kiadásával tudjuk módosítani a flow táblát, viszont ez egy nagyobb hálózat esetén nagyon körülményes és pont azt a lehetőséget hagynánk ki, amit az SDN nyújt. A központi szabályozás segítségével a kontrollerből tudunk kiküldeni olyan üzeneteket, amelyekkel módosíthatjuk a switch-ek flow tábláját, ezért sokkal jobban felügyelni tudjuk a hálózatunkat.

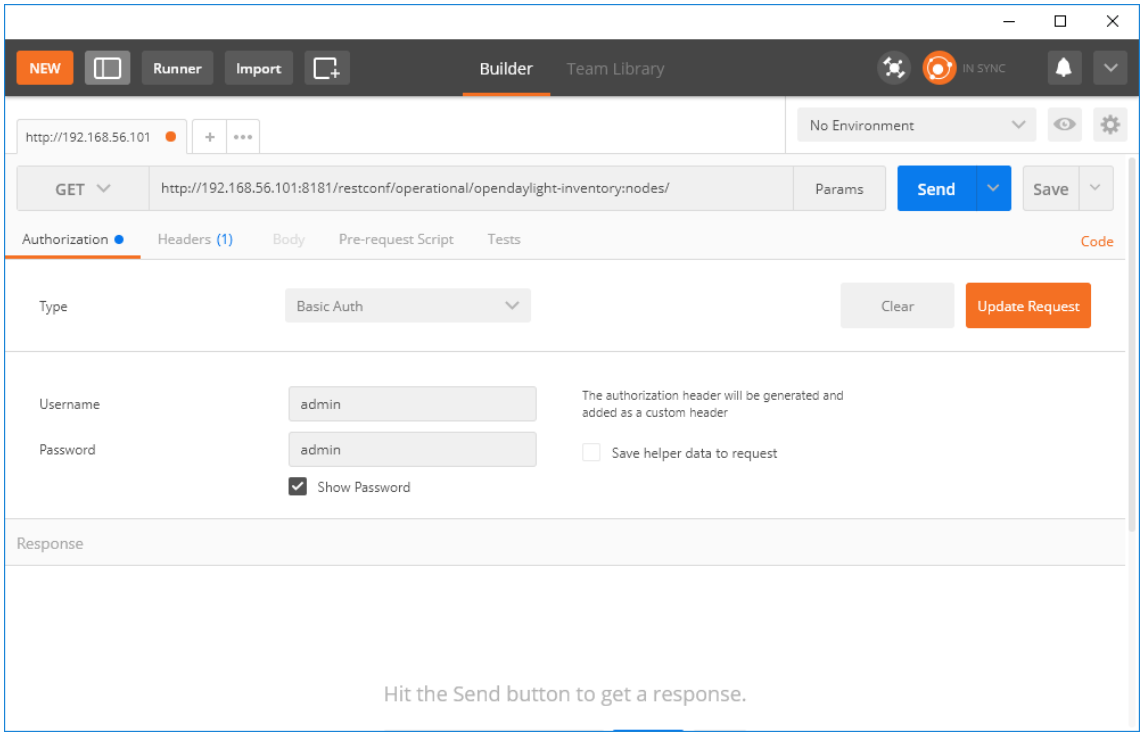
A Representational State Transfer vagy rövidítve REST [8] egy szoftverarchitektúra típus melyet általában a webes szolgáltatások kommunikációjának megvalósítására használnak. A felhő alapú szolgáltatások fejlesztésénél is nagyon sok szolgáltató alkalmazza, ugyanis kis súlyú kommunikációt igényel a felhasználó és a szolgáltató között. A jellemzően HTTP felett futó REST-nek van néhány megkötése:

1. Kliens-szerver architektúra: A szerverek és a kliensek el vannak választva egymástól, ezáltal nem foglalkoznak olyan feladatokkal, ami a másik oldalé lenne. Ennek az elválasztásnak köszönhetően külön lehet a két réteget fejleszteni, menedzselni.

2. Réteges felépítés: Jobban skálázhatóvá válik a rendszer, ha több rétegben építjük ki a kapcsolatunkat, egy kliens számára nem számít, hogy milyen útvonalon kapja meg a csomagokat.
3. Állapotmentesség: Minden kérés tartalmazza a kliensekről szükséges összes információt, ugyanis nincs eltárolva a szerverek oldalán a kliensek állapota.
4. Egységes interfész: Ez azért fontos, hogy az elválasztott kliens és szerver réteg egymástól külön is tudjon fejlődni, illetve egyszerűsíti a megvalósítást.
5. Gyorsítótárazhatóság: A kliensek számára meg kell adni a lehetőséget, hogy eltároljanak bizonyos információkat cache-ben, ezáltal ha jól van kezelve ez a gyorsítótárazás, rengeteg fölösleges interakciót elkerülhetünk, így sokkal jobb teljesítményt érhetünk el.

Ahhoz, hogy a kontrolleren keresztül módosítani tudjam a switch-ekben lévő flow-kat, először szükségem volt a POSTMAN nevű alkalmazás telepítésére, melyet a Google Chrome böngésző kiegészítői között találtam meg. A felületét a 27. ábra mutatja be. Egy flow injektálása előtt lekértem az összes node információt egy REST kérésen keresztül. Ehhez beállítottam header információt, autentikációs adatokat, illetve megadtam az URL-t is ahonnan lekértem ezeket az információkat.

- GET: http://192.168.56.101:8181/restconf/operational/opendaylight-inventory:nodes
- Header: Accept header – application/xml
- Authorization: Basic Auth, admin-admin



27. ábra POSTMAN felület, node információk lekérése

Ekkor a válasz részben megvizsgálhattam a hálózatban lévő node-okat amelyeket ezután az azonosítójuk megadásával egyesével is megtekinthettem. Így például az előző URL-t kiegészítve a *http://192.168.56.101:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1* címről lekérhetem az adott node-hoz tartozó információkat.

A flow-k injektálása ezután a megfelelő node ID beírásával egyszerűen történik. A POSTMAN-ben egy PUT kérést indítottam a megfelelő URL-lel, header beállításokkal, illetve egy előzetesen összerakott XML törzzsel.

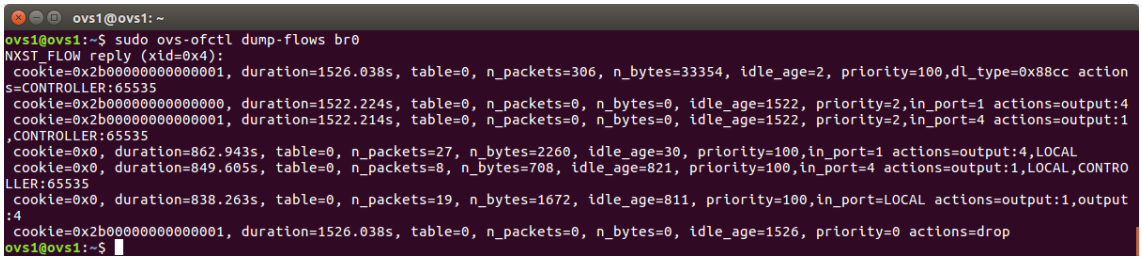
- PUT: http://192.168.56.101:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1
- Header:
    - Content-Type: application/xml
    - Accept: application/xml
    - Authorization: Basic Auth, admin-admin

- Body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
   <priority>4</priority>
   <flow-name>OVS1FLOW</flow-name>
   <match>
<in-port>4</in-port>
   </match>
   <id>1</id>
   <table_id>0</table_id>
   <instructions>
      <instruction>
      <order>0</order>
      <apply-actions>
        <action>
        <order>0</order>
        <output-action>
          <output-node-connector>
             LOCAL
          </output-node-connector>
          <output-node-connector>
            1
          </output-node-connector>
          <output-node-connector>
             CONTROLLER
          </output-node-connector>
          <max-length>60</max-length>
        </output-action>
        </action>
      </apply-actions>
      </instruction>
```

```
</instructions></flow>
```

Ha megtekintjük ezek után a switch flow tábláját, akkor láthatjuk, hogy bekerült a flow a megfelelő helyre. Erre bizonyíték a 28. ábra.



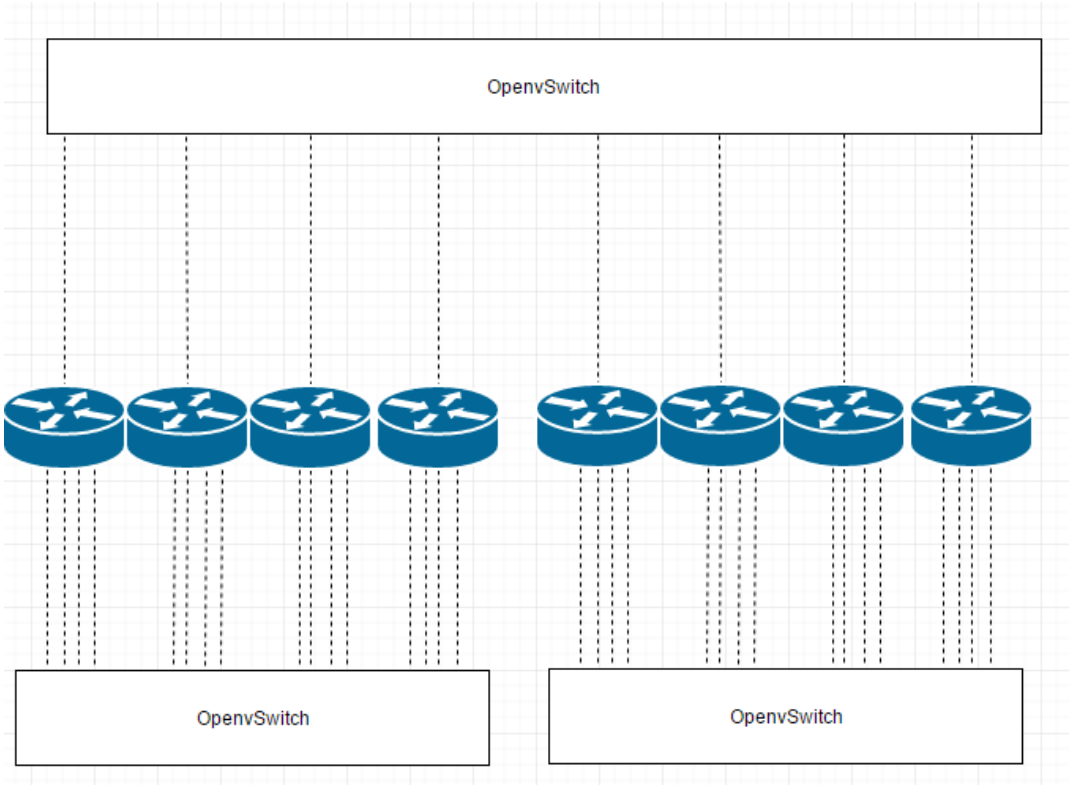
28. ábra A kontroller által pusholt flow bekerült a switch flow táblájába

Természetesen a fent feltüntetett flow egy nagyon egyszerű példa és ennél sokkal bonyolultabb szabályokat is küldhetünk a switch-nek, akár IP címek vizsgálatára vonatkozóan, akár vlan címkék módosítására is lehetőségünk van, ha a megfelelő XML tag-ek között jelöljük meg ezeket. A kontrollerből minden korábban bemutatott flow-t, amit eddig a switch-ekben állítottam be, a fenti módon, a törzs rész módosításával elküldtem a kontrollerből a switch-ekbe.

4.5 Fizikailag összekötött eszközök hálózata

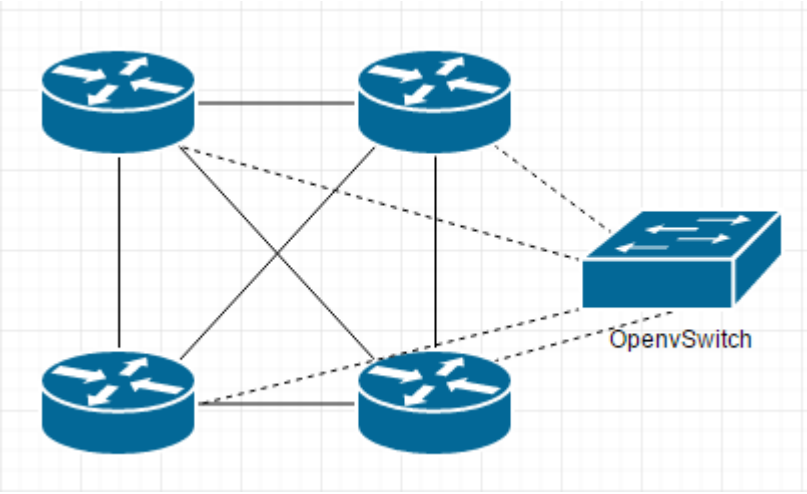
A hálózati megvalósítás következő lépcsőfoka a hálózati eszközök fizikai világba való átemelése, ahol ezután akár legacy eszközökkel is összekapcsolhatjuk őket. Erre a lépésre már nem jutott időm, viszont a tervezés fázisában elkészítettem pár tervet. A hypervisor rendszernek egy ESXi-s megoldást terveztem, amiben az eddigi SDN-es eszközök mellett a legacy routereket is virtualizáltan egy-egy megfelelő image telepítésével oldottam volna meg. Az ESXi egy úgy nevezett egyes típusú hypervisor, mely azt jelenti, hogy a hoszt számítógépünkre úgy telepítjük, mint egy operációs rendszer.[11] A Virtualbox ezzel ellentétben második típusú, ami azt jelenti, hogy szüksége van egy operációs rendszerre, ahova telepíthetjük és utána futtathatjuk. Ebben a lépésben az előző fejezethez hasonlóan minden egyes eszköz külön fut virtuálisan, a különbség, hogy az ESXi-ben futtatva ezeket a példányokat sokkal jobb teljesítmény érhetünk el, hiszen ez a hypervisor arra van optimalizálva, hogy ilyen virtualizációkat kezeljen. Emiatt a tesztelések sokkal közelebb kerülnek a valós eszközök által alkotott hálózatokon végzetthez.

A legacy routerekből kialakított hálózathoz készített tervet a 29. ábra mutatja be.



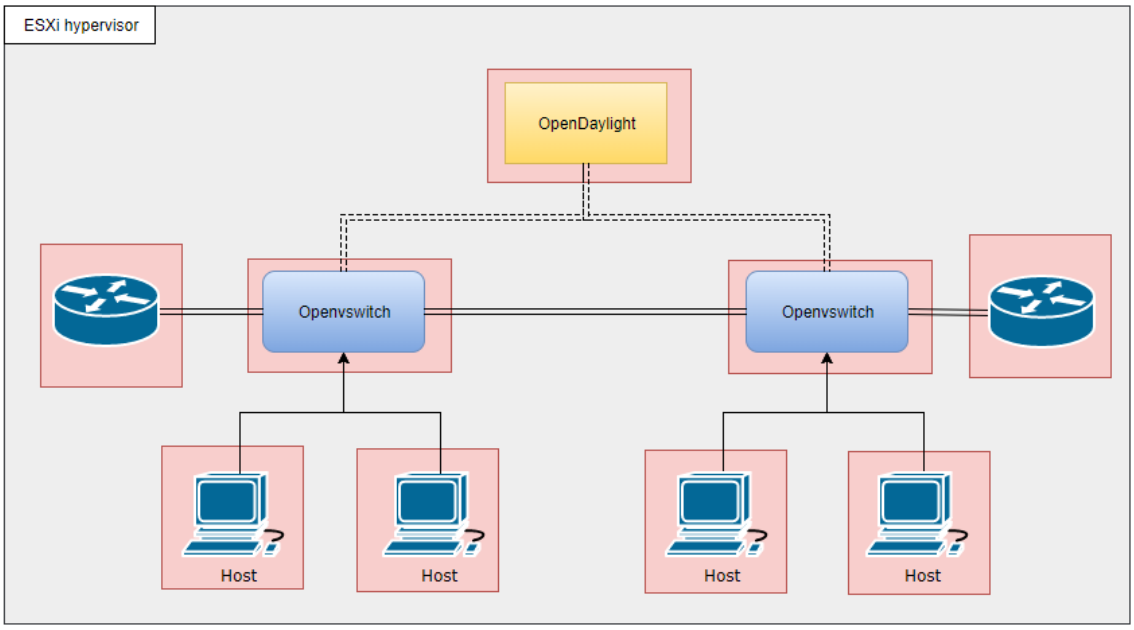
29. ábra Legacy eszközös elrendezés

A kialakításban a 29. ábra alsó felében lévő switch viselkedéséből adódóan a megfelelő interfészek beállításával akár full-mesh hálózatot is kialakíthatunk, míg a felső switch segítségével konfigurálhatjuk egyesével az eszközöket. A 30. ábra mutatja a logikai kinézetét a hálózatnak.



30. ábra Full-mesh hálózat

Az eddigi fejezetek alapján a 31. ábra mutatja be azt az elrendezést, amit az ESXi környezetben létrehozhatunk. Mivel terv szinten dolgoztam ki, ezért a két szélen lévő router ikon jelöli a legacy eszközöket, melyekből külön is alkothatunk egy hálózatot akár, az ábrán az egyszerűség kedvéért ezt nem tettem meg.



31. ábra Hálózat megvalósítása az ESXi környezetben

5 Hibrid hálózat megvalósítása OSPF segítségével

A hibrid SDN hálózat kialakítása érdekében foglalkoztam azzal, hogy lehet a legjobban leszimulálni egy fizikailag összekapcsolt hálózatot, ezáltal minél közelebb kerülni ahhoz, hogy a virtuális switch-ek valós, fizikai eszközökkel legyenek összekötve. Mivel a feltételezésem az volt, hogy olyan fizikai eszközöket kötnék a rendszerbe, amelyek nem képesek Openflow alapú kommunikációra, szükséges volt egy ötlet, hogy milyen más módon legyen megoldva az üzenetcsere.

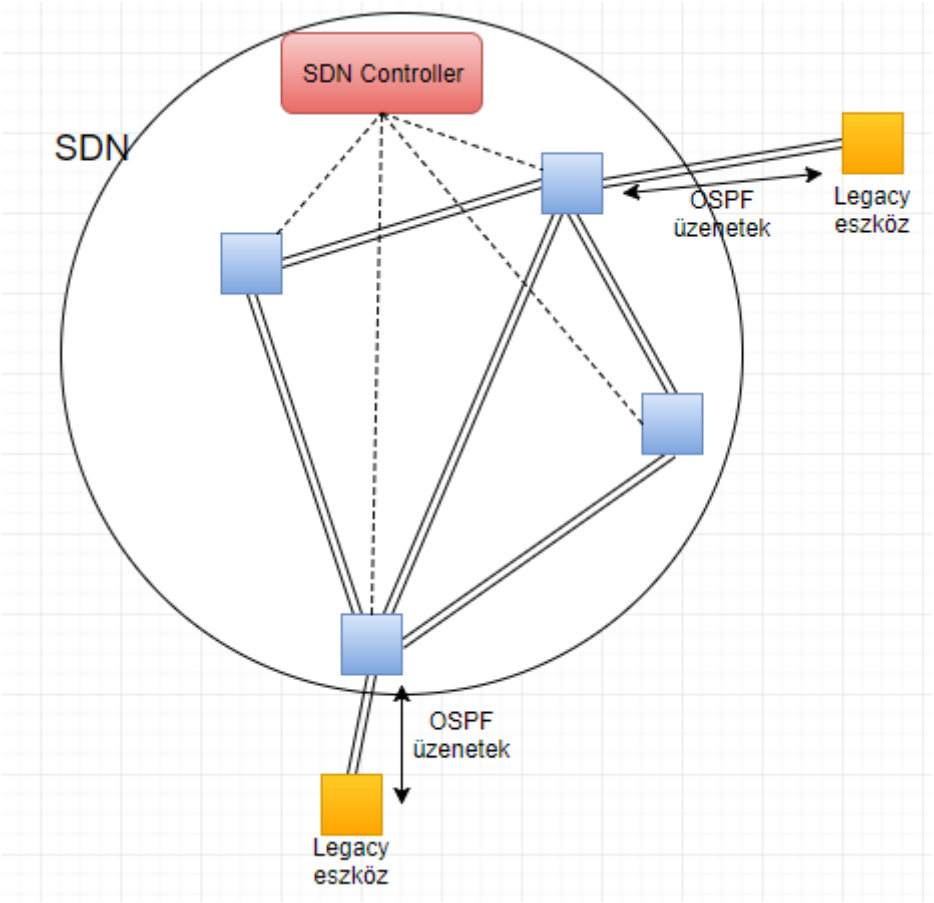
Az Opendaylight szabad kezet ad az alkalmazások fejlesztéséhez, így ötletként egy olyan alkalmazás fejlesztése merült fel, amelynek segítségével a kontroller képes lenne OSPF üzenetek kezelésére és küldésére. Az Openflow-ra nem képes routerek esetében szinte kivétel nélkül alkalmazható az OSPF, ezért döntöttem úgy, hogy ezt a protokollt kell megvalósítani a kontrollerben.

A legacy és SDN hálózatok közötti OSPF kommunikáció leírására a 32. ábra lesz segítségünkre. A koncepció szerint, a legacy eszközök OSPF kommunikáció használatára konfigurált eszközök, amelyek fizikai kapcsolatban állnak az SDN hálózatunkban lévő eszközökkel.

Az implementációhoz tartozó alapvető funkciók, amelyeket meg kell valósítani a kontroller oldalán:

1. Csomag küldése kontrollerből: A kontroller fog az OSPF üzenetekre válaszüzenetet generálni, majd elküldeni, ehhez kell egy csomagküldő szolgáltatást implementálni
2. OSPF szomszédosság kiépítés: A hálózati határon lévő eszközhöz beérkező OSPF üzenetekre a protokollnak megfelelő választ kell adni
3. A kontrollerben ki kell számolni a legrövidebb utakat.
4. A kontroller által irányított hálózatrészről tárolt link állapotokat terjeszteni kell
5. Az üzenetek továbbításához a különböző hálózatok határán lévő switch-ekbe olyan flow-kat kell implementálni, amelyek akár MAC, IP címek

módosításával, de továbbítják a megfelelő információkat a szomszéd legacy eszközöknek.



32. ábra Legacy és SDN hálózat kommunikáció

5.1 OSPF

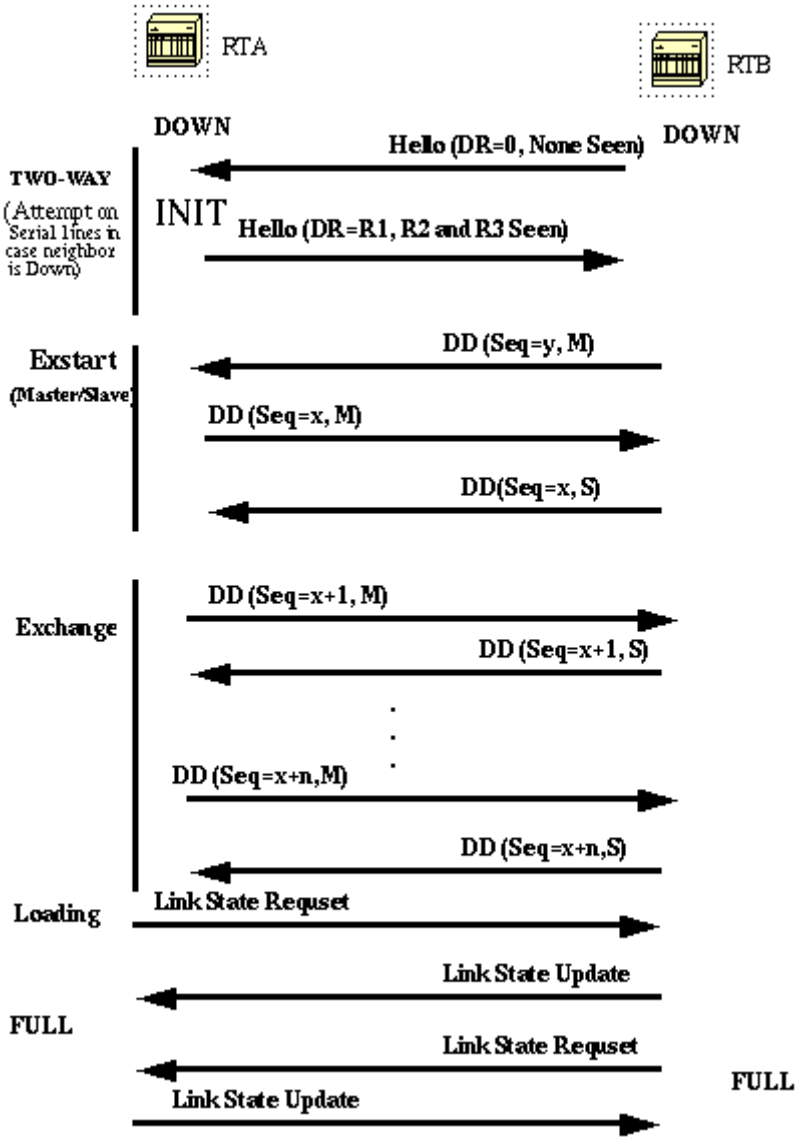
Az Open Shortest Path First [14] egy kapcsolat állapot (link-állapot) alapú protokoll, melynek az a lényege, hogy egy adott részén a hálózatnak routing információkat terjesszen, ezáltal kiépítve a forgalomirányítást. A link állapot alatt azt értjük, hogy az eszköz egyik interfészéhez tartozó információkat, mint például az IP cím, alhálózati maszk, hálózat típusa, amihez csatlakozik, kapcsolódó routerek stb. lekérdezzük és ebből az információkat adatbázisba gyűjthetjük. Az OSPF ahogy a nevében is benne van, legrövidebb utakat keres, amelyhez Dijkstra algoritmusát

használja. A routerek által használt algoritmus nagyvonalakban a következő módon épül fel:

1. Kezdetben, vagy ha változás áll be a routing információkban az egyik router egy úgy nevezett link-state üzenetet generál, amely tartalmaz minden routerhez köthető link állapot információt.
2. Minden router elárasztásos módszerrel kicseréli a link-state üzeneteket egymás között. Ha egy router kap egy ilyen információkkal teli csomagot, akkor bemásolja a saját adatbázisába, majd továbbküldi a csomagot.
3. Miután minden router adatbázisa elkészült, a router kiszámolja a Dijkstra algoritmus segítségével a legrövidebb utat az összes csomóponthoz.
4. Amennyiben nem történik változás a hálózatban az OSPF nem tesz semmit.

A link-state csomagoknak különböző fajtái vannak attól függően, hogy a hálózat mely részében használja az OSPF.

- Router Links: Egy adott területhez tartozó router információt írja le a csomag.
- Summary Links: Ezeket az üzeneteket olyan routerek generálják, amelyek különböző hálózati területek határán állnak, ezáltal olyan információk jutnak el a különböző területekre, amellyel elérhetjük a másik hálózatrészt
- Network Links: Az úgy nevezett Designated Router (DR) által generált üzenetek, amelynek segítségével információt kapnak az eszközök az összes routerről amelyik ugyanahhoz a multiacces szegmeshez tartoznak.
- External Links: Ezek az üzenetek a külső hálózatok eléréséről adnak bővebb információkat.

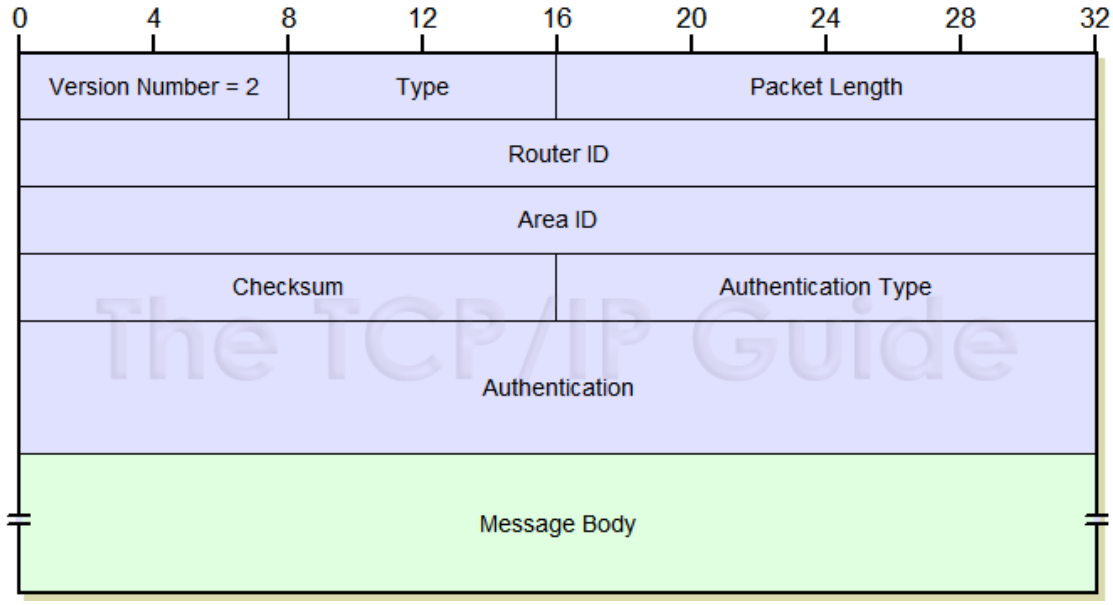


33. ábra OSPF üzenetváltások

A 33. ábra bemutatja, hogy az OSPF protokoll használata során milyen üzenetek váltása történik meg az egyes routerek között. Látható az ábrán, hogy a routerek különböző állapotokon mennek át attól függően, hogy milyen üzenetek érkeznek hozzájuk. Kezdetben a Hello protokoll segítségével történik a szomszédok közötti kommunikáció. Ebben a szakaszban, ha egy router az egyik szomszédja hello csomagjában látja önmagát megszólítva, átlépünk a „2-Way” állapotra. Az Exstart állapotban a két router megegyezik egy szekvencia számban, amelynek segítségével később fel lehet ismerni a már lejárt, vagy duplikálódott link állapot üzeneteket. Az Exchange állapotban úgynevezett Database Description (DD) csomagokat cserélnek a

routerek, amelyek rövidített link állapot leíró üzenetek. Ezek elég információt hordoznak ahhoz, hogy felismerhető legyen egy kapcsolat. A Loading állapotban link állapot kérő csomagokat küldenek egymásnak a szomszédok, amiben olyan információkat kérnek, amelyeket már felfedeztek, de még nem kaptak meg. A Full állapotba kerülés után a megfelelő routerek között kialakul a szomszédossági kapcsolat és a link-state adatbázisokban egyértelműen meghatározható az, hogy melyik router melyikkel áll kapcsolatban.

Az OSPF üzenet formátumát a 34. ábra mutatja be. A csomagban a Type mező adja meg, hogy mi az üzenet típusa. Az előző bekezdésben foglaltak alapján ez lehet Hello(1), Database Description(2), Link State Request(3), Link State Update(4), Link State Acknowledgement(5).

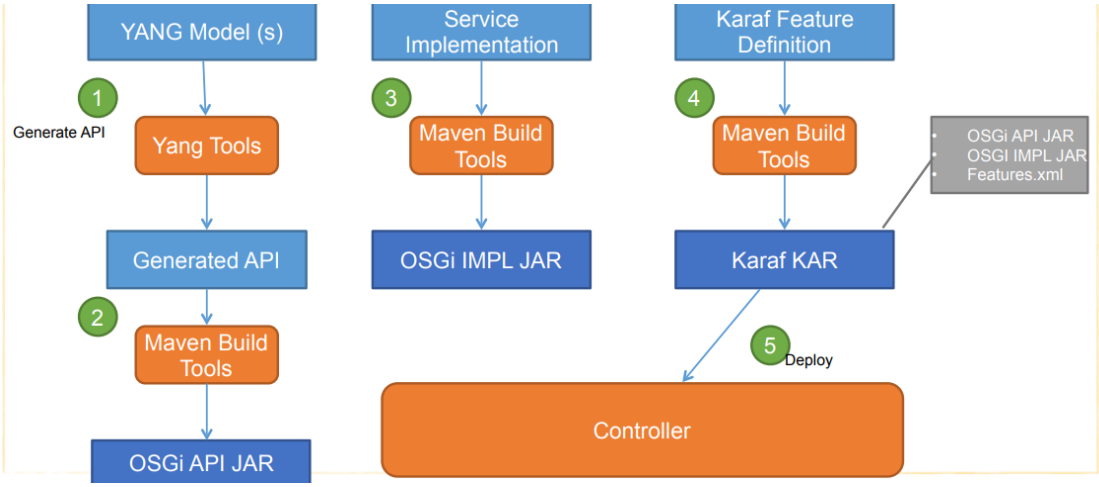


34. ábra OSPF üzenet felépítése [16]

5.2 Opendaylight módosítások

5.2.1 Az ODL-ben használt technológiák

Az Opendaylight [9] esetén több technológiát ötvöznek a különböző alkalmazások készítéséhez. A főbb programozói nyelv a Java, a modellek leírásához YANG-ot használunk, a buildet Maven segítségével oldjuk meg, az alap platform a Karaf, illetve a moduláris rendszerek készítéséhez OSGi-t használjuk. Az új szolgáltatások létrehozásának lépéseit nagyvonalakban a 35. ábra mutatja be.



35. ábra A szolgáltatások fejlesztésének lépései

A YANG [15] egy modell leíró nyelv, ami bemutatja az adatok hierarchikus elrendeződését egy fa segítségével, ahol minden csomópont névvel és értékkel vagy gyerek csomópontokkal rendelkezik. A YANG Tools eszközzel Java kódot tudunk generálni a megadott modellből. A YANG sok féle információt le tud írni, egy példa egy modellre, ami képes csomópontokat és azon található csatlakozókat leírni:

```
module opendaylight-inventory {
    namespace "urn:opendaylight:inventory";
    container nodes {
        list node {
            key "id";
```

```
                leaf id {
                    type node-id;
                }
                list "node-connector" {
                    key "id";
                    leaf id {
                        type node-connector-id;
                    }
                }
            }
        }
    }
```

Ahhoz, hogy megkezdhessük a fejlesztést, szükségünk lesz a következő csomagok telepítésére:

- Maven 3.1.1 vagy frissebb verzió
- Java 7-et vagy Java 8-at támogató JDK
- Egy megfelelő Maven settings.xml fájlra, amit az Opendaylight github-járól letölthetünk
    - `cp -n ~/.m2/settings.xml{,.orig} ; \wget -q -O - https://raw.githubusercontent.com/opendaylight/odlparent/stable/carbon/settings.xml > ~/.m2/settings.xml`

5.2.2 Üzenetküldés

Üzenetek küldését a Data Packet Service segítségével tehetjük meg, erről bővebben írtam a 2.2.1.1 fejezetben, kifejtve az egyes interfészek feladatát, illetve a rendszer működési elvét. Mivel alkalmazásként kell létrehozni ezt a funkciót, ezért vonatkoznak rá az előző fejezetben bemutatott alapok. Ha példaalkalmazást szeretnénk

megtekinteni, akkor az LLDP, vagy ARP Handler alkalmazásokat kell megnéznünk, azokban ugyanis vannak csomagküldések.

5.2.3 Legrövidebb út számolása, link állapotok

Az Opendaylight esetében Service Abstraction Layer alapon implementáltak egy switch-et, amely képes tanulni és optimalizálni tudja a csomagok továbbítását. Ezt L2Switch-nek nevezik. A L2Switch működése során több komponenst is használ:

1. *PacketHandler:*
    - Megvizsgálja a csomagokat a MAC-port párok kialakítása miatt
    - Értesíti az AddressTracker-t az új MAC-port párokról
    - Amikor kiderül egy csomag forrás és cél címe értesíti a FlowWriterService-t az új flow-król.
    - Megkeresi, hogy melyik portokra kell küldeni üzenetet, ha floodolni kell.
2. *AddressTracker:* Egy fában tárolja a MAC-Port párokat
3. *InventoryService:* A csomópontokról és a csomópontokhoz tartozó csatlakozókról ad információt
4. *FlowWriterService:*
    - Csomag továbbításához szükséges flow-kat ad az SAL adatokat tároló fájához.
5. ***TopologyLinkDataChangeHandler:*** Figyel a topológiaváltozásokra és értesíti a NetworkGraphDijkstra komponenst a változásokról.
6. ***NetworkGraphDijkstra***: Rendben tartja a hálózati gráfot és kiszámolja a legrövidebb utat a csomópontok között.

Ezek közül a komponensek közül a vastaggal kijelöltek mindenképpen érdekesek az új projekt megvalósítása során, hiszen szükség van olyan jellegű feladatok ellátására, amit ezek a komponensek megvalósítanak.

5.2.4 Flow által módosított csomagok

Ahogy azt már a 4.4.1 fejezetben is bemutattam lehetőségünk van a kontrollerből flow bejegyzéseket küldenünk a switch-ekbe, ezáltal irányítjuk a forgalmat. A forgalomirányítás mellett lehetőség van különböző adatok módosítására is flow-k segítségével, így például IP vagy MAC címet is átírhatunk az alábbi módon:

IP cím átírása:

```
<action>
    <order>0</order>
    <set-nw-dst-action>
       <ipv4-address>10.0.0.7/32</ipv4-address>
    </set-nw-dst-action>
</action>
```

MAC cím átírása:

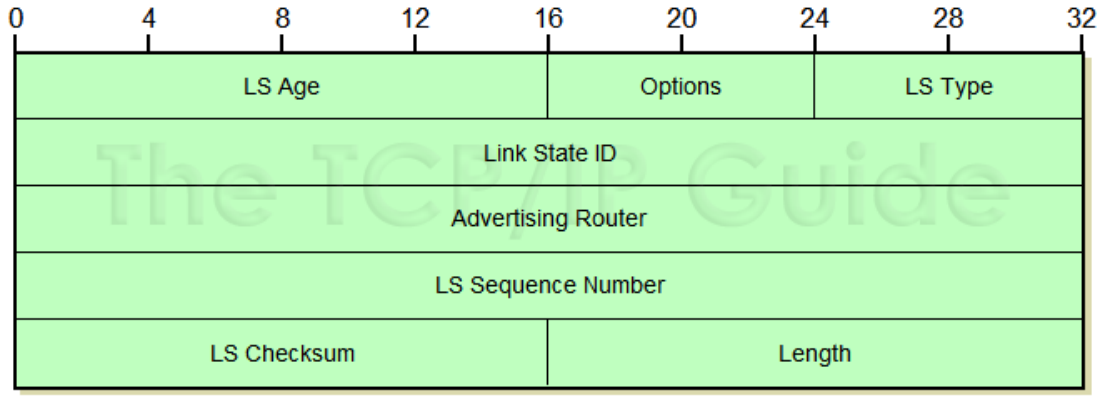
```
<action>
    <order>0</order>
    <set-dl-dst-action>
       <address>01:02:03:04:05:06</address>
    </set-nw-dst-action>
</action>
```

5.3 Az alkalmazás specifikációja

5.3.1 SDN hálózat link állapotainak terjesztése

Ahogy azt már láthattuk az SDN kontroller alapból tárol olyan információkat a hálózatról, mint például a linkek állapota, vagy a csomópontok közötti legrövidebb utak. Ezeket az információkat az SDN hálózaton kívülre is terjeszteni kell, így a kontrollerben készítsünk Link State csomagokat. A csomag header-höz tartozó információkat a 36. ábra, a 2. táblázat és a 3. táblázat mutatja. Az LSA body részéhez a

hivatalos dokumentációban [14] találhatunk információkat. Ezekkel a csomagokkal tudjuk terjeszteni az SDN hálózatunkban lévő kapcsolatok állapotát.



36. ábra Link State Advertisement header formátuma

Almező	Méret (byte)	Leírás
LS Age	2	LSA létrehozása óta eltelt idő
Options	1	Jelzi, hogy milyen opcionális OSPF lehetőségeket támogat az eszköz.
LS Type	1	Az LSA által leírt link típusa. Bővebben 3. táblázat
Link State ID	4	Meghatározza a linket, általában IP cím
Advertising Router	4	Az LSA forrásának ID-ja
LS Sequence Number	4	Régi vagy lejárt LSA-k meghatározására használt szám.
LS checksum	2	Adatok módosulásának felismerésére alkalmazott ellenőrző összeg.
Lenght	2	LSA hossza

2. táblázat LSA header mezők magyarázata

Érték	Link Típus	Leírás
1	Router-LSA	Link router felé
2	Network-LSA	Link hálózat felé
3	Summary-LSA (IP network)	Ha vannak területek (area) használva, akkor egy hálózatról ad összegző információt.
4	Summary-LSA (ASBR)	Ha vannak területek (area) használva, akkor egy határon lévő router-hez csatlakozó linkről ad infót.
5	AS-External-LSA	Az adott hálózati részen kívül eső link infó.

3. táblázat A link state állapotai

Ezek alapján, ha lekérjük egy csomópontunk kapcsolatainak állapotát, akkor generálhatunk csomagokat ezekről az információkról és elküldhetjük a hálózati határon lévő eszköznek.

5.3.2 OSPF válaszüzenetek

Ha a nem SDN-es hálózatból OSPF kommunikációra felszólító üzeneteket kapunk, akkor arra a protokollnak megfelelő válaszüzenetet kell küldeni. Ha beérkezik az üzenet, akkor a fogadó eszköz, továbbítja a kontrollernek és a kontroller visszaküld egy választ ugyanazon az eszközön keresztül. Ezzel kiépül az OSPF szomszédossági kapcsolat és tudunk link állapot információkat küldeni, fogadni. A protokollhoz tartozó üzeneteket a korábbi fejezetben találhatjuk meg. Érdemes lehet megjegyezni azokat az eszközöket, amelyekről ilyen OSPF kérések érkeznek a kontroller felé, ugyanis később ide fogjuk mindig visszaküldeni a legacy hálózatba szánt csomagokat.

5.3.3 Flow-k módosítása

Az üzenetküldések során mindig a kontrollerből küldünk ki információkat, így például az egyes csomópontokhoz tartozó szomszédossági információkat is. Ahhoz, hogy az OSPF megfelelően ki tudja alakítani a hálózati topológiát, az SDN részen mindig módosítanunk kell a csomagokban lévő forráscímet. Ezáltal a kontroller elfedi a

saját működését és a kinti hálózatban úgy látszik, mintha az egyes eszközök maguktól válaszolnának az OSPF kérésekre.

5.3.4 Útvonalak számolása

A kontrollerbe érkező OSPF információk alapján a kontroller kiszámolja az egyes csomópontokhoz tartozó legrövidebb utat, és ha ezzel megvan, akkor ezek alapján flow-kat küld az eszközöknek. Az eszközök ezek után bármikor kapnak egy olyan csomagot, ami a flow-ra illeszkedik, akkor továbbítani fogják a megfelelő szabály szerint.

5.3.5 Lehetséges hibák

Az implementáció nélkül nehéz megmondani mi a rendszer gyengesége, de mindenképpen érdemes lehet meggondolni, hogy milyen hálózati forgalmat generál a fenti megoldás. A kontroller számára rengeteg feladat jut, hiszen számolnia kell útvonalakat, flow-kat kell beállítani, üzeneteket kell küldenie. Érdemes lehet esetleg valamilyen elosztott megoldást alkalmazni, így egy esetleges terhelés miatti leállás kockázata csökkenthető.

Természetesen meg kell vizsgálni azokat az eseteket is, amikor valamelyik eszköz meghibásodik. Ha egy SDN hálózati eszköz meghibásodik, akkor jó eséllyel a kontroller ezt észreveszi és a szomszédjai nevében küld egy-egy LSA-t a hálózat többi részének. A kontroller meghibásodása okozhatja azt, hogy az egész SDN-es hálózati részről nem lesz információ a kinti hálózat részére. Ugyanez az eset akkor is megeshet, ha olyan switch-ek romlanak el, amelyek nélkül a kontroller nem éri el a határon lévő switch-et.

5.4 Implementáció, verifikáció és analízis

A megvalósításhoz szükséges szerverek beszerzése annyira elhúzódott, hogy nem jutott idő megvalósítani minden feladatot a dolgozatomban. A további feladatokat a fent vázolt mechanizmusok és specifikációk segítségével kell megvalósítani. Ezek a további munka részét képezik. A megvalósítás után még további feladatok adódhatnak, amelyek közé tartozik a rendszerhez tartozó különböző mérések elvégzése, amelyekkel a további fejlesztésekhez, kutatásokhoz lehet információkat nyújtani.

6 Összefoglalás

A dolgozatomban széles körben foglalkoztam azokkal a paradigmákkal, amelyek a Software Defined Networking alapjait képezik. Mivel a hálózati területen ez az irányvonal nagyon erősen fejlődik, fontos, hogy olyan jellegű ágait is vizsgáljuk ennek a területnek, mint hogy hogyan képesek a jelenlegi fizikai eszközök és az SDN-ben használt eszközök közösen együtt működni. A vállalatok folyamatosan dolgoznak ennek a problémának a megoldásán és jelenleg is kérdéses az, hogy milyen módszerekkel lehet a leghatékonyabban megoldani a kommunikációt, illetve a jelenlegi rendszerek fejlesztését.

Megvizsgáltam milyen lehetőséget kínál az, amikor hibrid csomópontokat alakítunk ki, tehát felkészítjük az eszközöket arra, hogy képesek legyenek olyan SDN-ben használt protokollok használatára, mint az Openflow. Ezen a területen a hálózati eszközök gyártói sok erőfeszítést tesznek, ezáltal versenyképessé tehetik partnereiket, azonban nem feltétlenül lehet minden legacy típusú eszközt felkészíteni az SDN-es világban való együttműködésre. Erre kerestem megoldási lehetőséget a diplomatervezés során.

A munkám során kialakítottam többféle hálózati elrendezést is, amelynek célja az volt, hogy fokozatosan jussak el olyan szituációhoz, ami fizikailag összeköttetett hálózatot valósít meg. Kezdetben kipróbáltam azt a lehetőséget, hogy milyen, ha egy számítógépen viszonylag kevés felhasználói energiabefektetéssel megvalósítja a teszthálózatot a Mininet. Következő lépésként virtuális gépek egymásba ágyazásával és namespace-ek használatával alakítottam ki egy egyszerűbb hálózati elrendezést. Ezután olyan módszert próbáltam ki, amikor minden egyes hálózatban szereplő eszköz egy saját virtuális gépen fut és a köztük lévő kapcsolat már nagyon közel áll a fizikai összeköttetéshez. Utolsó lépésként a hálózat ESXi-re való áthelyezésére adtam ajánlást. Ennek használatával a virtuális eszközök teljesítménye a valós eszközök teljesítményét emulálhatják és az esetleges mérésekből jobb következtetéseket lehet levonni. Több olyan probléma is lassította a feladat elvégzését, amely nem ténylegesen a hálózati tudáshoz kapcsolódott, ezért szisztematikusan megvizsgáltam összetevőnként a hálózatot és igyekeztem megtalálni minden hibára a megoldást.

A hálózat kialakítása mellett, specifikáltam egy lehetséges kommunikációs megoldást. Az OSPF használatához szükséges információkat összegyűjtöttem, illetve utánajártam milyen módon lehet az Opendaylight esetén saját alkalmazást készíteni. A specifikációban leírtam, hogy milyen módon működne a megoldás, illetve milyen jellegű módosításokat kell implementálni a rendszerben.

A témakör kiterjedése miatt a jövőben több irányba is el lehet indulni, ha fejleszteni szeretnénk ezt a rendszert. Egyrészt a hálózati oldalról lehet még foglalkozni az implementációs kérdésekkel, illetve több különböző hálózatot is össze lehet rakni. A különböző hálózatokon ezután akár különböző méréseket is lehet végezni, amelyeket összehasonlíthatunk és következtetéseket vonhatunk le az eredményekből. Az általam adott specifikáció megvalósítása mellett, más hasonló ötleteket is lehet implementálni a hálózatban, majd ezeket összehasonlíthatjuk és akár különböző szempontok szerint értékelhetjük. Összességében véve ez a témakör még rengeteg lehetőséget nyújt és érdemes még többet foglalkozni vele a jövőben.

Ábrajegyzék

Irodalomjegyzék

[1] Open Networking Foundation: Software-Defined Networking (SDN)
https://www.opennetworking.org/sdn-definition/

[2] OpenFlow: Enabling Innovation in Campus Networks
http://archive.openflow.org/documents/openflow-wp-latest.pdf

[3] Stefano Salsano(1), Pier Luigi Ventre(2), Francesco Lombardo(1), Giuseppe Siracusano(1),Matteo Gerola(3), Elio Salvadori(3), Michele Santuari(3), Mauro Campanella(2), Luca Prete(4): *Hybrid IP/SDN networking: open implementation and experiment management tools*
http://netgroup.uniroma2.it/Stefano_Salsano/papers/salsano-oshi-mantoo.pdf

[4] Shie-Yuan Wang, Chia-Cheng Wu and Chih-Liang Chou :*Hybridtrace: A Traceroute Tool for Hybrid Networks Composed of SDN and Legacy Switches*
http://ieeexplore.ieee.org/document/7543773/

[5] János Farkas, Stephen Haddock, Panagiotis Saltsidis: *Software Defined Networking Supported by IEEE 802.1Q*
https://arxiv.org/ftp/arxiv/papers/1405/1405.6953.pdf

[6] Ryu hivatalos oldala
https://ryu.readthedocs.io/en/latest/

[7] Open Network Operating System
https://onosproject.org/

[8] Todd Fredrich: What is Rest?
http://www.restapitutorial.com/lessons/whatisrest.html

[9] Opendaylight wiki oldala
https://wiki.opendaylight.org/view/Main_Page

[10] Virtualbox
https://www.virtualbox.org/manual/ch06.html

[11] ESXi
https://www.vmware.com/products/esxi-and-esx.html

[12] Openvswitch
http://openvswitch.org/

[13] Openvswitch parancsok dokumentációja
http://www.pica8.com/document/v2.3/html/ovs-commands-reference/

[14] OSPF RFC, dokumentáció-
https://www.ietf.org/rfc/rfc2328.txt

[15] YANG RFC, dokumentáció
https://tools.ietf.org/html/rfc6020

[16] The TCP/IP Guide
http://www.tcpipguide.com