



Önálló laboratórium beszámoló

Távközlési és Médiainformatikai Tanszék

Készítette:	Kosztka Sándor
Neptun-kód:	J9RYP4
Ágazat:	Infokommunikációs hálózatok és szolgáltatások
E-mail cím:	sandor.kosztka@gmail.hu
Konzulens:	Megyesi Péter
E-mail címe:	megyesi@tmit.bme.hu

SDN hálózatok monitorozása különböző kontroller platformok fölött

Feladat

Feladatom a téma során, hogy megismerkedjek az SDN hálózatok monitorozásának lehetőségeivel, tehát, hogy hogyan lehet olyan QoS (Quality of Service) paramétereket mérni SDN hálózatokban, mint a felhasznált és az elérhető sávszélesség, a csomagvesztés vagy a késleltetés. Illetve az ötletek megvalósítása hálózat emulátorban.

2015/2016. 2. félév

1. A laboratóriumi munka környezetének ismertetése, a munka előzményei és kiindulási állapota

Bevezető/elméleti összefoglaló

Az elmúlt pár évben az ipar és a kutatók is nagy figyelmet szenteltek az úgynevezett Software Defined Networks (SDN) koncepciónak, mely a jövőben épülő hálózatok alapja lehet.

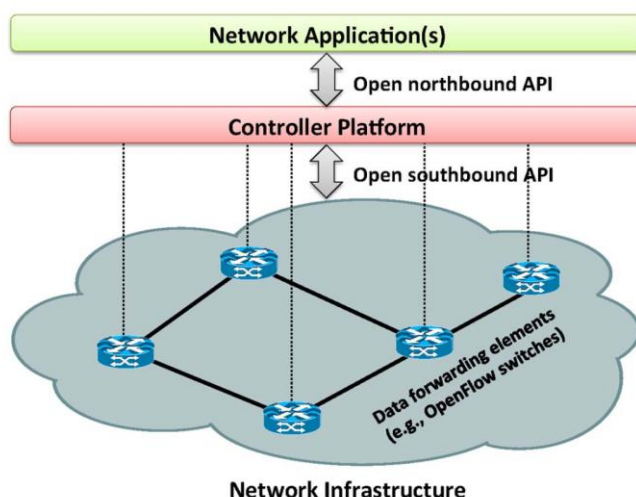
Az SDN hálózatok lényege, hogy ahelyett, hogy minden eszköz (egymással kommunikálva különböző protokollok segítségével, de) magától döntene a csomagok továbbítási iránya felől, csak továbbítják azokat. A hálózatban pedig elhelyezünk egy (esetleg több) kontrollert, mely az egész hálózatot átlátva dönt az ilyen szabályokról.

A koncepciót ma már sikeresen alkalmazzák a legnagyobb cloud szolgáltatók (pl.: Google, Facebook, Microsoft) a saját felhőikben is egyre nagyobb teret hódít, illetve a kisebb vállalatok által is használt private cloud szolgáltatások (pl.: OpenStack) is építenek rá, ám a teljes ipari váltásra még várni kell.

Témám során ezzel az új technológiával ismerkedtem meg, kezdve az elméleti alapoktól majd igyekeztem felderíteni, milyen rendszerfelügyeleti lehetőségeket kínál néhány kiválasztott controller, ezekkel pedig milyen QoS paraméterek mérése válik lehetővé. Ezt végigvezetem néhány egyszerű példán is és kitérek egy kidolgozott tervezetre is, ami nagyon jól illeszkedik az új hálózati koncepció architektúrájába.

Az SDN hálózatok alapjai:

Az eddig ismert, tradicionális hálózatokban az adattovábbításért és a kontroll/menedzsment rétegért felelős protokollok, szolgáltatások minden hálózati eszközben megtalálhatóak és az adott eszközök individuálisan döntenek az egyes csomagok továbbításáról. Ennek következményeképp egy új magas absztrakciós szintű igény megfogalmazódása esetén (pl. A tudjon kommunikálni B-vel) a hálózatüzemeltetőknek az egyes eszközökben meg kell változtatni azok konfigurációját alacsony szintű / gyártó specifikus nyelven.



1. ábra Egyszerűsített SDN architektúra

Az SDN ezzel szemben a kontroll logikát különválasztja a csomagtovábbító eszközöktől, így azok egyszerűbbek lesznek, csak az adatok továbbításáért felelnek. A logikai réteg pedig egy centralizált kontrollerbe kerül (nem feltétlen egyetlen kontrollerbe, gyártó függően ez kellőképpen skálázható), ami az egész hálózatot átlátva képes dönteni a csomagok továbbítási irányáról, módosításáról, esetleges eldobásáról.

Az 1. ábrán látható az architektúra egyszerűsített formája, a kontroller és az eszközök közötti kommunikációra szükségessé vált egy interfész, a Southbound API (Application Programming Interface), erre a legelterjedtebb jelenleg az OpenFlow. A kontroller és a különböző menedzsment alkalmazások közötti kommunikációra pedig a Northbound API. Ez utóbbit a későbbiekben fogom használni a monitoring alkalmazások teszteléséhez.

A munka állapota, készültségi foka a félév elején

A félévet gyakorlatilag tiszta lappal kezdtem, mindent a félév során készítettem. Konzulensemtől kaptam néhány példa scriptet a Mininet Python API használatának megismeréséhez, de ezeket mint a hivatalos dokumentáció kiegészítéseként használtam, nem emeltem át belőle semmit egy az egyben.

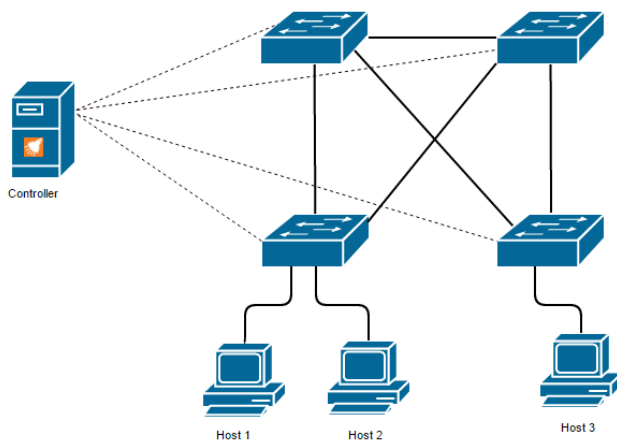
2. Az elvégzett munka és eredmények

A félév során először az SDN hálózatok elméleti hátterével kellett megismerkednem, ehhez nyújtott segítséget egy 2015. évi áttekintés[1], ami átfogó képet adott az új architektúráról, technológiáról és a használható szoftvekről, eszközökről is.

Ezek alapján kreáltam egy VPS (Virtual Private Server) a DigitalOcean nevű cloud szolgáltatónál, amire feltelepítettem a Mininet[4] nevű hálózati szimulátort és kiépítettem a 2. ábrán látható mérési / szimulációs környezetet.

A szerver publikus IP címe: 162.243.219.133

A szimulátor előre definiálva tartalmaz néhány hálózati topológiát, amelyeket használhatunk (pl.: fa, lineáris, tórusz, stb.), vagy egyedieket is definiálhatunk a Mininet Python API-n keresztül, ami ezen felül néhány automatikus tesztelésre is lehetőséget nyújt, például iperf vagy ditg használatával mérhetünk elérhető sáv szélességet a hálózaton belül, esetleg saját külső scripteket is futtathatunk a virtuális hosztokon.



2. ábra A mérési környezet

A switchek egyszerű mesh-be vannak kötve, ami az iparban egy elég elterjedt megoldás így ezt választottam de a Mininet beépített kontrollere egy egyszerű bridge/learning switch (OpenFlow reference controller) és nem támogatja a hurkok kezelését. Így már az elejétől fogva külső kontrollereket kellett használnom,

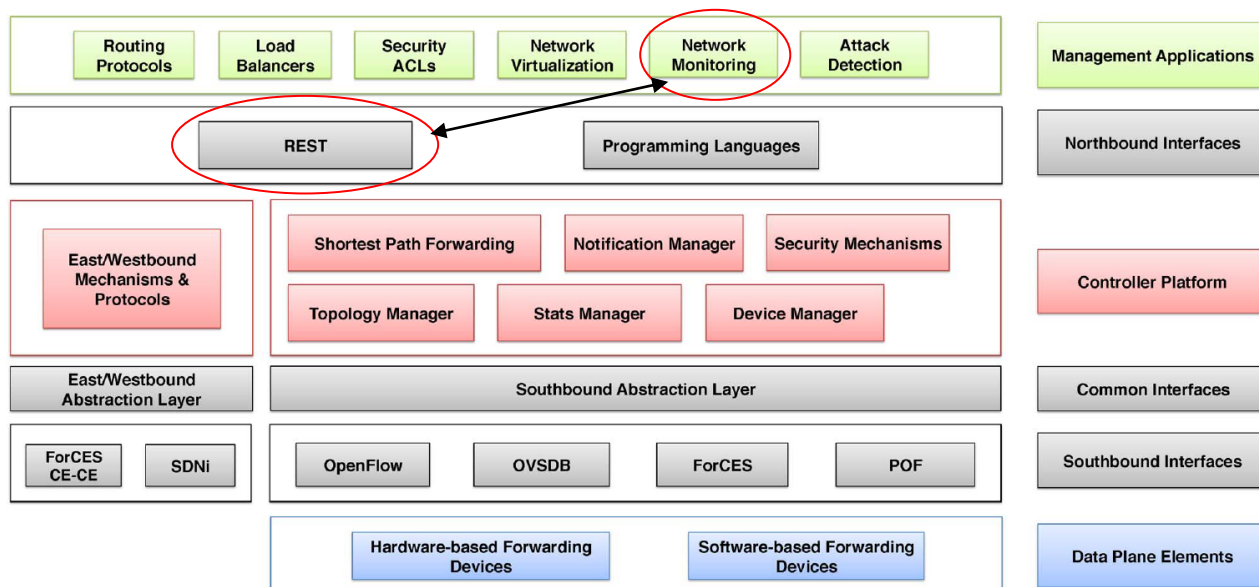
Az első controller a Mininet eszköztárában is megtalálható POX[4] volt, ebben már van Spanning Tree Protocol (STP)[7] implementálva így ellenőrizhetővé vált a hálózat működése.

Monitorozni egyszerűbb SDN hálózatok monitorozásához ideális lehet az OpenNetMon[3], ami nyílt forráskódú implementáció OpenFlow-t használó SDN hálózatok monitorozására. Lehetséges vele flow szintű metrikákat mérni, úgymint átviteli képesség, késleltetés és csomagvesztés.

Viszont eszemben nem működött feltételeztem, hogy a redundáns kapcsolatok miatt, de azokat megszüntetve sem tudtak a virtuális hosztok egymással kommunikálni, ha a teszt alatt futtattam az OpenNetMon is.

A második tesztelt controller a Floodlight[6]. Ezt a Mininet-et is futtató szerverre telepítettem, egyszerűen futtatható java alkalmazás. Érdekessége, hogy nem STP-t használ – röviden: nem tiltja le a redundáns linkeket hanem azokat engedi az eszközöknek párhuzamosan használni így azok több csomagot tudnak továbbítani adott idő alatt a több linken ezt nevezik Multipath-nak [4].

A Floodlight-ban már implementáltak Northbound interface így lehet erre a kapcsolatra alkalmazásokat építeni (3. ábra), például eszemben rendszerfelügyeletre.



3. ábra Northbound interface és alkalmazások

Ahogy a bemutatkozójukban is írják ez már egy enterprise-class, ipari alkalmazásra fejlesztett controller, már használható RESTful API-val (Representational State Transfer API) rendelkezik, ahol egyrészt le lehet kérni http kérések segítségével a controller állapotát, konfigurációt, mindezeket módosítani is.

Feladatom során a ezt az API-t próbáltam ki és kértem le statisztikát a kontrollerről. Az éppen aktuális verzióban is már elég sok mindent lehet lekérdezni, ha viszont nem lenne elég, szabadon bővíthető a controller API, ehhez dokumentációt is nyújtanak így ha még nincs olyan

funkció, amit mi szeretnénk, megvalósíthatjuk magunknak.

A lent leírt URL-en egy http GET kéréssel megkaphatjuk az összes a hálózatunkban lévő switch négyes portjának statisztikáit JSON (JavaScript Object Notation – nyílt szabvány, adatok kulcs-érték párokkal történő reprezentációjára) formátumban – a továbbított és fogadott adatmennyiségről.

URL: <http://162.243.219.133:8080/wm/statistics/bandwidth/all/4/json>

A kimenetként kapott JSON formátuma:

```
{
  "dpid": "00:00:00:00:00:00:00:02"
  "port": "4"
  "updated": "Wed May 04 15:37:44 EDT 2016"
  "bits-per-second-rx": "33"
  "bits-per-second-tx": "33"
}
```

Egyéb statisztikák is lekérdezhetők, ezeket közvetlenül az eszközökről kérdezi le a REST API-n keresztül így sok eszköz esetén nem célszerű mert, nagy overheadet jelenthet, de esetünkben még megteszi. Példaképp az összes switch aggregált statisztikája lekérdezésből az egyes switchre vonatkozó adatok:

URL: <http://162.243.219.133:8080/wm/core/switch/all/aggregate/json>

```
"00:00:00:00:00:00:00:01": {
  "aggregate": {
    "version": "OF_13"
    "flowCount": "1"
    "packetCount": "173"
    "byteCount": "13391"
    "flags": "0"
  }
}
```

Ezen felül lehetséges még port szintű statisztikákat is lekérdezni minden switchről, ezekben is releváns, felügyelet szempontjából használható adatok jelennek meg, mint ki és be irányú csomageldobások, kerethibák, ütközések stb.:

URL: <http://162.243.219.133:8080/wm/core/switch/all/port/json>

```
"00:00:00:00:00:00:00:03":
{
  "port_reply": [1]
  0: {
    "version": "OF_13"
    "port": [4] 0: {
      {
        "portNumber": "2"
        "receivePackets": "74"
        "transmitPackets": "116"
        "receiveBytes": "5759"
        "transmitBytes": "8970"
        "receiveDropped": "0"
        "transmitDropped": "0"
        "receiveErrors": "0"
        "transmitErrors": "0"
        "receiveFrameErrors": "0"
        "receiveOverrunErrors": "0"
        "receiveCRCErrors": "0"
        "collisions": "0"
        "durationSec": "362"
      }
    }
  }
}
```

```
"durationNsec": "532000000"
}}}
```

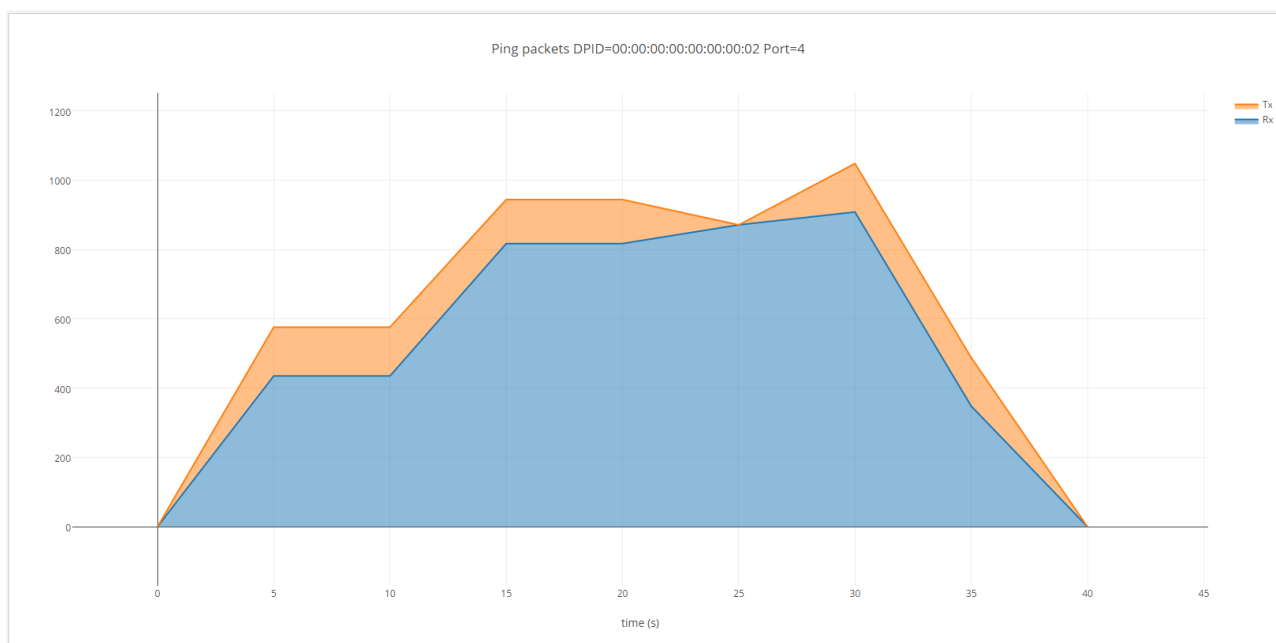
Ezeket a mai eszközökkel viszonylag egyszerűen feldolgozhatjuk, adatbázisban tárolhatjuk. Példaként egy port forgalmát monitoroztam, ahol két hoszt közötti ping forgalma zajlott - nagyjából egy percig.

A méréshez használt script:

```
while sleep 5; do curl
http://162.243.219.133:8080/wm/statistics/bandwidth/00:00:00:00:00:00:02/2/json
> portbw `date +%s`; done
```

Ebből kaptam 10db JSON fájlt, amit utólag feldolgozva grafikonon tudtam ábrázolni (plot.ly használatával [12]), de ehhez léteznek eszközök, amikkel mindezt valós időben is meg lehet tenni, sőt, ha az értékek bizonyos határokon kívül esik ezek is tudnak már riasztásokat indítani.

A 4. Ábrán látható a végeredményül kapott grafikon, egy port forgalmáról az időben, 5 másodpercenként kértem le az adatokat, de csak 10 másodpercenként frissült a statisztika, ami lekérdezhető az API-n keresztül.



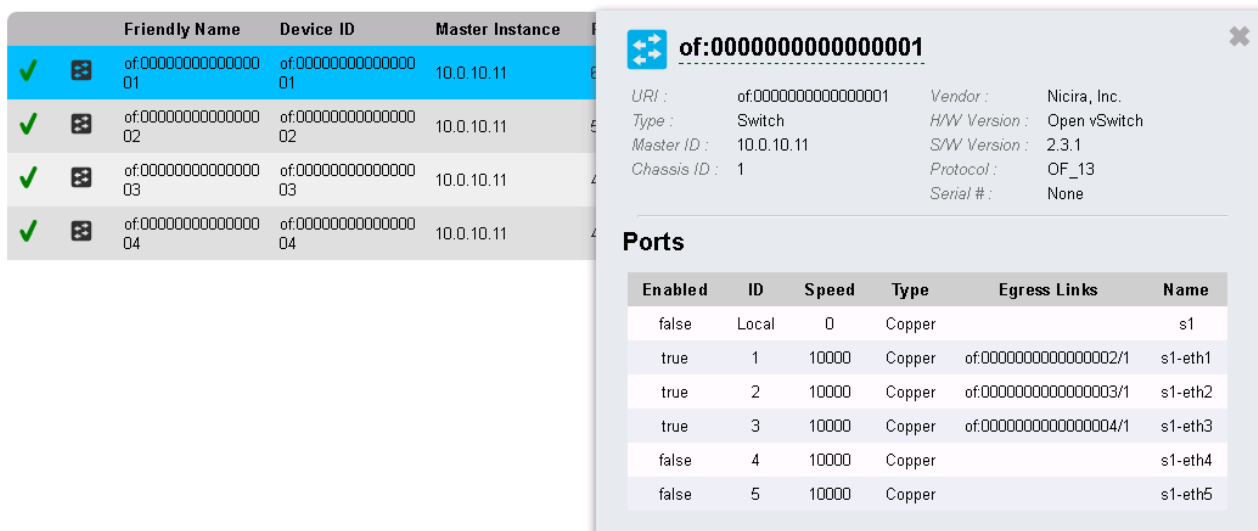
4. ábra Generált port statisztika

A harmadik controller az ONOS[7], ami talán a legnagyobb funkcionalitással bír a három közül. Elég fejlett webes interfésze van és rengeteg elemét támogatja az OpenFlow-nak. Ezt az SDN lehetőségeit kihasználva a hálózati szimulációs környezettől távol telepítettem egy másik, saját szerveremre, a fejlesztő projekt által kiadott hivatalos docker imaget használva.

Kezdeti nehézséget jelentett, hogy először nem tudtak a controllerhez csatlakozni a virtuális switchek, mert nem indult el magától az OpenFlow támogató alkalmazás a controlleren, ezt a webfelületen el is tudtuk indítani, ami után csatlakozni tudtak a switchek a 6633-as porton. Meg is jelentek a felismert topológiában és részletes információ is lekérdezhető az eszközökről (5. ábra), de a gépek egy mással még nem tudtak kommunikálni, ehhez engedélyezni kellett még egy alkalmazást (ez volt a reactive forwarding), hogy működjön itt is a Multipath csomagtovábbítás. Ez a controller már támogatja a legfrissebb OpenFlow verziót (1.3) amiben

megjelentek az úgynevezett meter-ek, ahol flow szinten lehet különböző, egyszerű QoS metrikákat mérni és ezeket kombinálni összetettebb framework-ökkel.

Devices (4 total)



The screenshot shows the ONOS web interface. On the left, there is a table listing 4 devices. On the right, a detailed view of a switch is shown.

Friendly Name	Device ID	Master Instance
of:0000000000000001	of:0000000000000001	10.0.10.11
of:0000000000000002	of:0000000000000002	10.0.10.11
of:0000000000000003	of:0000000000000003	10.0.10.11
of:0000000000000004	of:0000000000000004	10.0.10.11

Switch Details:

URI : of:0000000000000001 Vendor : Nicira, Inc.
 Type : Switch H/W Version : Open vSwitch
 Master ID : 10.0.10.11 S/W Version : 2.3.1
 Chassis ID : 1 Protocol : OF_13
 Serial # : None

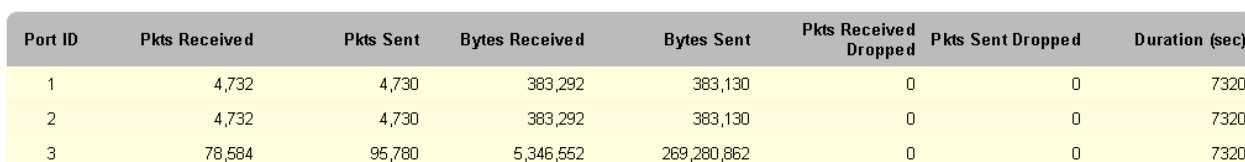
Ports

Enabled	ID	Speed	Type	Egress Links	Name
false	Local	0	Copper		s1
true	1	10000	Copper	of:0000000000000002/1	s1-eth1
true	2	10000	Copper	of:0000000000000003/1	s1-eth2
true	3	10000	Copper	of:0000000000000004/1	s1-eth3
false	4	10000	Copper		s1-eth4
false	5	10000	Copper		s1-eth5

5. ábra ONOS felülete, switch információk

Az eszközökről egyszerű statisztikák is lekérhetők a felhasználói felületen, mint az a 6. ábrán is látható.

Ports for Device of:0000000000000001 (3 Ports total)



Port ID	Pkts Received	Pkts Sent	Bytes Received	Bytes Sent	Pkts Received Dropped	Pkts Sent Dropped	Duration (sec)
1	4,732	4,730	383,292	383,130	0	0	7320
2	4,732	4,730	383,292	383,130	0	0	7320
3	78,584	95,780	5,346,552	269,280,862	0	0	7320

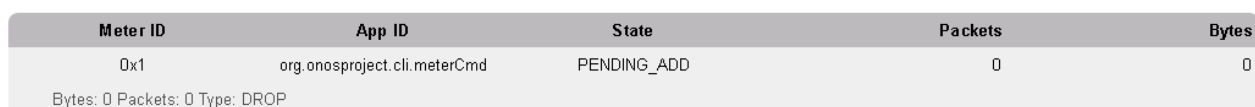
6. ábra Port információk

Példaként először a kontroller parancssoros felületén fel kellett venni egy adott switch-hez egy metert, ahol meg kell adni az switch azonosítóját (DPID: DataPath ID)

Vegyünk fel az egyesre végződő switch-re egy metert a parancssoros felületén (ez a futó konténer 8101-es portján érhető el SSH-n keresztül), ami utána a webfelületen is felkerült az eszköz információi közé (7. ábra)

```
onos:add-meter 0000000000000001
```

Meters for Device of:0000000000000001 (1 total)



Meter ID	App ID	State	Packets	Bytes
0x1	org.onosproject.cli.meterCmd	PENDING_ADD	0	0

Bytes: 0 Packets: 0 Type: DROP

7. ábra Eszközhöz rendelt meter

Ezután az ONOS REST API-ján lekérhetjük a rendszerünkhöz milyen meter-ek tartoznak és azok számlálói hogy állnak.

URL: `http://phoenix:8181/onos/v1/meters`

A válasz itt is JSON formátumban érkezik:

```
{ "meters": [ [1]0: {
  "id": "of:000000000000000001"
  "life": 0
  "packets": 0
  "bytes": 0
  "referenceCount": 0
  "unit": "KB_PER_SEC"
  "burst": false
  "deviceId": "1"
  "appId": "DefaultApplicationId{id=82, name=org.onosproject.cli.meterCmd}"
  "state": "PENDING_ADD"
  "bands":
  [1]0: {
    "type": "DROP"
    "rate": 500
    "packets": 0
    "bytes": 0
    "burstSize": null
  }
} ] ] }
```

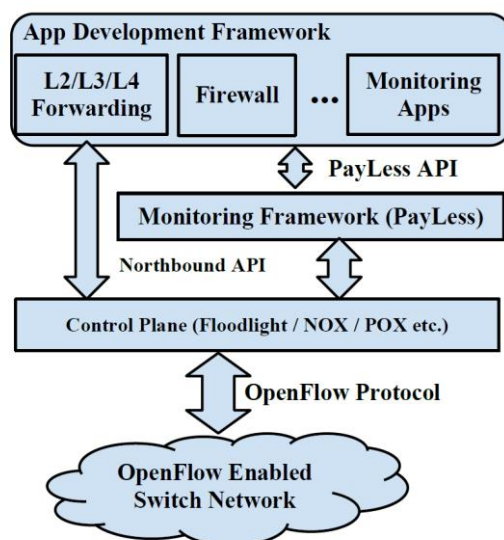
Összefoglalás

Eddig lehetőségünk volt aktív teszteket futtatva megkapni az elérhető sávszélességet például iperf használatával, de az foglalja is a sávszélességünket, kevesebb elérhető marad a hasznos adatoknak, ezért nem célszerű maximum hibakereséskor használhatjuk.

A kontrollerek REST API-ján keresztül már lekérhetők különböző metrikák, de ezek sem fedik le minden esetben teljes körűen azokat az igényeket, amik felmerülnek a QoS felügyeletéhez. A Floodlight megteremti a lehetőséget, hogy saját magunk fejlesszünk API modulokat és azt monitorozzuk, amit szeretnénk, de alapértelmezésben kevés az implementált modul. Az ONOS már támogatja a az 1.3-as OpenFlow meter-eit, amivel lehetőségünk adódik több mindent ellenőrizni, anélkül, hogy nekünk is fejleszteni kellene a kontrollert. De ez is még kevésnek tűnik, ahhoz képest, amire nekünk szükségünk van.

A PayLess[3] egy ígéretesnek tűnő monitoring keretrendszer, ami a kontrollerek Northbound interfészére épül (8. ábra). Egy saját, rugalmas RESTful API-t biztosít, ami elfedné a controller specifikus interfészt és rajta keresztül lehet flow szintű statisztikákat gyűjteni a hálózatunkból.

Sajnos még nem publikálták a szoftvert csak labor körülmények között tesztelték sikerrel a Floodlight controllerrel, de számomra meggyőzőek az eredményeik.



8. ábra A PayLess az SDN szoftver stackben

A rendszer magas szintű lekéréseket vár a különböző alkalmazásoktól, amiket feldolgoz, önmagában képes aggregálni és tárolni a statisztikai információkat így csak végeredményeket ad outputon a ráépülő szolgáltatásoknak.

```
{ "MonitoringRequest": {
    "Type": ["performance"],
    "Metrics": [
        { "performance": [
            "throughput",
            "latency",
            "jitter",
            "packet-drop",
        ] },
    ],
    "Entity": ["flow": "<flow_specification>"],
    "AggregationLevel": ["flow"],
    "Priority": ["real-time"],
    "Monitor": ["adaptive"],
    "Logging": ["default"]
  }
}
```

Ebben a félévben, az *Önálló laboratórium tárgy* keretében az alábbi új ismeretekre tettem szert, és ezeket készítettem tételesen:

- Elolvastam/megismerkedtem az SDN hálózatok alapjaival
- Készítettem 150 sor Python nyelven írt scriptet, ami felépíti a mérési környezetem
- Megtanultam kezelni a POX, Floodlight és ONOS SDN kontrollereket illetve ezek REST API-jait
- Feltérképeztem ezen kontrollerek monitorozási lehetőségeit
- Kipróbáltam miként lehet feldolgozni a kapott adatokat

3. Irodalom, és csatlakozó dokumentumok jegyzéke

A tanulmányozott irodalom jegyzéke:

- [1] D. Kreutz, *"Software-defined networking: A comprehensive survey"*, Proc. IEEE Vol. 103, No. 1, 2015
- [2] N.L.M. Van Adrichem, C. Doerr and F.A. Kuipers, *"OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks"*, Network Operations and Management Symposium (NOMS), 2014
- [3] S. R. Chowdhury *"PayLess: A low cost network monitoring framework for software defined networks"*, Proc. 14th IEEE/IFIP Netw. Oper. Manage. Symp., 2014
- [4] Multipath transport <https://tools.ietf.org/html/rfc6356>
- [5] POX kontroller, <http://www.noxrepo.org/pox/about-pox/>
- [6] Floodlight kontroller, <http://www.projectfloodlight.org/floodlight/>
- [7] ONOS kontroller, <http://onosproject.org/>
- [8] "Iperf: TCP/UDP Bandwidth Measurement Tool" <http://iperf.fr/>
- [9] "D-ITG, Distributed Internet Traffic Generator"
<http://traffic.comics.unina.it/software/ITG/>
- [10] "Mininet: An Instant Virtual Network on your Laptop (or other PC)" <http://mininet.org>
- [11] Spanning Tree Protocol <http://standards.ieee.org/getieee802/download/802.1D-2004.pdf>
- [12] "Plotly: state of the art in data viz, dashboards, and collaborative analysis" <https://plot.ly>