Gradus Vol 4, No 2 (2017) 453-460
ISSN 2064-8014

SDN PROGRAMOZÁS, A PYRETIC PROGRAMOZÁSI NYELV

SDN PROGRAMMING, THE PYRETIC PROGRAMMING LANGUAGE

Agg Péter András [1], Johanyák Zsolt Csaba [1], Halczman Szilvia [1]*

[1] Informatika Tanszék, GAMF Műszaki és Informatikai Kar, Neumann János Egyetem, Magyarország

Kulcsszavak:
SDN
Szoftver által definiált hálózat
SDN programozás
SDN programozási nyelvek
Pyretic

Keywords:
SDN,
Software-Defined Network
SDN programming
SDN programming languages
Networks

Cikktörténet:
Beérkezett: 2017. szeptember 25
Átdolgozva: 2017. október 8.
Elfogadva: 2017. október 18.

Összefoglalás

A szoftver által definiált hálózatok (SDN) egy meglehetősen új hálózati megoldás. Az SDN szétválasztja a vezérlő síkot és az adatsíkot. Három fő résszel rendelkezik: az adatsíkkal, a vezérlő síkkal és az alkalmazási síkkal. Az alkalmazási sík tartalmazza a programozási nyelveket. Ezen nyelvek használatával a felhasználó vezérelheti és módosíthatja a hálózat viselkedését. Az SDN az alacsony szintű programozási nyelvektől a magas szintűig számos lehetőséget biztosít számunkra.

A Pyretic magas szintű, nyílt forráskódú és Python alapú programozási nyelv. A Pyretic párhuzamos és egymás utáni operátorok használatával statikus és dinamikus továbbítási irányelveket határoz meg. Fő jellemzője a „ha - akkor" kapcsolat és a függvények használata.

Cikkünkben bemutatjuk általánosságban az SDN programozást, a Python és a Frenetic nyelvek jellemzőit. Ismertetjük a Pyretic nyelv szabályait és szerkezetét. Ismertetünk néhány programozási problémát és annak megoldását, mint például dinamikus, vagy lekérdezési szabályok létrehozása, módosítása.

Abstract

Software-Defined Networking (SDN) is a new networking approach that separates the control plane and the data plane, and it uses three layers, i.e. the data, the control and the application layer. The application layer contains the programming languages. Using SDN programming languages one can control and modify the behaviour of the network.

Pyretic is a high-level programming language that is open-source and Python based. Pyretic specifies static and dynamic forwarding policies by using parallel and sequential operators. Furthermore, it also uses if and return statements, variables and function definitions.

In this paper, first we introduce the SDN programming and the Python languages in general. Then, we show some policies and the structure of the Pyretic language. We also present typical problems and solutions, e.g. creating and modifying dynamic or query policies.

* Kapcsolattartó szerző. Tel.: +36-76-516-418
E-mail cím: agg.peter@gamf.kefo.hu

1. Bevezetés

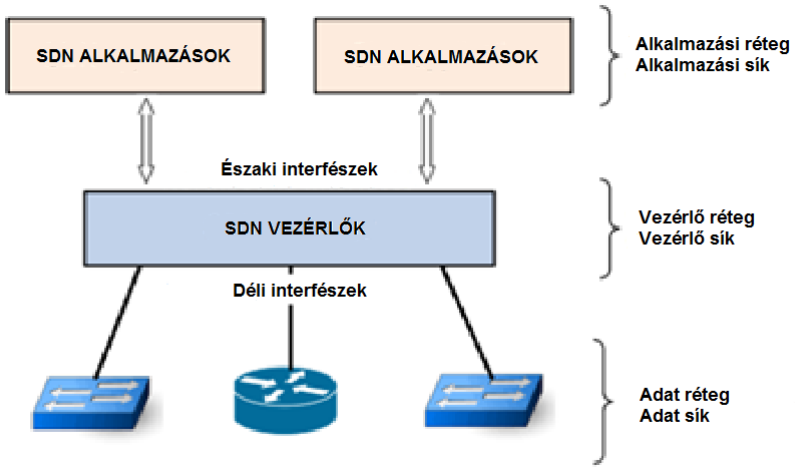
A szoftver által definiált hálózatok (Software-Defined Networks - SDN)[1][4] egyik nagy újítása volt, hogy amellett, hogy szétválasztotta a vezérlő és az adatsíkot, centralizálta a hálózati vezérlést is. A központosított vezérlés segítségével a hálózat irányítása sokkal hatékonyabbá és gyorsabbá válhat. A vezérlés, irányítás során különböző SDN alkalmazásokat használhatunk. Az SDN programozási nyelvek, melyek az SDN harmadik, legfelső rétegéhez (Alkalmazási réteg) tartoznak, teszik lehetővé a szoftverfejlesztők számára, hogy több szinten, más-más technikákkal fejlesszék ezen alkalmazásaikat. A programok által létrehozott szabályokat, utasításokat le kell fordítani egy olyan nyelvre, amelyet a hálózati eszközök érthetnek. A lefordított utasítások segítségével a programozási interfészeken (Application Programming Interface - API) keresztül automatizálhatóvá és programozhatóvá válik a hálózat. Az SDN-ben használt kapcsolók alacsony szintű interfészt használnak. A szerver oldali eszközök az API-k révén a hálózat funkcióihoz is hozzáférhetnek, ezáltal megvalósítható az újraprogramozás és a felhasználók igényeinek megfelelő beállítások.

Az SDN programozási nyelveket három csoportba sorolhatjuk, ezek az alacsony szintű, az API alapú, és a domain-specifikus nyelvek. Az alacsony szintű nyelvek kevésbé felhasználóbarátok, de nagyon hatékonyak és gyorsak. Az API alapú programozási nyelvek [7] közül a helyi API-k nagyon eszközfüggők, míg a távoli API-k már szabadabb lehetőségeket biztosítanak a fejlesztő számára. A domain-specifikus nyelvek használatakor a programozó számos módszerrel tudja létrehozni az új szabályokat, illetve lehetősége van a már meglevő szabályok dinamikus módosítására is.

Cikkünk második és harmadik szakaszában bemutatjuk röviden az SDN rétegeit, majd összefoglaljuk a három szint jellemzőit. A negyedik és ötödik szakaszban bemutatjuk a Frenetic [9] és a Python [12] nyelv fő ismérveit, melyek felhasználásával kialakult a Pyretic magasszintű SDN programozási nyelv. Végül a hatodik szakaszban a Pyretic nyelv jellemzőit, alap, lekérdezési, és dinamikus szabályait vázoljuk fel példák segítségével. A következtetések és az összegzés a hetedik szakaszban találhatóak.

2. Szoftver által definiált hálózatok

A szoftver által definiált hálózat egyik legfontosabb jellemzője, hogy szétválasztja a vezérlési síkot (control plane) és az adatsíkot (data plane) (eszközsíkot) az eszközökben [4]. Az SDN nyitott architektúrán alapul, ami a hálózat konfigurálását és menedzselését rugalmassá és programozhatóvá teszi. Az SDN architektúrának három fő rétege van (1. ábra), az adat, a vezérlő és az alkalmazási réteg, melyeknek további alrétegei is vannak. A továbbiakban röviden áttekintjük a három sík jellemzőit.



1. ábra. SDN tervezési architektúra [1]

2.1. Adatsík

Az adatsík biztosítja a hálózati csomagok mozgatását az egyes hálózati szegmensek vagy végpontok között. Az adatsík két fő része a hálózati infrastruktúra (Network Infrastructure) és az ún. déli interfész (Southbound interface). A hálózati infrastruktúra gyakorlatilag megegyezik a korábbi hálózati eszközökkel, azzal a megkötéssel, hogy a hardverek képesek kell legyenek kommunikálni a felsőbb rétegekkel. A déli interfésze biztosítja a kapcsolatot az adat és a vezérlő sík között, kezeli azokat az információkat, amelyek fontosak a hálózati operációs rendszerek számára.

2.2. Vezérlő sík

A vezérlő sík egyik legfontosabb eleme a virtualizáció. Az itt használt programok megszüntetik az időigényes műveletek használatát, biztosítják az automatikus hálózati konfigurálást, és lehetővé teszik a dinamikus hozzáférést és az adminisztrációt. Egy központi vezérlő konzolon keresztül anélkül alakítható a hálózati forgalom, hogy közvetlenül konfigurálnánk az adott eszközt. Az ebben a rétegben működő hálózati operációs rendszer (Network Operating System) biztosítja a hálózati menedzsment problémáinak megoldását a központosított vezérlés terén. A vezérlő réteg és az alkalmazási réteg közötti kommunikációt az északi interfész biztosítja.

2.3. Alkalmazási sík

A harmadik rétegben a menedzsment eszközök helyezkednek el. Feladatuk a pontos utasítások kiadása a vezérlés felé. Három alrétege a nyelv alapú virtualizáció (Language-based virtualization)[1], a programozási nyelvek (Programming language) és a hálózati alkalmazások (Network Applications). SDN programozási nyelveket használva a probléma-orientáltságra fókuszálva biztosíthatjuk az adatok áramlásának megbízhatóságát, és felgyorsíthatjuk az adatkommunikációt központi felügyelet mellett.

A szoftver-definiált hálózatkezelés támogatja a központosított programozási modellt a számítógépes hálózatok kezelésére. Az SDN programozási nyelvek lehetővé teszik a programozó számára, hogy egyszerű programozási konstrukciókat használjon központi centralizált algoritmikus szabályok deklarálására, amelyek felelősek egy teljes hálózat viselkedéséért. Számos SDN programozási nyelv létezik, amelyek különböző funkciókat használnak, és több hálózati probléma megoldására összpontosítanak.

3. SDN PROGRAMOZÁS

A hálózatba érkező csomagok továbbítása a csomagfejléc mezők illeszkedése alapján történhet. Ilyenkor vizsgálhatjuk akár a csomag IP-címét vagy akár MAC-címét is. Ha sikerült beazonosítani a célt, akkor a csomag továbbításra kerül, ellenkező esetben mindenképp a vezérlőn keresztül kell megoldanunk a továbbítást. A vezérlő az elvégzendő feladatoktól függően módosítja kapcsoló tábláit a létező alkalmazások segítségével. Ezen alkalmazások három különböző szinten programozhatók:

- alacsony szinten,
- API-alapon [7]
- domain-specifikus nyelv segítségével [6][8]

3.1. Alacsony szintű programozás

Az alacsony szintű programozást a hálózati programozás gépi kódú nyelvének is tekinthetjük, ahol a végrehajtás a déli interfészen keresztül történik. Ezeknél a programozási nyelveknél a Control-Data-Plane Interface-n (CDPI) keresztül vezérlik a hálózatot. Az adminisztrátornak az eszközöket egyenként kell programoznia, és a hibákat is manuálisan kell kezelni. A CDPI üzenetek cseréjére ezen a programozási szinten egy általános célú nyelv (GPL) használata szükséges (pl. C / C ++, Python vagy shell parancsfájl).

3.2. API-alapú programozás

API-alapú programozási alkalmazások esetén egy osztály/függvény gyűjtemény (API) függvényeinek meghívásával állítjuk elő alkalmazásunkat. Az így elkészült programokat a vezérlő szintén CDPI üzenetekbe fordítja. Két API-típust különböztetünk meg: helyit vagy a távolit. A helyi API-t használó SDN-alkalmazásoknál a fejlesztő általában a vezérlő saját nyelvén írja le az utasításokat, és csak az adott gépen (vezérlőn) futtatható a program. Ezzel szemben a távoli API-k lehetővé teszik, hogy a különböző technológiákkal készült alkalmazások egymás között információt cseréljenek, például Extensible Markup Language (XML) segítségével. Ezért a távoli API-t használó alkalmazások bármely GPL-ben megírhatók, és eltérő gépeken is futtathatók. A távoli API-k forráskódja tisztább, mint a helyi API alapúaké [8].

3.3. A Domain-specifikus nyelv

A Domain-specifikus nyelv [6][8] (DSL) egy olyan programozási nyelv, amely általában csak egy adott problématartomány megoldására korlátozódik megfelelő kifejezések és absztrakciók révén. A DSL-eknek van egy fordítója (vagy keretrendszere), amely biztosítja a magas szintű utasítások olyan nyelvekre történő fordítását, amelyeket a vezérlő API-ja értelmezni tud. A DSL-k magas szintű fejlesztési lehetőségeket biztosítanak új absztrakciók hozzáadásával a különböző programozási megoldások, például a felügyelet, a biztonság és a virtualizáció számára. Segítenek továbbá a hálózati erőforrás-kihasználás optimalizálásában.

4. Frenetic nyelv főbb jellemzői

A Frenetic [9][10][14] volt az egyik első SDN programozási nyelv, a meglévő SDN programnyelvek többsége a Frenetic leszármazottja. Funkcionális könyvtárakat tartalmaz a magas szintű csomagküldési szabályok leírásához, illetve használható benne egy SQL-szerű lekérdező nyelv is, mellyel szükség szerint osztályozható és lekérdezhető a hálózati forgalom. Lehetővé teszi a moduláris programozást és a kód újrafelhasználását is A lekérdező nyelv használata lehetőséget biztosít az információk csoportosításához csomagok száma, csomagméret vagy idő szerint, melyek akár jelentésekben is rögzíthetők. Az alábbi példa azt mutatja meg, hogy hogyan lehet 30 másodpercenként az egyes számú fizikai porton keresztül beérkezett csomagok méretét összegezni a cél MAC cím alapján csoportosítva.

```
def host_query():
return (Select(sizes) *
        Where(inport_fp(1)) *
        GroupBy([dstmac]) *
        Every(30))
```

5. Python

A Python [12][13] egy nyílt forráskódú, bővíthető programozási nyelv, amely segítséget nyújt a programozás moduláris és objektum orientált megközelítéséhez, ami fontos kapcsolatot biztosít a Frenetic nyelvvel. A Python nyelvet Guido van Rossum kezdte el fejleszteni. Sokféle platformon használható, egyaránt megfelel rövid és hosszú forráskódú programoknak is. Szintaxisa egyszerű, tömör, jól olvasható alkalmazások írhatók vele. Hasonló feladatot ellátó C vagy C++ programok hosszához képest jóval rövidebb, ami által jelentősen lecsökken a fejlesztési idő, és a karbantartás is egyszerűbb. Beavatkozás nélkül kezelhetők az erőforrások egy hivatkozás számláló mechanizmus segítségével. A Pythonra jellemző a dinamikus változó és típuskezelés, minden objektumnak meghatározott típusa van, amit nem kell előre definiálni. A Python alap standard könyvtárak használatát és kiegészítő csomagok használatát is lehetővé teszi. A külső modulok beillesztése egyszerű, megbízható működésű. A Python XML kezelésére is alkalmas nyelv. Két implementációja használható napjainkban. Az egyik közvetlenül Python bytecode-t generál, míg a másiknál a programok portabilis utasításokra vannak lefordítva, melyeket egy virtuális gép hajt végre.

6. Pyretic = Python + Frenetic

A Pyretic [11][15] az SDN programozási nyelvek Frenetic [10][14] családjának tagja. A Pyretic lehetővé teszi a hálózati programozók és az operátorok számára, hogy tömör moduláris hálózati alkalmazásokat készítsenek hatékony absztrakciókkal. A Pyretic egy programozó barát domain-specifikus nyelv, amely Python-ba van ágyazva, és olyan futásidejű rendszert biztosít, amely a Pyretic nyelven írt programokat valósítja meg a hálózati kapcsolókon. Statikus és dinamikus továbbítási szabályokat határoz meg. Különböző szabályokat (policy) alkalmaz párhuzamos és egymást követő operátorokkal. A Pyretic *if* és *return* nyelvi elemeket használ, valamint konkrét változó és függvény definíciókat. A Pyretic alapvető „könyvtárat” használ (beépített függvények) a hálózati topológia absztrakcióhoz, amely a virtualizációban használható. A virtualizáció irányelvei más szabályokkal együtt használhatók, vagy egy másik virtualizációs réteg alapjaként szolgálhatnak.

6.1. Pyretic futtatás

Ahhoz, hogy egy Pyretic programot futtassunk egy virtuális környezetet hasznos létrehoznunk, ha tesztelni szeretnénk. Például egy Mininet VM-et. A futatáshoz a következő parancsot kell kiadnunk parancssorból:

$ pyretic.py

Ehhez a parancshoz három féle kapcsolót használhatunk.

–m kapcsoló
interpreted (i)
reactive (r0)
proactive (p0)

A -m kapcsoló a Pyretic futásidejű üzemmódját jelöli. Értelmező mód (*interpreted*): minden csomag feldolgozása a vezérlő futásidejében történik. Természetesen lassú, de nagyon hasznos módszer a hibakereséshez. Reaktív mód (reactive): ez a módszer is viszonylag lassú, hisz ha egy olyan csomag éri el a kapcsolót, melyet nem tud továbbküldeni, akkor ezt először a vezérlőhöz kell továbbítania, majd az ott kapott utasítások (szabályok) alapján próbálja meg a megfelelő címzetthez elküldeni. Ez a módszer mindig az aktuális forgalmat tükrözi. Proaktív mód (proactive): általában ez a jelenleg elérhető legmagasabb teljesítményű mód. Az ismert szabályokat már előre a kapcsolókra küldik, így nem szükséges a csomag érkezéskor kommunikálni a vezérlővel. Hátránya, hogy nem tükrözheti az aktuális forgalom állapotát. Egy konkrét futtatási példa:

$ pyretic.py -m p0 pyretic.modules.mac_learner

6.2. Pyretic főfüggvény

Hasonlóan a legtöbb programozási nyelvhez a Pyretic is igényel egy főfüggvényt (*main*()). A program zavartalan működéséhez először be kell tölteni a szükséges alap „könyvtárakat”, ezek a Pyretic core library (minimum) és a Pyretic query library. Betöltésük megoldása az alábbi:

```
from pyretic.lib.corelib import *
from pyretic.lib.query import *
```

A főfüggvény tartalmaz egy visszatérési értéket, ami a Pyretic nyelv valamelyik osztálya lesz. Például:

```
def main():
    return mac_learner()
```

6.3. Pyretic alapszabályok (Basic Policies)

A Pyretic programozási nyelv számos úgynevezett alapszabályt használ (1. táblázat). Az itt található alapszabályok önállóan és különböző operátorok segítségével összekapcsolva is működhetnek, így újabb és újabb szabályok hozhatók létre, akár dinamikusan is.

1. Táblázat: Pyretic alapszabályok

Szabály	*Szintaxis*	*Példa*
match	*match(f=v)*	*match(dstmac=EthAddr('00:00:00:00:00:01'))*
drop	*drop*	*drop*
identity	*identity*	*identity*
modify	*modify(f=v)*	*modify(srcmac=EthAddr('00:00:00:00:00:01'))*
forward	*fwd(a)*	*fwd(1)*
flood	*flood()*	*flood()*
parallel	*A + B*	*fwd(1) + fwd(2)*
sequential	*A >> B*	*modify(dstip=IPAddr('10.0.0.2')) >> fwd(2)*
negation	*~A*	*~match(switch=1)*

A táblázatban látható *drop* utasítást használva eldobhatjuk a csomagot, és üres értékkel térhetünk vissza. A *match* ellenőrzi, hogy az *f=v*, és visszatér a megfelelő logikai értékkel. A *modify* szabály a megfelelő értékre módosítja például a forrás fizikai címét, míg az *identity* a csomag másolatát adja vissza. A *forward* továbbítja a csomagot „a”-ra. A *flood* szabály a csomagot a feszítőfa protokoll által meghatározott portokra továbbítja. Operátok használatával párhuzamos és szekvenciális megoldásokat is alkalmazhatunk. A fenti példában (*A+B*) a visszatérés a kettő uniója lesz. Szekvenciális esetben *A* kimenete lesz *B* bemenete.
Fontos megjegyezni, hogy a Pyretic programozási nyelv – logikai predikátumokat használ, mint például *AND*, *OR*, *NOT*. Használ úgynevezett virtuális csomag fejléc mezőket is. Ebben a fejlécben a csomag metaadatainak megjelenítése egységes (pl. a belépési port), módosítására a *mod*() parancsot használhatjuk.

6.4. Lekérdezési szabályok (Query Policies)

A hagyományos OpenFlow [2][3][5] programokban a forgalmi statisztikák gyűjtésére alapszabályokat használnak, majd lekérdezéseket hoznak létre a számlálók kezelésére, a válaszok értelmezésére. Amint ezek az adatok beérkeztek, újabb szabályok használata következik ezek feldolgozására. A Pyretic programozás során használhatóak úgynevezett hálózati forgalom monitor szabályok (network traffic monitoring policies) amelyek összekapcsolhatók más szabályokkal, így nem kell több lépcsőben elvégezni a statisztikai kiértékelést. Az alábbi példák azt mutatják meg, hogyan lehet lekérdezni egyszerű SQL parancsokkal a szükséges statisztikákat. Megszámlálunk minden csomagot, az adott intervallumban (az adott időnek megfelelően, pl. ha *t*=60, akkor percenként), és csoportosítjuk f_1-nek, f_2-nek stb. megfelelően.

count_packets(interval=t, group_by=[f_1,f_2,...])

Megszámlálunk minden beérkezett byte-ot, az adott intervallumban (az adott időnek megfelelően, pl. ha t=60, akkor percenként), és csoportosítjuk f_1-nek, f_2-nek stb. megfelelően.

count_bytes(interval=t, group_by=[f_1,f_2,...])

6.5. Dinamikus szabályok (Dynamic Policies)

Olyan szabályokat nevezünk dinamikus szabályoknak, melyek viselkedése megváltozhat a programozó igényeinek megfelelően (pl. mikor továbbítson egy csomagot). Az aktuális értéket mindig a *self.policy*() tartalmazza. Ha változtatunk a szabályokon, akkor a *self.policy*()-t frissítenünk kell. Ezek a dinamikus szabályok készülhetnek alapszabályok megváltoztatásával, vagy akár alapszabályok párhuzamos vagy szekvenciális kapcsolásával is.

A következő példaprogram azt mutatja meg, hogy hogyan tudunk egy tűzfalszabályt egyszerűen, dinamikusan módosítani. Első lépésként a programrész ellenőrzi, hogy szerepel-e a *mac_1* és *mac_2* fizikai címpár a tűzfalszabályban. Ha igen, a visszatérő logikai érték igaz lesz, így egyszerűen tájékoztat arról, hogy ez a bejegyzés már létezik. Ellenkező esetben a logikai értéket igazra állítja, ezzel hozzáadja az új bejegyzést a tűzfalhoz, majd kiírja, hogy mit végzett el, és frissíti a szükséges szabályt (*self.update policy())*

```
def AddRule (self, mac_1, mac_2):
   if (mac_1, mac_2) in self.firewall:
     print “The rule already exists”
     return
   self.firewall[(mac_1, mac_2)] = True
   print “Adding new rule in %s : %s” % (mac_1, mac_2)
   self.update policy()
```

7. Következtetések

SDN alkalmazások készítéséhez számos programozási nyelv használható. Ezek közül a magas szintű Domain-specifikus nyelvek a legelterjedtebbek. A Pyretic programozási nyelv is ebbe a csoportba tartozik. Használata arra ösztönzi a programozókat, hogy a hálózati szabályokat magas szintű absztrakcióval határozzák meg, az alacsony szintű OpenFlow mechanizmusok helyett. A hálózati igényeknek megfelelően a programozó a Pyretic nyelv alapszabályait kombinálva (párhuzamos, vagy szekvenciális módon) dinamikus szabályokat hozhat létre, amivel jelentősen meggyorsíthatja a hálózat működését.

A Pyreticben elkészített kód jóval rövidebb, mint a nyelv elődjeiben megírt kódsorozat, elsősorban annak köszönhetően, hogy a Pyretic deklaratív, erőteljes, mégis tömör szabályalkotó nyelv. Az általunk bemutatott példákban jól látható, hogy milyen egyszerűen lehet lekérdezési szabályokat létrehozni, amelyeket aztán akár a hálózati forgalom változásának függvényében dinamikusan átalakíthatunk. A felsorolt előnyök hatására vált a Pyretic nyelv az egyik legelterjedtebb SDN nyelvvé. Az SDN programozási nyelvek nagyon gyorsan fejlődnek, bővülnek, ezért biztos, hogy a jövőben újabb és újabb nyelvekkel fogunk találkozni.

8. Köszönetnyilvánítás

Köszönettel tartozunk a kutatás támogatásáért, amely az **EFOP-3.6.1-16-2016-00006 „A kutatási potenciál fejlesztése és bővítése a Neumann János Egyetemen” pályázat keretében valósult meg.** A projekt a Magyar Állam és az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával, a Széchenyi 2020 program keretében valósul meg.

9. Irodalomjegyzék

[1] Casado, N. Foster, and A. Guha, “Abstractions for software-defined networks,” Commun. ACM, vol. 57, no. 10, pp. 86–95, Sep. 2014.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: enabling innovation in campus networks,” SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, Mar. 2008.

[3] OpenFlow Switch Specification http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf [Megtekintés: 2017. 09. 04.]
[4] Fernando M. V. Ramos, D. Kreutz, Paulo Verissimo, “Software-defined networks: On the road to the softwarization of networking” Agile Product Management & Software Engineering Excellence, Business Technology & Digital Transformation Strategies Cutter Business Technology Journal, vol. 28., pp. 6-13., May. 2015
[5] A. Lara, A. Kolasani, and B. Ramamurthy, “Network innovation using OpenFlow: A survey,” IEEE Communications Surveys & Tutorials, 16(1), pp. 493-512. 2014.
[6] A. Van Deursen, P. Klint, and J. Visser, “Domain-specific languages: An annotated bibliography.” Sigplan Notices, vol. 35, no. 6, pp. 26–36. , 2000.
[7] Network Application Programming Interfaces (APIs) http://compnetworking. about.com/od/softwareapplicationstools/g/bldef_api.htm [Megtekintés: 2017. 08. 27.]
[8] M. Mernik, J. Heering, and A. M. Sloane, “When and how to develop domain-specific languages,” ACM computing surveys (CSUR), vol. 37, no. 4, pp. 316–344, 2005.
[9] N. Foster, M. J. Freedman, R. Harrison, J. Rexford, M. L. Meola, and D. Walker, “Frenetic: a high-level language for OpenFlow networks,” in Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow. ACMA, p. 6., 2010.
[10] N. Foster, A. Guha, M. Reitblatt, A. Story, M. J. Freedman, N. P. Katta, C.Monsanto, J.Reich, J.Rexford, C.Schlesinger et al., “Languages for software-defined networks,” Communications Magazine, IEEE, vol. 51, no. 2, pp. 128–134., 2013.
[11] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, “Modular SDN programming with Pyretic,” Technical Report of USENIX, 2013.
[12] Harvey M. Deitel, et al, Prentice Hall; “Python How to program” 1292 pages, ISBN: 0130923613, February 4, 2002.
[13] John E. Grayson, ”Python and Tkinter Programming” Manning Publications; 1st edition, 688 pages, ISBN: 1884777813, January, 2000.
[14] Pyretic http://frenetic-lang.org/pyretic/ [Megtekintés: 2017. 09. 15.]
[15] Christopher Monsanto, Joshua Reich, Nate Foster†, Jennifer Rexford, David Walker: Composing Software-Defined Networks, 10th USENIX Symposium on Networked Systems Design and Implementation, Proceedings of NSDI ’13, pp. 1-15., 2013.