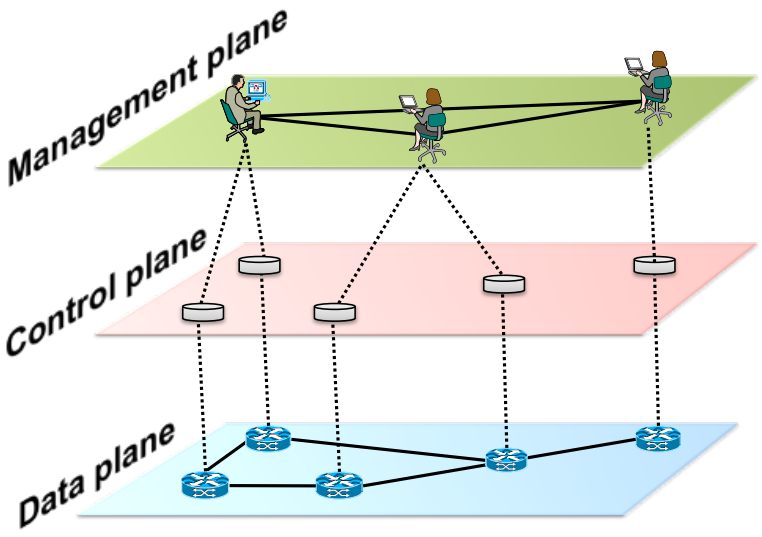
1. fejezet

A hálózat mint platform

> „Think of it as a general language or an instruction set that lets me write a control program for the network rather than having to rewrite all of code on each individual router."
>
> — Scott Shenker, Berkeley

Kivonat – A 2000-es évek vége óta a számítógéphálózatok világában vírusszerűen terjed a szoftveresen vezérelhető hálózatok (Software Defined Networking SDN) paradigmája, ami a szoftvervilágra jellemző architekturális elemekre épül. Az SDN-ben a hálózat kapcsolóit és összeköttetéseit együttesen tekintjük a hardvernek, melyekhez hálózati operációs rendszeren (Network Operating System NOS) keresztül lehet hozzáférni. A operációs rendszer felett alkalmazások futtathatók, amelyek a hálózat funkcionalitását a kapcsolóeszközök programozásán keresztül megadják. Hasonló ez ahhoz, hogy az okostelefon hardverén fut az operációs rendszer (IOS, Android stb.), ami felett alkalmazások futtathatók. Az SDN hálózatok villámgyors előretörése és terjedése azt is magával hozta, hogy ezek ismerete nem hiányozhat egy magára valamit is adó B.Sc.-s informatikus kelléktárából.

Mielőtt az SDN tárgyalására rátérnénk idézzük fel a hagyományos hálózati architektúra néhány jellemvonását, hogy aztán minél jobban érezhessük az SDN paradigmaváltás újszerűségét.



1.1. ábra. A hálózatok síkjai

1.1. A hagyományos hálózati architektúra

A számítógép hálózatok három jól elkülöníthető funkcionalitású síkra oszthatók, az *adat*, *vezérlő* és *menedzsment* síkok (ábra). Az adatsík gyakorlatilag a hálózati (kapcsoló) eszközöket tartalmazza, amelyek feladata nem más, mint a csomagok hatékony továbbítása. A vezérlő sík különböző protokolljai oldják meg, hogy a kapcsolóeszközökben levő kapcsolási táblázatok megfelelően kitöltődjenek, míg a menedzsment síkban lehet megadni és nyomon követni a kívánt hálózati funkcionalitást. Tehát a menedzsment sík definiálja, a vezérlő sík kikényszeríti, az adatsík pedig végrehajtja a kívánt működést. A konkrét példa kedvéért vegyük azt, hogy a hálózatunkon legrövidebb útválasztást szeretnénk megvalósítani. Ezzel a menedzsment síkban annyi a teendőnk, hogy minden eszközön OSPF[1] (Open Shortest Path First) modulokat indítunk el. Az OSPF mint vezérlési protokoll elosztott módon kiszámolja a legrövidebb utakat és beállítja az útválasztási táblákat az eszközökben, amelyek végül az adatsíkon ennek megfelelően irányítják a csomagokat.

A hagyományos IP hálózatokban a vezérlő és az adat síkok szinte minden esetben egy fizikai eszközben kerülnek megvalósításra (tehát a fenti példában az OSPF protokoll a routerben van megvalósítva) és az architektúra nagy mértékben elosztott (vagyis az OSPF egy-egy példánya minden routeren fut). Ennek oka elsősorban az, hogy a számítógéphálózatok korai szakaszában a hibatűrés elsődleges szempont volt, és extrém jó hibatűrést elosztott viselkedéssel lehet megvalósítani.

[1] Mi az OSPF

A hagyományos rendszerek, amennyire hibatűrőek, annyira bonyolultak és statikusak. A bonyolultságot a fenti OSPF-es példán illusztrálhatjuk. Ahhoz, hogy a világ legegyszerűbb útválasztási algoritmusát implementáljuk egy teljes elosztott protokollt kell tervezni. Természetesen a tervezés nagy része nem magának az útválasztásnak az implementálására megy (ami kb. öt perc), hanem arra, hogy az elosztott hálózati protokoll valami egységes viselkedés felé konvergáljon. A statikusság egész egyszerűen az, hogy gyakorlatilag lehetetlen a gyors innováció egy hagyományos hálózatban, hiszen minden egyes új szolgáltatáshoz, egy új elosztott protokollt kell megvalósítani és a routerekbe feltölteni. Ráadásul mivel a vezérlő és az adatsík is a kapcsolóeszközökben van implementálva, gyakorlatilag az eszközgyártók határozzák meg, hogy mire lehet egy hálózatot egyáltalán használni.

Ezen felül a hálózat menedzsmentje is elképesztően problémás. Ezt jól illusztrálja az, hogy a hibás hálózati beállításokból eredő helytelen működés ez egyik legelterjedtebb hibajelenség ma a hálózatok körében. Egy rosszul beállított eszköz természetesen hat a környezetére is, ami sokféle nehezen felderíthető hibajelenséget produkálhat (pl. csomagvesztés, továbbítási hurkok, hibás útvonalak). Bár szerencsére ritka, de egyetlen rosszul beállított router órákig megzavarhatja a teljes Internet viselkedését.

Jelenleg a hálózati eszközök menedzsmentjére minden nagyobb gyártónak (CISCO, Juniper stb.) saját zárt megoldása van egy adott hardver-szoftver konfigurációval rendelkező eszközre. Magyarul, egy heterogén eszközökből felépített hálózati infrastruktúra menedzsmentjéhez speciális szakértő csapatok kellenek, amelyek az egyes gyártók menedzsment termékeivel tisztában vannak (és akkor még nem is beszéltünk a különböző gyártók eszközeinek inkompatibilitásából adódó problémákról). Ez rettenetesen drága és felesleges. Egy hálózat beruházási és működtetési költsége rendkívül magas lett, hosszú megtérülési időkkel ami szintén az innováció ellen hat, tovább növelve a rendszer statikusságát. A hálózati eszközök rugalmatlansága oda vezetett, hogy az innovatív megoldásokat (pl. terheléskiegyenlítés, energiahatékonyság, biztonság stb.) külön eszközökbe (middlebox) kezdték el telepíteni (pl. tűzfalak, behatolásérzékelők, csomaganalizátorok). Mára ott tartunk, hogy az hálózatba csatolt middlebox-ok száma gyakran nagyobb mint a kapcsolóeszközök száma [23]. Arról ne is beszéljünk, hogy a middlebox-okat szintén menedzselni kell ...

1.2. Az SDN definíciója, síkjai és rétegei

Az SDN paradigma alapjaiban alakítja át azt a képet, ahogyan a számítógép hálózatokról gondolkodunk. Az SDN hálózatokat a következő négy alapelv segítségével definiáljuk:

1. Definíció (Software Defined Networking (SDN)).

- *A vezérlő és adatsíkok szétválasztása. A vezérlési funkciókat kivesszük a kapcsolóeszközökből, amik ezek után egyszerű csomagtovábbító elemekké*

válnak mindenféle intelligencia nélkül.

- *A kapcsolási döntéseket nem csomag, hanem folyamszinten hozzuk meg. Egy folyam azonosítása elég szabadon, a csomagfejléc mezők illesztésével történik. Az illesztés egy rendkívül gyors folyamat (nagyjából egy általánosított logikai ÉS művelet), amiben egy ún. folyamtáblában megnézzük, hogy az éppen feldolgozott csomagra van-e a táblában érvényes bejegyzés. Például definiálhatunk egy folyamot a forrás IP, cél IP párossal, de akár portszámok vagy hardvercím alapján is. Egy folyamhoz aztán a folyamtáblában megadjuk, hogy egy adott kapcsolóeszközben milyen művelet hajtódjon végre a folyamhoz tartozó csomagokon. Ugyanahhoz a folyamhoz tartozó csomagokon értelemszerűen ugyanazokat a műveleteket hajtjuk végre [25], [26]. A folyamszintű kapcsolás lehetővé teszi, hogy eddig különféle hálózati eszközöket (router, switch, tűzfal és egyéb middlebox-ok) közös keretben valósítsunk meg [27]. Például egy tűzfal megvalósítása történhet úgy, hogy egy SDN kapcsolóban egy adott tiltani kívánt folyam csomagjaira az eldobás (DROP) műveletet hajtjuk végre. A folyamszintű kapcsolás rendkívül flexibilis és a képzeletnek szinte csak a folyamtáblák (általában tartalomcímezhető tárban tárolt táblázatok) mérete szab korlátot [9].*

- *A vezérlési logikát (ami hagyományos IP hálózatokban a kapcsolókban van) egy külső entitásba, a kontrollerbe más nevén a hálózati operációs rendszerbe (Network Operating System (NOS)) költöztetjük. A NOS egy szoftver platform (mint pl. az Android) ami teljesen szokványos olcsó hardveren (akár egy mezei laptop) is képes futni és azokat az alapfunkciókat tartalmazza, melyeket a kapcsolóeszközök folyamtábláit fel lehet programozni, akár dinamikusan is. A programozáshoz a NOS a hálózatról alkotott logikailag központosított képpel rendelkezik. Ez azt jelenti, hogy a NOS-nak rendelkezésére áll minden releváns információ a hálózatba csatolt eszközökről és összeköttetésekről, de pl. nem lényeges számára, hogy egy adott kapcsolóeszköz milyen gyártó által került legyártásra. A NOS ilyen értelemben nagyon hasonlít a hagyományos operációs rendszerekre, amely a gyártók által kiadott vezérlőszoftverek (driver) segítségével absztrakt módon tud kezelni egy csomó eszközt. A logikailag központosított azt jelenti, hogy nem feltétlenül vannak egy helyen ezek az adatok,*

- *A hálózat programozható a NOS felett futó alkalmazások segítségével. Az alkalmazások kommunikálhatnak a kapcsolóeszközökkel és dinamikusan változtathatják azok viselkedését. Ez a lehetőség az SDN legnagyobb ígérete.*

A fenti négy alapelvből nagyon sok minden következik. Először is a vezérlés logikai központosítása miatt nem kell elosztott algoritmusokat implementálni (gondoljunk csak bele, mennyivel egyszerűbb egy adott gráf felett a legrövidebb útválasztást leprogramozni). Másrészről a NOS felett futó alkalmazások magas szintű nyelveket is használhatnak ezzel, az alkalmazásfejlesztők sokkal inkább a funkcionalitásra (én nem arra, hogy hogyan fordítsák az adott funkciót egy adott eszköz alacsony szintű nyelvére) koncentrálhatnak. Ráadásul a

vezérlőprogram automatikusan reagálhat a hálózat állapotváltozásaira, miközben a hálózat alapvető működésmódján (pl. legrövidebb útválasztás) nem kell változtatni. Végül, a kontrollerben összegyűjtött és az alkalmazások számára elérhetővé tett információ, sokkal kifinomultabb alkalmazások és szolgáltatások megvalósítását teszi lehetővé.

1.2.1. Az SDN síkjai

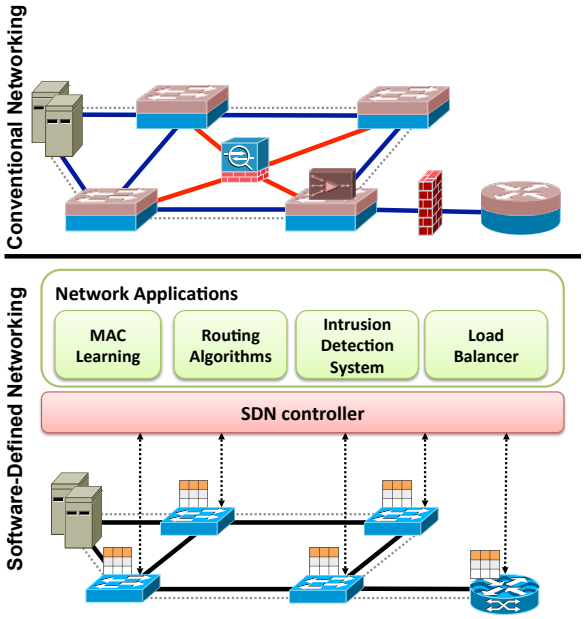
A hagyományos számítógép hálózatokhoz hasonlóan az SDN koncepciót három síkra (adat, vezérlési és menedzsment) oszthatjuk, melyek egyben absztrakciós szintek is. Az absztrakciós szintben azt absztrakciós azt jelenti, hogy az alatta levő dolgokat nem teljes részletességükben és bonyolultságában látjuk, hanem csak egy egyszerűsített „absztrakt" nézetben. Hasonló ez ahhoz, hogy a Linuxnak nem kell azt tudnia, hogy egy adott gyármányú merevlemezre íráshoz pontosan hogyan kell mozgatni a lemezfejeket, hanem a Linux absztrakt szintjén elég azt tudni, hogy egy blokkos eszközről van szó.

Az SDN *adatsíkja* a kapcsolóeszközökből és a közöttük levő összeköttetésekből áll. Az SDN adatsík lehetővé teszi, hogy a hálózati alkalmazások tetszőlegesen felprogramozhassák a kapcsolóeszközöket, miközben elrejti az eszközök hardveres részleteit. Az alkalmazások szempontjából nézve ugyanis teljesen mindegy, hogy egy adott SDN kapcsolóban a folyamtáblák hogyan vannak megvalósítva, elég az, hogy egy megfelelően specifikált interfészen keresztül a folyamtáblák állítgathatók. Az OpenFlow pl. egy lehetséges protokoll, amely megvalósítja a kapcsolók absztrakcióját a vezérlősík felé, melynek működése ezért teljesen analóg az eszköz meghajtókával (device driver) a klasszikus operációs rendszerekben.

A *vezérlősík* feladata, hogy elrejtse az alkalmazások elől a hálózatok elosztottságának bonyolult részleteit. Mindez úgy történik, hogy a hálózatok alapvetően elosztott működését egy logikailag központosított absztrakt nézetbe transzformáljuk. A vezérlősík egyrészt megoldja a kapcsolási logika elosztását a kapcsolóeszközök között, illetve folyamatosan monitorozza az adatsík topológiáját és viselkedését (pl. számlálókkal) és ezt az információt egy megfelelő interfészen keresztül elérhetővé teszi a hálózati alkalmazások számára.

Az SDN *menedzsment* síkja gyakorlatilag nem más mint a megvalósított hálózati alkalmazások (pl. tűzfal, útválasztás, terheléselosztás), amelyek a hálózat topológiájához és a kapcsolóeszközökhöz a vezérlősík által felkínált absztrakt interfészen keresztül hozzáférnek.

Ezen a ponton illik az SDN síkjairól összehordott általánosságokat példával illusztrálni. Vegyük ismét a legrövidebb útválasztást mint hálózati alkalmazást. Ennek futtatása SDN-ben kb. a következőképpen történik. A kapcsolóeszközök megfelelő SDN protokoll (pl. OpenFlow) használatával a dedikált kontrollhálózaton keresztül bejelentkeznek a vezérlősíkban levő kontrollerhez, amely regisztrálja a csatlakozott eszközöket. Ezt az információt egy API-n keresztül az alkalmazás lekérdezi és az összeköttetések felderítésével gráfot épít belőle. A gráf felett egy bármilyen nyelven megírt gráfos API-val (pl. python-igraph) kiszámoljuk a legrövidebb utakat és visszaadjuk a kontrollernek, hogy az útvonalon



1.2. ábra. A hálózat mint platform

levő kapcsolókat állítsa be. A kontroller lefordítja a beállításokat és a kapcsolóeszközökön futó SDN protokoll (pl. OpenFlow) segítségével felkonfigurálja a őket. Az 1.2 ábráról vizuálisan is érzékelhetjük, hogy az SDN azért nagyon más mint a hagyományos megközelítés.

1.2.2. Az adat és vezérlősík szétválasztásából adódó előnyök

A hagyományos hálózatok esetében a vezérlő és adatsíkok ugyanabban az eszközben történő megvalósítása nagyon megnehezíti a hálózat funkcionalitásának bővítését, erről korábban már beszéltünk. Egyszerűen mondva, ha a teljes hálózat vezérlésén változtatni szeretnénk, akkor az összes eszköz vezérlősíkjában egyszerre módosításokat kell végrehajtani. Ez rendkívül nehézkes és drága, hiszen ezt legalább firmvare, rosszabb esetben hardver frissítésekkel kell megoldni, ráadásul teljes mértékben eszközre szabottan. Nem meglepő, hogy ezek után az innovatív hálózati megoldások nem a hálózati elemekbe, hanem plusz egységekbe ún. middlebox-okba költöztek. Ezek aztán még rugalmatlanabbá teszik a hálózat struktúráját.

Ezzel szemben az SDN a vezérlősíkot kiveszi az eszközökből és egy külső elembe, a hálózati operációs rendszerbe, vagy kontrollerbe teszi át. Ennek számos előnye van:

- Sokkal könnyebb a hálózatot új funkciókkal bővíteni, hiszen egy SDN al-

kalmazás egyszerre használhatja, a kontroller által nyújtott információkat és a magas szintű programozási nyelveket.

- Minden alkalmazás használhatja a hálózatról rendelkezésre álló információkat ezért sokkal hatékonyabb szolgáltatások készíthetők és a vezérlősík szoftvermoduljai több modulnál is újrahasznosíthatók.

- Az alkalmazások nagyon könnyen újrakonfigurálhatják a hálózat bármely részében levő kapcsolókat, ezért nincs szükség előre eldönteni, hogy hova helyezzük az egyes funkciókat (pl. terheléselosztó, tűzfal)

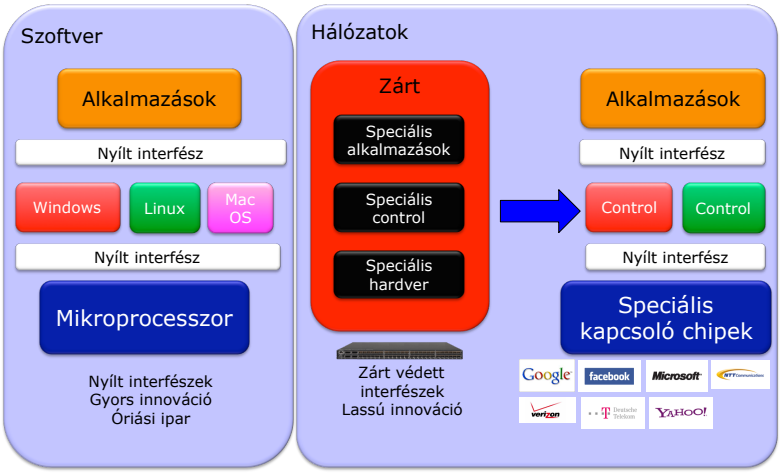
- A szolgáltatások integrációja sokkal egyszerűbb. Pl. egyszerűen megadhatjuk, hogy a terheléselosztó alkalmazásnak nagyobb prioritása legyen a routing alkalmazásnál.

1.2.3. Analógia a szoftver platformok és az SDN között

Már többször érzékeltettük az analógiát a hardver-szoftver platformok és az SDN mint hálózati platform között, most dolgozzuk ezt ki teljesen. Az egyszerűség kedvéért tekintsük az okostelefont, mint platformot. Azt valószínűleg mindenki érzi, hogy az okostelefon egy elképesztően sikeres platform, hiszen okostelefonra alkalmazások milliói érhetők el. Milyen architektúra van emögött a sikeres platform mögött. Legalul ott van a hardver (pl. processzor), amit egy adott hardvergyártó cég legyárt. Ezek a hardverek vezérelhetőek egy nyílt, mindenki számára elérhető interfészen keresztül. Ez azt jelenti, hogy a hardver fölé a nyílt interfész ismeretében bárki írhat operációs rendszert. Ezért a hardver és a operációs rendszerek fejlesztése, innovációja teljesen függetlenül végbemehet, amíg ez az interfész változatlan. Az operációs rendszerek ezt a logikát alkalmazzák felfelé, vagyis egy nyílt interfész biztosítanak az alkalmazások számára, amin keresztül szabályozott keretben hozzáférhetnek a hardver erőforrásokhoz. A nyílt interfész ismeretében (pl. Android SDK) bárki írhat alkalmazásokat az operációs rendszer fölé. Az innováció és a fejlesztések így mind a hardver, mind az operációs rendszer és mind az alkalmazások szintjén párhuzamosan mehetnek, nem meglepő, hogy a szoftveripar elképesztő növekedést produkált az elmúlt pár évtizedben.

A számítógéphálózatok szegmenségben erre a struktúrára való átállás most zajlik le. Hogy miről is kell átállni, az a 1.3 ábrán a hálózatok rész pirossal színezett részében látható. A hagyományos architektúrára az a jellemző ugyanis, hogy mind a hardver, mind az operációs rendszer (vezérlés), mind pedig az alkalmazások fejlesztését egy adott gyártó tudja csak elvégezni, hiszen a rétegek között zárt interfészek vannak, így harmadik fél számára lehetetlen a struktúrába való beavatkozás. Ez nagyon jó az eszközgyártóknak (monopolhelyzet és dől a lé), de nagyon betesz az innovációnak, hiszen minden új megoldásra az eszközgyártók fejlesztőmérnökeinek bólintania kell. A több éves innovációs ciklus pl. az adatközpontok esetében elképesztően hosszú, így nem véletlen, hogy pl. a Google az első között volt aki az SDN mellé állt, sőt a Google adatközpontjainak egy része ma már SDN alapokon működik. Az SDN hasonlóan

a hardver-szoftver architektúrákhoz nyílt interfészeket használ mind a hálózati hardver és a operációs rendszer, mind pedig az operációs rendszer és az alkalmazások között. Bár az SDN architektúra még az elterjedés kezdeti fázisában van, már most sokkal több alkalmazás elérhető SDN platformra, mint ami a hagyományos architektúrán valaha is rendelkezésre állt.

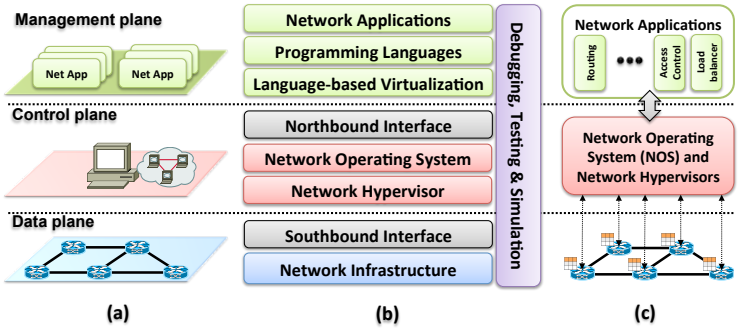


1.3. ábra. A hálózat mint platform

1.3. Kicsit mélyebben: Az SDN rétegei

Az SDN architektúra a síkok alatt tovább bontható az SDN rétegekre. Azért érdemes egy szinttel lejjebb menni, mert a konkrét funkciók, megoldások és érdekességek ezen a szinten mesélhetők el. Hasonlóan más rétegesen elképzelhető architektúrákhoz (mint pl. az emberi hallás) a rétegek és az őket elválasztó interfészek szerepe az, hogy az interfészek rögzítésével az egyes rétegekben az innováció (vagy az evolúció) párhuzamosan történhetnek ezzel a fejlődés üteme sokkal gyorsabb lesz. Az SDN rétegeit az 1.4 ábrán a (b) panel mutatja. Ugyanezen az ábrán az (a) és (c) panelek rendre az SDN síkokat és az SDN rendszer logikai felépítését mutatják párhuzamosan. Aki ezt az ábrát megérti mindent tud az SDN koncepcióról.

A továbbiakban alulról felfelé haladva minden rétegnek szentelünk egy rövid szakaszt, ahol legfontosabb funkciókat és technológiákat ismertetjük. Már most megjegyezzük, hogy a legfontosabb rétegek, mint a déli interfész, a NOS, az északi interfész és az alkalmazások minden valós SDN hálózatban megtalálhatóak, míg pl. a hálózati hypervisor nem kötelezően van jelen, csak ha a hálózatban virtualizációt is megvalósítunk.

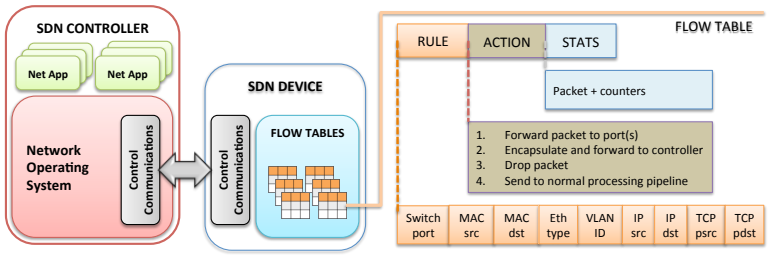


1.4. ábra. Az SDN síkjai, rétegei és rendszerarchitektúrája

1.3.1. 1. Réteg: Infrastruktúra

Az SDN infrastruktúra a hagyományos hálózatokhoz hasonlóan hálózati eszközökből (switch-ek, routerek és egyéb berendezések) áll. Nagy különbség azonban, hogy az SDN eszközök nagyon egyszerű kapcsolóelemek mindenféle beágyazott vezérlőszoftver és egyéb intelligencia nélkül. Az SDN koncepció ugyanis minden intelligenciát a NOS-ba helyez. A kapcsolóeszközökkel szemben egyetlen lényeges kritérium van: az eszközben levő kapcsolási logikát ki kell nyitni és dinamikusan programozhatóvá kell tenni a NOS számára. Az, hogy a kapcsolási logikát ezek után milyen módon oldja meg (pl. általános célú hardver felett szoftverben, vagy speciális hardverrel pl. tartalomcímezhető memóriával) a kapcsolóeszköz az lényegtelen. Sőt ez ad teret az innovációnak hiszen innentől kezdve bárki csinálhat SDN kapcsolót bármilyen alapon ha a kapcsolási logika a NOS előtt nyitva áll. Fontos lépés ez a hagyományos hálózatokhoz képes, hiszen egyetlen NOS segítségével elképesztő heterogenitású eszközparkot (pl. vegyesen HP, CISCO, Juniper switch-ek és egyéb szoftver switch-ek pl. extreme) képesek lehetünk egyszerűen irányítani.

Bár nem az egyetlen de az első gyakorlatban is alkalmazható SDN architektúra az OpenFlow. A OpenFlow kapcsolók logikai működése nagy vonalakban a következő (1.5 ábra). A kapcsolóeszközben egy (vagy több) programozható táblázat, ú.n. folyamtábla van. Minden bejegyzés a folyamtáblában három részből áll: (1) illesztési szabály, (2) akciólista és (3) számlálók. Az illesztési szabály megmondja, hogy az utána következő akciólista mely (pl. Forrás és cél IP és port-okkal adott) folyamokra fusson le. Az akció lehet pl. továbbítás, eldobás stb. A számlálók pedig monitorozzák, hogy a bejegyzésre hány csomag illeszkedett. A folyamtáblába a kontroller (NOS) szabadon hozzáadhat illetve törölhet bejegyzéseket ezzel megadja a hálózat működését. A folyamtáblák kezelése és egyéb infrastruktúrával kapcsolatos adatok áramoltatása a kapcsolók és a kontroller között a déli interfészen keresztül történik.



1.5. ábra. OpenFlow-képes eszköz logikai felépítése

1.3.2. 2. Réteg: Déli interfész

A déli interfész specifikálja a kontroller és a kapcsolóeszközök közötti kommunikáció módját. A déli interfész megfelelő részeit ezért mind a kapcsolóeszközökben, mind a kontrollerben implementálni kell. A hagyományos hálózatokban a vezérlősík és az adatsík kommunikációját a kapcsolóeszközök fizikai felépítése alapvetően befolyásolja ezért van az, hogy ahány különböző gyártó és eszköz annyi különféle kontrollfelület. Az SDN koncepció zászlóshajója az OpenFlow nagy dobása az volt, hogy először definiált olyan déli interfészt, amely a kapcsolóeszközök fizikai felépítésétől függetlenül le tudta írni a logikai viselkedést. Az OpenFlow hatékonyságát és népszerűségét jelzi, hogy szinte minden fontosabb hálózati eszközgyártó (CISCO, Juniper stb.) palettáján megtalálhatók OpenFlow-kompatibilis eszközök. Az OpenFlow egy olyan közös platform, ami felett gyártófüggetlen módon lehet a hálózat funkcionalitását manipulálni.

Az OpenFlow lelke az OpenFlow protokoll, amely a kapcsolóelemek és a kontroller (még mindig NOS) közötti üzeneteket definiálja. Alapvetően háromféle üzenetcsoport van. Az első csoport eseményvezérelt üzeneteket tartalmaz, melyeken keresztül a kapcsolóeszköz jelezni tudja, ha valami történt a fizikai környezetében (megszakadt egy link, lement egy port stb.) A második csoport a hálózatban menő forgalomról tájékoztatja a kontrollert (pl. a folyamtábla számlálóinak lekérdezése). Az utolsó csoport pedig a kapcsoló logikai működésének megadására szolgát. Ezeken keresztül értesülhet a kontroller, ha egy kapcsoló nem tudja mihez kezdjen egy adott csomaggal és egy megfelelő folyamtábla bejegyzéssel megadhatja a kívánt működést.

1.3.3. 3. Réteg: Hálózati hypervisor

A virtualizáció – melynek alapeleme az ú.n. hypervisor, ami a fizikai eszköz felett virtualizálja a virtuális rendszerek számára szükséges erőforrásokat – rendkívül elterjedt technológia a számítógépes rendszerek világában. Mindezt jól mutatja, hogy a virtuális szerverek száma a különböző adatközpontokban ma már meghaladja a valós fizikai szerverekét. Miért ez a nagy siker?

Először is azért, mert a virtualizáció lehetővé teszi, hogy ugyanazt a fizikai rendszert (számítógépet) több egymástól független virtuális rendszer egy időben

használja. A virtualizáció jelentette az alapját a felő számítástechnikának (cloud computing), ahol a felhasználók rugalmasan foglalhatnak erőforrásokat (CPU, memória, háttértár) egy hatalmas erőforrás silóból sokkal olcsóbban mintha megvásárolnák a számukra szükséges fizikai eszközöket. Ennek oka, hogy a felhő szolgáltató sokkal jobb kihasználtsággal tudja üzemeltetni a rendszerét, amely így sokkal gazdaságosabban működik (az, hogy egy felhasználó a gépét száz százalékban ki tudja használni olyan ritka mint a fehér holló).

Másodszor a virtualizáció azt is lehetővé teszi, hogy a virtualizált rendszereket/szolgáltatásokat elmozgassuk (pl. közelebb a felhasználóhoz), fel - vagy éppen leskálázzuk (ha nagyobb az érdeklődés a szolgáltatásra több szervert indítunk, ha kisebb akkor lekapcsoljuk ami nem szükséges).

A virtualizáció hulláma most éri el a számítógéphálózatok világát. Első hallásra talán meredeknek hangzik, hogy hálózatokat virtualizáljunk. A hálózatok világában megőszült mérnökök közül sokan csak legyintenek amikor ezt meghallják. Pedig a hálózatvirtualizációnak, amennyiben a hálózatot erőforrásnak (és miért ne tekintenénk annak) tekintjük teljesen ugyanaz a logikája és legalább olyan erős a létjogosultsága mint a számítógépek virtualizációjának. Egy hálózat üzemeltetőjének elemi érdeke, hogy a hálózat kihasználtsága minél jobban közelítse a száz százalékot. Amennyiben ezt nem tudja saját felhasználóival biztosítani, úgy a hálózat kihasználatlansága gyakorlatilag veszteség. Ilyenkor érdemes a hálózatot más (virtuális) szolgáltatók felé megnyitni és ugyanazon a fizikai hálózati infrastruktúrán több virtuális hálózatot párhuzamosan üzemeltetni. A hálózatvirtualizáció még fontosabb adatközpontokban, ahol az üzleti partnerek nemcsak a szervereket, de az őket összekötő hálózati megoldásaikat is szeretnék egyre inkább a felhőbe tolni. Márpedig ez a hagyományos technológiával vagy egyáltalán nem, vagy csak hosszú idő alatt megoldható feladat.

Az SDN kulcsfontosságú vívmánya, hogy a hálózatvirtualizáció elképesztően egyszerűen megoldható. A legegyszerűbb megoldás, ha a kapcsolóelemek és a kontroller(ek) közé egy proxy-t iktatunk be, ami pl. bizonyos kapcsolóktól való üzeneteket csak a megadott kontrollereknek továbbít. Vagy pl. különböző folyamtereket különböző kontrollerekhez lehet irányítani (így pl. IP cím tartományokkal különíthetünk el virtuális hálózatokat). Ezt az elvet használja pl. a FlowVisor nevű alap SDN virtualizációs technika. Ma már ennél sokkal fejlettebb hálózatvirtualizációs megoldások is vannak.

1.3.4. 4. Réteg: Hálózati operációs rendszerek / kontrollerek

A operációs rendszerekkel analóg módon a hálózati operációs rendszer (Network Operating System NOS) egy logikailag központosított nézetbe transzformálja a hálózat erőforrásait és magas szintű API-t valamint alapvető szolgáltatásokat ad a fejlesztők kezébe, hogy a hálózat erőforrásait alkalmazásokon keresztül vezéreljék. Alap szolgáltatások pl. az eszközök és a topológia felderítése, eszközök konfigurációja és monitorozása. A NOS segítségével a fejlesztőknek már nem kell elosztott protokollokban gondolkodniuk, egyszerűen megírhatják az alkalmazást, a NOS pedig gondoskodik az eszközök megfelelő beállításáról. Ez

természetesen csökkenti a hibalehetőségeket és nagy mértékben elősegíti az innovációt. Ne feledkezzünk meg azért arról, hogy ez alapvetően a NOS és az eszközök közötti dedikált kontrollhálózat miatt tehető meg.

Az utóbbi években szinte minden fontosabb programnyelven (c/c++, python, java, ruby) írtak NOS-t az OpenFlow architektúrához. Ezek általában jól strukturált API-t adnak a fejlesztők kezébe a megfelelő programnyelven. Így viszont némi bábeli zűrzavar alakult ki a NOS és az alkalmazások kommunikációjában. Például egy pythonban írt hálózati alkalmazás nem képes akármelyik NOS felett futni. Erre a problémára jelenthet majd megoldást a NOS és az alkalmazások közötti ú.n. északi interfész szabványosítása. Jöjjön tehát most ez.

1.3.5. 5. Réteg: Északi interfész

A déli és az északi interfész az SDN két kulcsfontosságú absztrakcióját valósítja meg. Mint már említettük, a déli interfész egy kapcsolóelemet absztrakcióját valósítja meg (aki még mindig nem érti mi az az absztrakció előadás alatt, vagy után feltétlen kérdezzen), vagyis lehetővé teszi, hogy egységes módon megadjuk a kapcsolóeszköz logikáját a fizikai felépítéstől teljesen függetlenül. Az északi interfész a teljes hálózat absztrakcióját valósítja meg, így az északi interfészt használó elemek nem egy való hálózatot látnak, hanem pl. egy gráfot. A déli interfészre már van széles körben elfogadott javaslat (az OpenFlow protokoll), az északi interfészre jelen pillanatban nincs elfogadott protokoll de a fejlesztés kutatás nagyban folyik. Az északi interfész egy teljesen szoftveres interfész, míg a déli inkább hardver közeli. A szoftveres interfészek kialakulásában, elfogadásában a tapasztalat szerint inkább az implementáció szokott döntő lenni. Ezért a közeljövőben várható számos javaslat és elérhető megvalósítás, amelyek közül valamelyik minden bizonnyal de facto szabvánnyá fog válni.

A standard északi interfész kialakulása elengedhetetlen ahhoz, hogy a különféle platformon fejlesztett NOS-os és alkalmazások együtt tudjanak működni. Az északi interfész számos analógiát mutat a POSIX szabvánnyal, amely gyakorlatilag megszabja azt, hogy hogyan kell olyan szoftvert írni, amely bármilyen Unix-szerű operációs rendszeren képes futni. Az hordozhatóság tehát biztosított a POSIX-ot implementáló operációs rendszerek között. (Érdekes dolog, hogy egy időben még a Microsoft is implementálta a POSIX szabvány bizonyos részeit, de mára teljesen felhagyott vele.)

1.3.6. 6-7. Réteg: Programozási nyelvek

A programozási nyelvek evolúciója talán mindenki számára ismert. A hardver jelentette platformra először a hardver közeli nyelvek (pl. assembly) fejlődtek ki. As assembly már absztrakt formában kezeli a gép bizonyos részeit, így egy új platformot hoz létre, melyen új nyelvek evolúciója kezdődhet meg. Így jutunk el szépen egyre feljebb és feljebb a C/C++-on keresztül a magas szintű nyelvekig (pl. Python és Java).

Ehhez nagyon hasonló amit az SDN programozásánál látunk. Az SDN legelterjedtebb alacsony szintű nyelve maga az OpenFlow protokoll. Ha tisztán az OpenFlow üzeneteivel akarnánk egy hálózatot vezérelni, az nagyon bonyolult lenne, hiszen csomó hardver közeli apróságra kellene figyelni (hasonlóan az assembly-hez), mint pl. konkurens folyamtábla-szerkesztés, átfedő folyamtábla bejegyzések, dinamikus működésből adódó inkonzisztens állapotok stb. Ráadásul ilyen alacsony szintű nyelven nehéz jól strukturált újrafelhasználható kódot írni.

Ezen hivatottak segíteni a magasabb szintű nyelvek. A fő céljuk ezeknek a nyelveknek az, hogy a fejlesztőknek ne kelljen annyit a hálózat megvalósításával foglalkozniuk, ehelyett a konkrét megoldandó feladatra (logikai működés) tudjanak fókuszálni. A magasabb szintű nyelvek használatának legfontosabb előnyei:

- A hálózat magas szintű absztrakciója, melyen keresztül a hálózat funkcionalitása (pl. legrövidebb útválasztás) könnyen megadható. A fordító mechanizmusok feladata ezután, hogy a magas szinten megadott leírást végül konkrét OpenFlow üzenetekké transzformálják.

- Produktív, problémaközpontú környezet kialakítása, a fejlesztés és innováció gyorsítása, hibalehetőségek minimalizálása.

- Moduláris, újrahasznosítható kód létrehozása.

- Nyelv alapú hálózati virtualizáció. Ez kb. azt jelenti, hogy a hálózat elemeihez virtuális objektumokat rendelhetünk (egy kapcsolóhoz akár többet is), így magán a hálózati alkalmazáson belül hálózatvirtualizációt valósíthatunk meg.

1.3.7. 8. Réteg: Alkalmazások

Az SDN architektúra (SDN stack) tetején ülnek az alkalmazások. Ezek adják meg a hálózat funkcióját, működési logikáját. A legegyszerűbb funkció pl. a routing. Egy végletekig leegyszerűsített routing alkalmazás pl. hogy A és B hosztok tudjanak egymással kommunikálni. Ehhez tulajdonképpen azt kell megvalósítani, hogy a NOS-tól lekérjük a hálózat topológiáját, útvonalat számítunk rajta a hosztok között és utasítjuk a NOS-t, hogy állítsa be a megfelelő kapcsolókat az útvonalon.

Az alap útválasztáson kívül, nagyon sokféle SDN alkalmazást fejlesztettek az elmúlt években. A hagyományos alkalmazások közé tartozik, a routing-on kívül a terheléselosztás és biztonság. Ezeken felül van alkalmazás, mely a hálózat hibatűrését növel, energiafogyasztását minimalizálja, virtualizációt valósít meg, szolgáltatásminőséget biztosít (QoS), mobil eszközöket kezel, adatközpont hálózatot optimalizál vagy éppen hálózati kódolást valósít meg. Ráadásul ezeket az alkalmazásokat egymással kombinálni is lehet, ami elképesztő testre szabhatóságot nyújt a hálózatok világában.

1.4. NFV

Az NFV nem más, mint a hardveres middlebox-ok virtualizált verziója. Ilyenekbe általában olyan funkcionalitást szoktak tenni, ami a kapcsolókban nem valósítható meg (pl. részletes csomagvizsgálat, forgalomanalízis, hálózati kódolás, behatolás detektálás stb.). A middlebox-ok virtualizációja kényelmessé teszi használatukat, könnyen menedzselhetővé varázsolja őket, a virtualizált verziókat könnyű áthelyezni, frissíteni vagy éppen kivonni a forgalomból.

1.5. Ellenőrző kérdések

- Melyek a hagyományos hálózatok legfontosabb jellemzői, előnyei és hátrányai!
- Ismertesse a hagyományos hálózatok és az SDN hálózatok közötti legfontosabb architekturális különbségeket!
- Ismertesse a NOS szerepét és funkcióit az SDN architektúrában!
- Soroljon fel néhány SDN alkalmazást!
- Érveljen a hálózatvirtualizáció mellett!
- Miben hatékonyak a magas szintű programozási nyelvek az SDN architektúrában?
- Mit jelent az absztrakció és mi a jelentősége?

1.6. Ajánlott irodalom