Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Költséghatékony erőforrás-vezénylés heterogén felhő környezetben

Készítette:
Haja Dávid, Szalay Márk

2017. október 26.

Konzulensek:
Dr. Sonkoly Balázs
Dr. Toka László

Tartalomjegyzék

Kivonat **4**

Abstract **6**

1. Bevezetés **8**

1.1. Heterogén felhő 8

1.2. Hálózati funkciók virtualizálása 11

1.3. Online hálózati szolgáltatások létrehozása 12

2. Az erőforrásvezénylő probléma **14**

3. A DARK algoritmus **18**

3.1. Szimulációs keretrendszer 18

3.2. Végrehajtási sorrend kialakítása 19

3.3. Szolgáltatás gráf leképzése az erőforrás gráfra 20

3.4. VNF migrálás fog-ból cloud-ba 23

3.4.1. Kompatibilis ködök kiválasztása 25

3.4.2. Migrálási lehetőségek összegyűjtése 25

3.4.3. Migrálások végrehajtása 26

4. A DARK hatékonyságának mérése **27**

4.1. Hálózati topológiák generálása 28

4.2. Szolgáltatás igények generálása 30

4.3. Szimulációs eredmények 31

5. Konklúzió **36**

Köszönetnyilvánítás

Köszönjük mindenekelőtt Dr. Sonkoly Balázsnak és Dr. Toka Lászlónak, hogy konzultációs tevékenységükkel folyamatosan irányt mutattak munkánk során és beszélgetéseinkkel hozzájárultak a témában való szemléletmódunk formálásához. Továbbá köszönettel és hálával tartozunk Németh Balázsnak és Szabó Mártonnak, akik segítségükkel hozzájárultak a dolgozatunk sikeres elkészüléséhez.

Szintén köszönettel tartozunk a BME TMIT tanszékének, hogy rendelkezésünkre bocsájtotta a munkánkhoz szükséges eszközöket és infrastruktúrát.

Kivonat

A távközlő hálózatok területén egy új irányelv figyelhető meg, miszerint a nem is olyan távoli jövőben mindenki mindenkivel illetve minden mindennel legyen összekapcsolódva, egy egységes tudásbázist biztosítva, melyből a felhasználók elérhetik a számukra szükséges információkat. A hagyományos eszközök fokozatos okossá tétele, vagyis a „kapcsoljunk internetre mindent" szemlélet egyre inkább terjedőben van. Egymás után nyílnak az új okos hotelek, irodák vagy házak, de például a közlekedés területén is számos forradalmian új megoldást találhatunk (például áruszállító drónok, önvezető autók, forgalom vezérlés, stb.).

Nem csak az eszközöknek, hanem a felhasználóknak is bővülnek az elvárásaik a hálózatokkal szemben. Ilyen lehet például a virtuális valóság vagy a tapintás alapú internet. Az így létrejövő új követelmények eredményezik azt, hogy a hálózati operátoroknak a versenyképesség szempontjából olyan távközlő hálózatokat és új szolgáltatásokat kell létrehozniuk, melyek képesek kezelni az IoT (Internet of Things) miatt megnövekedett kliensek számát, a szenzorok által folyamatosan generált nagy mennyiségű adatot, az új felhasználói igények azonnali kiszolgálását és a számítási kapacitást, amely ezeket együttesen képes kezelni.

A ma is használt hálózatok meglehetősen rugalmatlanok az ilyen típusú alkalmazások bevezetésére, hiszen habár a sávszélességi követelményeket mindinkább teljesítik, mégis az azonnali reakcióhoz szükséges nélkülözhetetlen tulajdonságnak, a késleltetési igénynek nem felelnek meg. E kihívások teljesítésére a megoldás a dinamikusan változtatható szoftverizált hálózatok (Software Defined Networking, SDN) és a virtualizálásra alkalmas felhő adatközpontok összekapcsolásában és a hálózati funkciók virtualizálásában (Network Function Virtualization, NFV) található. Egy szolgáltatási kérés több funkcióból (Service Function Chain, SFC) állhat össze, melyeknek különböző erőforrásigényei lehetnek a hálózattal és futtató szerverekkel szemben (CPU, memória, sávszélesség, késleltetés stb.). Magától értetődően a klienshez közeli szervereken szükséges létrehozni a leginkább késleltetés igényes funkciókat és e kritérium értékének enyhülésével haladhatunk beljebb a távolabb lévő felhőrendszerek felé. Erre alkalmas a fog networking architektúra, mely kiterjeszti a hagyományos felhő architektúrát azzal, hogy a hálózat szélein virtualizálásra alkalmas csomópontokat helyez el. Az elkövetkező 5G rendszer egyik eleme, hogy

a különböző szolgáltatók egy egységes felhő hálózatot hoznak létre, így lehetővé téve bármely felhasználónak a kiszolgálását az éppen aktuális helyzetétől függetlenül. Azonban a szolgáltatóknak céljuk lehet, hogy minél jobban a saját erőforrásaik használatára törekedjenek, elkerülve a plusz költségeket a többi hálózati operátortól igénybevett erőforrásokért cserébe.

Tudományos dolgozatunk egy olyan általunk létrehozott erőforrásvezénylő algoritmust mutat be, mely a fent említett feltételeket teljesíti és minimalizálja az egyéb szolgáltatóktól igénybevett erőforrások költségét. A megvalósítás során szempont volt a bejövő szolgáltatás igények lehetséges azonnali kiszolgálása, az algoritmust futtató szolgáltató saját infrastruktúra használatának maximalizálása, illetve a késleltetésérzékeny funkciók kötelező implementálása - a kliens mozgását követve - a közeli erőforrásokon. Az algoritmus megalkotása során különös figyelmet fektettünk a valós idejű hálózati funkció mozgatására (live migration), ezzel is csökkentve az idegen hálózatból való erőforrás bérlés költségeit. A dolgozatunkban különböző szimulációs helyzetekben mutatjuk be az általunk javasolt algoritmus teljesítményét, összehasonlítva azt egyéb megoldásokkal.

Abstract

In the field of telecommunications a new emerging trend can be observed, which aims to connect everyone with everyone and everything with everyting in the near future, thus forming a unifyed knowledge base, from where the users can reach all required information. Making the traditional devices smart and connecting them to the internet is a phenomenon that is present nowadays. Smart hotels, offices, houses are opening one after another, and also for example in the field of transportation we can find many revolutionary solutions (e.g. freight drones, self driving cars, traffic control).

Not only the requirements of the devices, but also the end-users' expectations have increased towards the networks. These can be for example the virtual reality or the tactile internet. The newly created applications force the network operators, in order to maintain their competitiveness, to establish networks and services that can serve a large number of IoT (Internet of Things) devices, a large amount of traffic generated by the sensor devices, user demands with near real-time response times and offer increased compute capacity.

Today's widely deployed networks are not flexible enough to fulfill all of these new use-cases. Although they more or less meet the bandwidth requirements, they cannot satisfy the latency requirements which are essential for low response times. To overcome these challenges, the solution can be found in the dynamically reconfigurable software-based networks (Software Defined Networks, SDN), the interconnected cloud datacenters, which provide loads of compute resources, and the virtualization of the network functions (Network Function Virtualization, NFV). One service request may consist of more elementary functions (Service Function Chain, SFC), which can set different resource requirements to the underlying network and to the virtualization environments (e.g. CPU, RAM, bandwidth, latency). As can be felt, that servers which are the closest to the clients are the best suited to run functions with the strictest latency, and as these requirements weaken we can move functions closer to the datacenters located in the core of the network. The fog is a conform network architecture with the considerations written above, which extends the traditional cloud concept with computing nodes at the edge of the network. In the future 5G networks an important element is that the different service

providers are going to creat a common unified cloud network, thus enabling each user to be served independently of their actual physical location. In such a multi-provider cloud environment the goal of the participants may be to utilize their own infrastructure the most efficiently, and to minimalize the additianal cost that comes from using external resources.

The target of our scientific paper is to propose a resource orchestration algorithm which is conform with the conditions written above, and minimizes the cost of resource usage at other providers. During the implementation we focused on the fastest possible serving of incoming service requests, on maximalizing the utilization of the service provider's internal infrastructure and the placement of the latency-aware network functions - by following the movement of the user - on the suitable resources. When we were designing our algorithm, we paid attention to support the live migration of virtual service functions, thus further decreasing the cost of resources from external infrastructures. In our work, we show our proposed algorithm in different simulation scenarios and compare its performance to other solutions.

1. Bevezetés

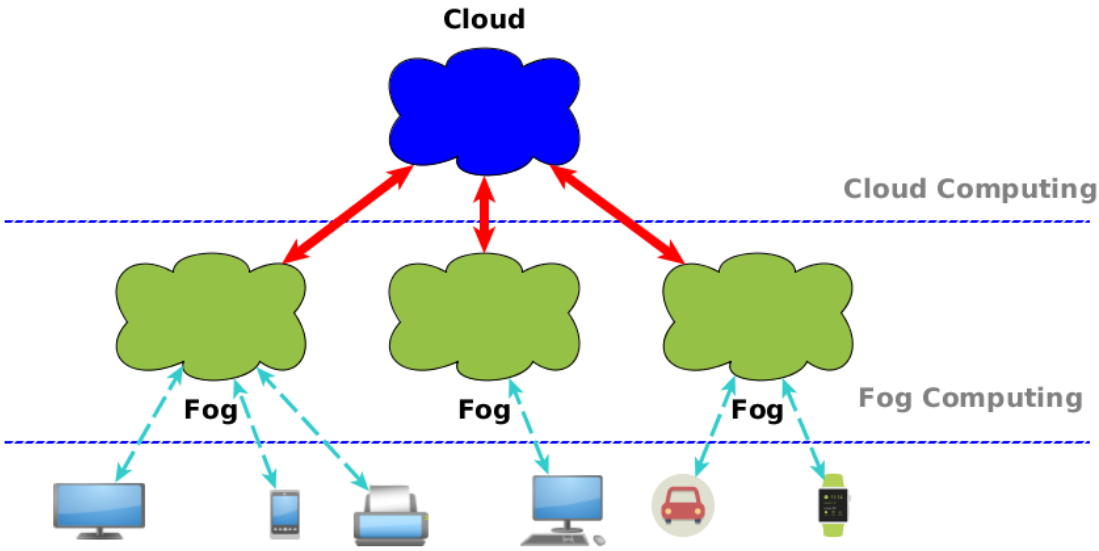
Az 5G hálózatok [1] megjelenése várhatóan merőben új hálózati szolgáltatások elterjedését teszi majd lehetővé [2]. Ez természetesen nem csak a rádiós interfészen történő kommunikációt érinti, hanem a mai hálózatok belső topológiáját is, melyben különböző változtatások is szükségesek lesznek. A mai hálózatok jelenlegi formájukban nem eléggé rugalmasak ahhoz, hogy az új szolgáltatások által támasztott várható követelményeknek megfeleljenek. Például a hálózati funkciók futtatása most még általában speciális célhardverekhez kötött (például hardveres tűzfal). A hálózati funkciók virtualizálása révén azonban lehetőség nyílik ezeket a szolgáltatásokat általános célú szervereken futtatni, ezáltal megszűnnek a hardverek fizikai elhelyezkedéséből fakadó korlátok, és sokkal dinamikusabban konfigurálható hálózatot kapunk, mely így már képes lehet megfelelni a növekvő elvárásoknak.

A hálózat belsejében elhelyezkedő adatközpontok helyett a hálózat széleihez közel elhelyezett számítási kapacitások megjelenése (pl. Mobile Edge Computing, MEC [3]) számos új lehetőséget nyithat meg előttünk. Szükség esetén képesek lennénk hálózati funkciókat a végfelhasználóhoz nagyon közel, kis válaszidőt garantálva, futtatni, míg az egyéb komponenseket - melyek nem annyira érzékenyek a késleltetésekre - célszerű a szűkös erőforrású végpontoktól távol, a hálózat belsejében [4] üzemeltetni. Ezáltal lehetőség nyílik olyan új szolgáltatások bevezetésére, amelyek a jelenlegi hálózati architektúrán nem megvalósíthatók. Ilyenek például az önvezető autók és ipari robotok vezérlése, az intelligens közlekedési rendszerek működtetése vagy a hálózat peremén elhelyezett tartalomgyorstárazási (content caching) szolgáltatások [5]. A továbbiakban ezt az új irányt lehetővé tévő fontosabb megoldások részletes bemutatásai következnek.

1.1. Heterogén felhő

Heterogén felhő alatt egy olyan komplex rendszert értünk, mely kiterjeszti az informatika területén hagyományos értelemben vett „felhő” technológiát a felhasználókhoz közel elhelyezett, a rendszerhez hozzákapcsolt számítási és hálózati erőforrásokkal. Az ilyen felhasználóközeli erőforrások a gyakorlatban csoportosítva helyezkednek el. A szakirodalomban elterjedt megnevezése egy ilyen csoportnak:

köd (angolul: fog). Ez a kiterjesztett rendszer lehetővé teszi különböző új fajta szolgáltatások létrehozását. Az ilyen új típusú szolgáltatások jellemzően a ködök által nyújtott új lehetőségeket használják ki, mint például: kis késleltetés, helyfüggetlenség, széleskörűen elosztott földrajzi elhelyezkedés, mobilitás, vezeték nélküli kapcsolat. Ebből a rendszerből előálló paradigmát ködalapú számítástechnikának (Fog Computing) nevezzük [5, 6].



1. ábra. Ködalapú számítástechnika (fog computing) architektúrája

Felépítését tekintve a Fog Computing három rétegre bontható (1. ábra). A legfelső rétegben található egy vagy több központi felhő szolgáltatás, mely lehet egy szolgáltatóhoz tartozó privát felhő, vagy akár az interneten is bárki számára elérhető publikus felhő. A legelterjedtebb megvalósítások szerint a felhőt használó alsóbb rétegeknek és felhasználóknak ez a szolgáltató végtelen fizikai erőforrást képes nyújtani, és a felhasznált erőforrások után kell fizetni. A legismertebb ilyen szolgáltatók az Amazon és a Microsoft.

A ködalapú számítástechnika által támogatott szolgáltatások komponenseinek jelentős része nem használhatja a központi felhőt, ugyanis a szolgáltatás működéséhez szükséges egy adott maximális késleltetés biztosítása egyes komponensek között. A központi adatközpontok mint említettük többnyire az Interneten, vagy

egy gerinchálózaton keresztül érhetők el, így az ott elhelyezett komponensekhez nem garantálható egy adott mértékű maximális késleltetés. Természetesen az olyan komponensek elhelyezésének az eldöntése, melyek a késleltetésre nem érzékenyek, nem technológiai kérdés, sokkal inkább üzleti.

A középső szinten a ködök találhatók. A ködök fizikai és hálózati erőforrásokat nyújtanak az alattuk található rétegnek, a felhasználóknak. Az erőforrás mennyiségét tekintve a köd rétegben kevesebb számítási és tárolási kapacitás található, mint a felette lévő felhőben, azonban sokkal komplexebb számításokat képes elvégezni, mint az alatta lévő szinten található eszközök. A köd(ök)ben található erőforrások mennyisége erősen korlátos, azonban ezen a szinten egy szolgáltató képes garantálni különböző hálózati feltételek teljesítését, mint például maximális késleltetés két fizikai csomópont, és a rajtuk futó szolgáltatás komponensei között. A ködökre jellemző továbbá, hogy földrajzilag elosztottan helyezkednek el. Csatlakozási pontokat nyújtanak az alsóbb rétegbeli végfelhasználói hardvereknek. Az egyes ködökben található fizikai csomópontok kapcsolatára jellemző, hogy alacsony a késleltetés, és viszonylag nagy a sávszélesség közöttük.

Legalul találhatók az olyan fizikai eszközök, melyek a felhasználókhoz tartoznak és a szolgáltatásokat igénybe veszik. Az ilyen hardverek a ködök által nyújtott kapcsolódási pontokhoz csatlakoznak, akár vezetékesen, akár vezeték nélkül. Jellemző rájuk az alacsony számítási és tárolási kapacitás, továbbá az itt található hardverek egy része nincs helyhez kötve.

A gyakorlatban legelterjedtebb Fog Computing szolgáltatások működése hasonló. Futásuk során a szolgáltatásban résztvevő felhasználói eszközök adatokat gyűjtenek és továbbítanak a ködben lévő szolgáltatási komponenseknek, melyek meghatározott késleltetésen belül válaszolnak az eszközöknek és menedzselik a működésüket. Az olyan komponensek, melyek a késleltetésre nem érzékenyek, például statisztikai adatok gyűjtését végző elemek, a korábban bemutatott központi felhőbe kerülhetnek.

A Fog Computing ideális rendszert és hálózatot teremt napjaink egyik leggyorsabban fejlődő, legfoglalkoztatottabb mérnöki területének a „dolgok internete” (Internet of Things, IoT) technológiáknak [7,8]. Leggyakoribb IoT-ra jellemző alkalmazások működésük során a legalsó szinten található hardverek különböző környezeti vagy belső állapotényezőket monitoroznak. A mért értékeket elküldik a hálózaton

egy távoli számítási erőforrásnak, mely az összegyűjtött adatokból meghatároz egy új állapotot, és válaszol a vele kapcsolatban álló eszközöknek, valamint tovább küldi az adatokat egy másik távoli egységnek, ami a statisztikákat gyűjti össze és készíti el. Megfigyelhető az ilyen működés során, a korábban bemutatott Fog Computing hierarchiába az egyes elemek célszerű elhelyezkedése.

A Fog Computing-ban megvalósítható megoldásokra akár több lehetséges példát is olvashatunk az openfog konzorcium által kiadott dokumentumban [9]. Egy okosváros esetén, ahol az utak mellett különböző szenzorok monitorozzák az aktuális forgalmat fontos követelmény lehet, hogy a lehető leggyorsabban összegezzük a különböző szenzorok által küldött adatokat és a többszörös forrásoknak köszönhetően az aktuális forgalmi viszonyokról információhoz jussunk. Szintén feladat, - például egy forgalmi dugó esetén - hogy az éppen közlekedő autókat a lehető leggyorsabban értesítsük, hogy lassítsanak, vagy a következő kijáratnál hajtsanak le, ezzel is tehermentesítve a forgalom szempontjából kritikus szakaszokat. Ebben az esetben azon funkciónak, mely a szenzorokkal és az autókkal kommunikál, célszerű az utat lefedő ködben futnia, hogy a késleltetést minimalizáljuk, hiszen akkor már nincs értelme a pályáról kihajtási figyelmeztetésnek, hogy ha az adott jármű már elhagyta a kijáratot. Ebben a konkrét példában a fog jelentheti a mobil bázisállomásokban elhelyezett szervereket, amik az antenna által lefedett cellában tartózkodó szenzorokkal és járművekkel kommunikálhatnak. Ekkor az antenna egy szolgáltatás elérési pontként (angolul: Service Access Point - SAP) üzemel, mivel ezen keresztül képesek elérni az okos eszközök a fog-ban futtatott virtuális funkciókat.

1.2. Hálózati funkciók virtualizálása

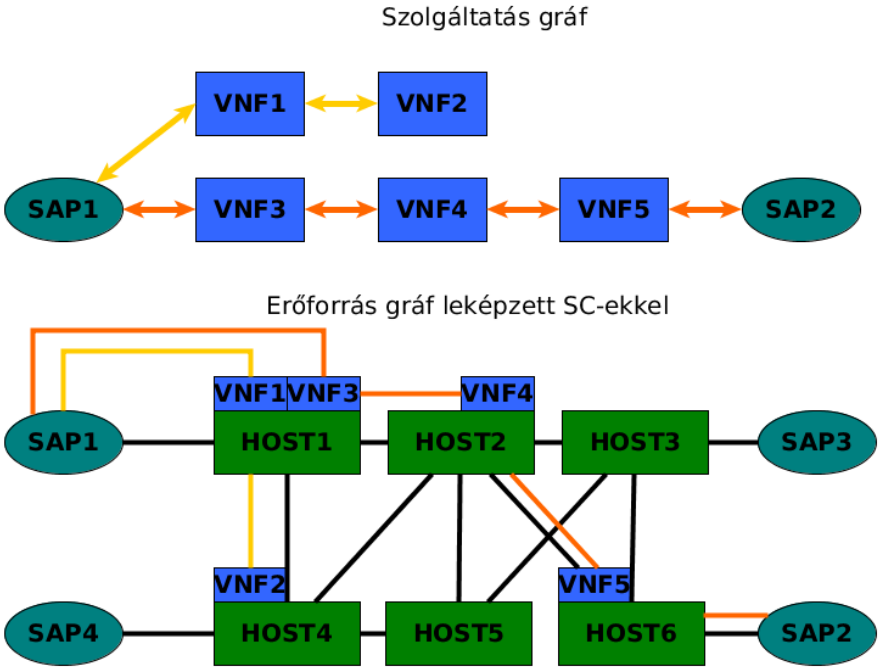
Az NFV (Network Function Virtualization) egy olyan hálózati koncepció, mely különböző virtualizációs technológiák felhasználásával hálózati eszközöket és azok belső működését virtualizálja [10]. Az NFV előnye, hogy nincs szükség konkrét gyártóspecifikus hardverre, a virtualizált megoldások általános célú hardveren futtathatók. Egy virtuális szolgáltatás több elemből állhat össze, mely elemek tetszőlegesen programozhatók a felhasználási igény szerint, könnyen skálázhatóak és mozgathatóak a hypervisor szerverek között. Egy ilyen virtualizált hálózati elemet VNF-nek (Virtual Network Function) nevezünk. A különböző VNF-ek között,

lehetőségünk van logikai kapcsolatok definiálására is, továbbá működési elvárásokat fogalmazhatunk meg mind a VNF-ekre, mind a közöttük lévő kapcsolatokra. Ilyen követelmények lehetnek például a VNF-re vonatkozó CPU, RAM és tárhely kritériumok, illetve a kapcsolatokra vonatkozó sávszélesség és késleltetés. Az SFC (Service Function Chaining) nem más, mint virtualizált hálózati funkciók rendezett listája, melyben az elemek egymással kapcsolatban állnak és egymással kommunikálnak [11, 12].

1.3. Online hálózati szolgáltatások létrehozása

Ebben a fejezetben a Network Service Embedding (NSE) probléma általános esetekben történő alkalmazását mutatjuk be. A hálózati szolgáltatások modellezése úgynevezett szolgáltatás gráfok segítségével történik (angolul: Service Graph, SG). Ezekben a szolgáltatásokat a gráf csomópontjai, míg a köztük lévő logikai kapcsolatokat a gráf élei írják le. A Network Serivce Embedding (vagy Service Graph Embedding) eljárás célja, hogy a virtuális hálózati szolgáltatásokat leképezze a fizikai infrastruktúrára, méghozzá olyan módon, hogy figyelembe vegye az egyes csomópontokban rendelkezésre álló erőforrások (számítási, tárolási, memória), valamint a szolgáltatás gráf éleire megfogalmazott követelmények (maximális késleltetés, minimális sávszélesség) teljesülését.

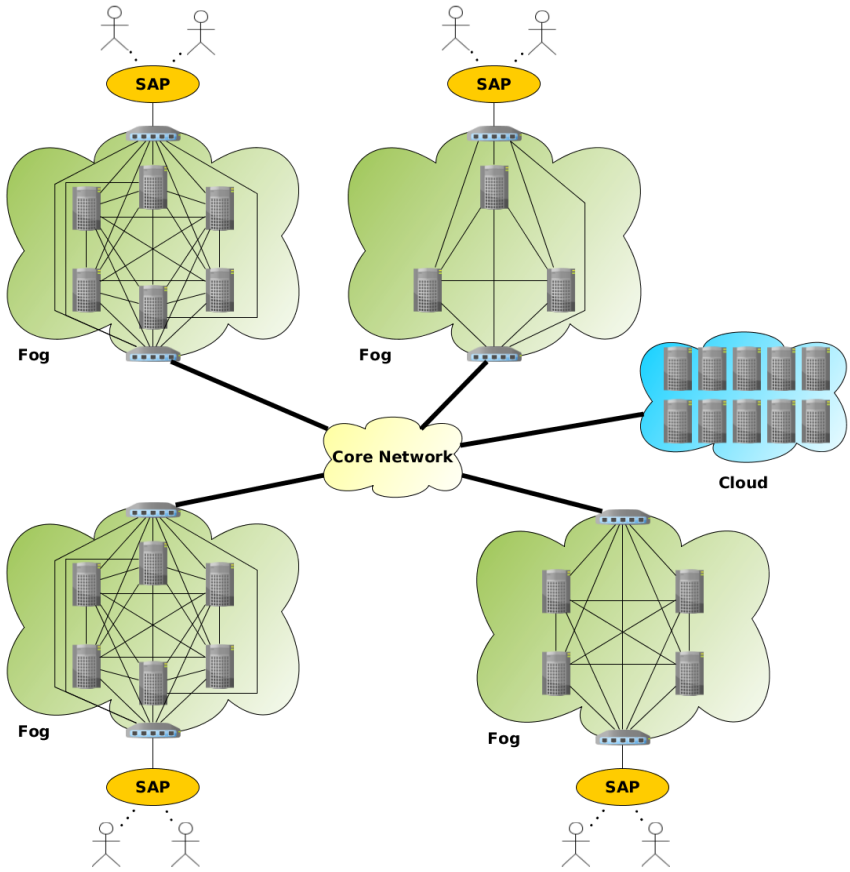
Az NSE problémára egy szemléltető példa a 2. ábrán látható. A szolgáltatás gráf tartalmazza a virtuális hálózati funkciókat, valamint a köztük értelmezett logikai kapcsolatokat. Egy SG több szolgáltatás láncot is tartalmazhat. Az ábrán a narancs és a sárga nyilak által meghatározott utak egy-egy ilyen SFC-t mutatnak be. Az élekhez tartozhatnak például sávszélességre vagy késleltetésre vonatkozó követelmények is. A fizikailag rendelkezésre álló erőforrásokat az erőforrás gráf (angolul: Resource Graph - RG) írja le. Az ábrán megfigyelhető a szolgáltatás gráf elemeinek az erőforrás gráfra történő leképzése is, melynek során például a narancs színnel jelölt szolgáltatás lánchoz tartozó hálózati funkciók futtatására a HOST1, HOST2 és a HOST6 fizikai eszközök lettek kijelölve. A SAP (Service Access Point) csomópontok jelölik a kapcsolatot a külvilág felé, melyen keresztül felhasználók vagy egyéb szolgáltatók kapcsolódni tudnak az infrastruktúránkhoz.



2. ábra. Szolgáltatásgráf, erőforrásgráf és Service Graph Embedding bemutatása

2. Az erőforrásvezénylő probléma

Dolgozatunkban az előző fejezetben bemutatott technológiák együttes alkalmazása esetén felmerülő alapvető kérdésre kerestük a választ, miszerint: Hogyan lehet költséghatékonyan elhelyezni szolgáltatás láncokat heterogén, felhőkből és ködökből álló topológiában? Az általunk elképzelt általános topológia a 3. ábrán látható.



3. ábra. Általános topológia

Tegyük fel, hogy egy szolgáltató cég vagyunk, aki rendelkezik több földrajzilag elosztottan elhelyezkedő ködde, a ködökben bizonyos mennyiségű erőforrással. A ködben található erőforrások használatáért külön nem kell fizetnünk. Ezen felül kapcsolatban állunk egy felhő szolgáltatóval, akinek mi ügyfele vagyunk, tehát „végtelen” mennyiségű erőforrást allokálhatunk a szolgáltatótól, azonban a lefoglalt

erőforrások után fizetnünk kell. A felhasználóink olyan kéréseket fogalmazhatnak meg, melyek során SFC-ket helyezünk el a hálózatunkban teljesítve az azokra meghatározott követelményeket. A hálózatunkban ismerjük az egyes szerverek között található útvonalak késleltetését, a ködök közti maximális sávszélességet, továbbá a fizikai eszközök és a felhő közti késleltetést és maximális sávszélességet.

Mindezek függvényében egy olyan algoritmus megalkotása a célunk, mely a rendelkezésre álló erőforrásokra úgy helyezi el a kérések komponenseit, hogy maximalizáljuk a kérések után a bevételt úgy, hogy minimalizájuk az olyan jellegű kiadások összegét, melyek a migrálásból származó költségekből és a felhő szolgáltatónak fizetendő díjakból állnak. A feladat a Service Graph Embedding [13] problémához hasonlóan a Virtual Network Embedding (VNE) [14] általánosításaként értelmezhető, amely egy $\mathcal{NP}$-nehéz probléma [15]. A feladat matematikai megfogalmazásához szükséges jelölések a 1. táblázatban láthatóak.

1. táblázat. Matematikai jelölések

Jelölés	**Jelentés**
V_s, E_s	Szolgáltatás gráf csúcsai és élei
V_r, E_r	Erőforrás gráf csúcsai és élei
$x_u^i : i \in V_s, u \in V_r$	1 ha az i. NF az u. csomópontra lett leképezve
$y_{u,v}^{i,j} : (i,j) \in E_s, (u,v) \in E_r$	1 ha (u,v)-t tartalmazza az (i,j) SG él fizikai útvonala
$\vec{r_i}$	Az i. NF által megkövetelt erőforrások
$\vec{\rho_j}$	A j. csomópontban rendelkezésre álló erőforrások
$\delta_{u,v}$	az (u,v) fizikai link késleltetése
$d^{i,j}$	Az i. és j. NF között elfogadható maximális késleltetés
$\varrho_s \subset V_s$	SG-ban lévő SAP-ok
$\varrho_r \subset V_r$	RG-ban lévő SAP-ok
$\varrho_s \subseteq \varrho_r$	SG SAP-ok megfeleltethetőek az RG SAP-oknak
c_u^i	i. NF u. node-on való futtatásának költsége
$\gamma \in V_r$	Cloud node végtelen erőforrással

Az általunk használt Network Service Embedding megfogalmazása ILP feladatként a következőkben látható, a magyarázat minden sorhoz alább olvasható.

$$\forall i \in V_s : \sum_{u \in V_r} x_u^i = 1 \tag{1}$$

$$\forall (i,j) \in E_s, \forall u \in V_r : \sum_{v:(u \to v) \in V_r} y_{u,v}^{i,j} - \sum_{w:(w \to u) \in V_r} y_{w,u}^{i,j} = x_u^i - x_u^j \tag{2}$$

$$\forall u \in V_r, \forall i \in V_s : \sum_{i \in V_s} x_u^i \vec{r_i} \leq \vec{\rho_u} \tag{3}$$

$$\forall (i,j) \in E_s : \sum_{(u,v) \in E_r} y_{u,v}^{i,j} \delta_{u,v} \leq d^{i,j} \tag{4}$$

$$\forall (u,v) \in E_r : \sum_{(i,j) \in E_s} y_{u,v}^{i,j} b^{i,j} \leq \beta_{u,v} \tag{5}$$

$$\forall i \in \varrho_s, : x_i^i = 1 \tag{6}$$

$$\min \sum_{u \in V_r} \sum_{i \in V_s} x_u^i c_u^i \tag{7}$$

A leképzés során teljesül, hogy minden VNF pontosan egy fizikai csomóponthoz van hozzárendelve (1). Minden szervererre teljesül, hogy a hozzá bemenő virtuális élek száma megegyezik a belőle kimenő virtuális élek számával(2). Egy fizikai szerverre elhelyezett VNF-ek erőforrásainak összege nem haladja meg a szerver összes fizikai erőforrását (3). Ez hasonlóan elmondható a virtuális linkek leképzésére, azaz azon fizikai linkek sorozata melyre le lett helyezve a virtuális link, teljesíti a virtuális link által megkövetelt késleltetést (4), valamint a megkövetelt sávszélességi igényeket(5). A SG-ben található SAP-ok megfelelnek az RG-ban található SAP-oknak(6). Az optimalizálás célfüggvénye az, hogy minimalizálja a külső féltől igénybevett erőforrásokért fizetendő díjat (7): egy VNF futtatásának költsége a következő(8) függvénnyel adott:

$$c_u^i = \begin{cases} 0, & \text{ha } u \in V_r, u \neq \gamma. \\ F(\vec{r_i}), & \text{különben.} \end{cases} \tag{8}$$

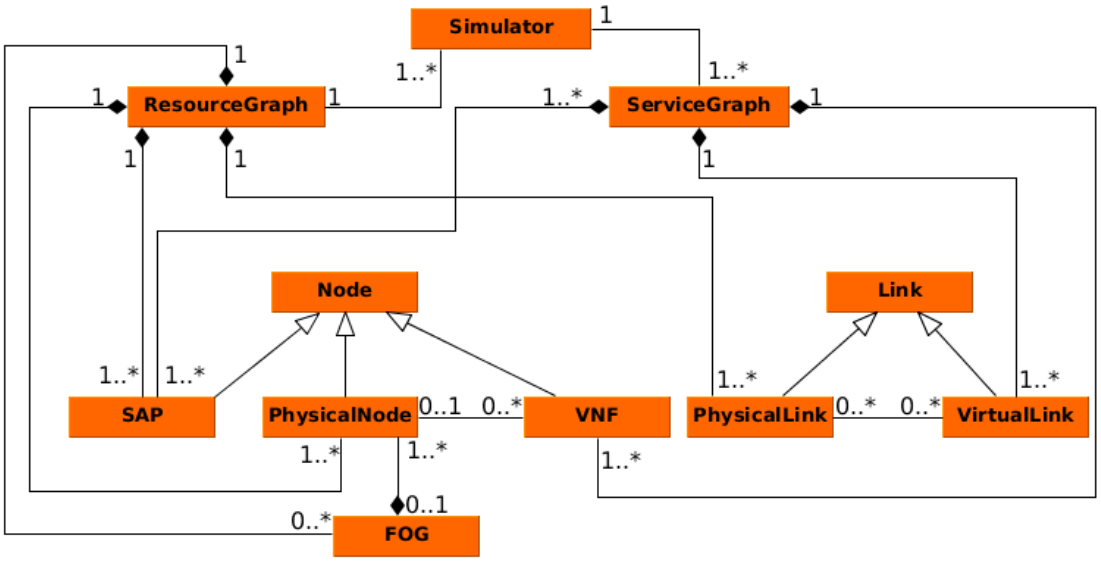
Ez azt jelenti, hogy a VNF futtatása a saját infrastruktúránkat igénybe véve nem jelent többletköltséget, a központi felhőben pedig az igénybevett erőforrásoktól függő díjat kell fizetni érte.

3. A DARK algoritmus

Munkánk első lépéseként egy keretrendszert készítettünk, amelyen képesek vagyunk megvalósítani az elképzelt topológiákat, továbbá a saját implementációnkat használva egyszerűen tudtuk tesztelni rajta az erőforrás-vezénylő algoritmusunkat. Ez utóbbi a futása során szolgáltatás gráfokat dolgoz fel szekvenciálisan a beérkezési sorrendjük szerint. Az algoritmusunk ismertetése során csak a legfontosabb funkciók kerülnek bemutatásra.

3.1. Szimulációs keretrendszer

Az általunk tervezett keretrendszer implementációjának az osztálydiagramja a 4. ábrán látható. A kialakítás során igyekeztünk a lehető legegyszerűbb struktúrával reprezentálni az elképzelt fizikai és virtuális topológiákat.



4. ábra. Az általunk tervezett keretrendszer osztálydiagramja

A modell felépítését az erőforrás gráf és a szolgáltatás gráf megtervezésével kezdtük. Minden erőforrás gráf rendelkezik legalább egy fizikai csomóponttal és egy vagy több SAP-pal. A csúcsok (kivéve a cloud-ot reprezentáló csúcs), és SAP-ok hozzá vannak rendelve ködökhöz, melyek szintén megtalálhatóak az erőforrás gráfban. A

csúcsokat fizikai linkek kötik össze, melyek szintén egy listát képeznek az RG-ben. A szolgáltatás gráfban szintén megtalálhatóak ugyanazon SAP-ok, mint az erőforrás gráfban, továbbá legalább egy VNF, valamint a virtuális linkek listája, melyek az előbb említett virtuális csúcsokat kötik össze. Mivel sok közös tulajdonsággal rendelkeznek a SAP-ok, fizikai és virtuális csomópontok, így létrehoztunk egy közös csúcs osztályt, amiből leszármaztattuk őket. Hasonló a helyzet a linkeknél is.

A modellünket továbbá úgy formáltuk, hogy a szolgáltatás-erőforrás leképzések is jól követhetőek legyenek. Ennek következtében a fizikai csúcsok (és linkek) tárolják a hozzájuk elhelyezett VNF-eket (virtuális linkeket), és a VNF-ek (virtuális linkek) is tárolják, hogy melyik fizikai csomópontra (fizikai linkekre) lettek elhelyezve.

Végezetül pedig készítettünk egy szimulátor programot, melynek ha megadunk egy erőforrás gráfot, szolgáltatás gráfokat, és egy, az elkészített modellel kompatibilis erőforrás-vezénylő algoritmust, akkor szimulálja a működést a megadott elemeken.

3.2. Végrehajtási sorrend kialakítása

A DARK algoritmusunk első lépésként a beérkező kérést - előző komponens, virtuális link, aktuális komponens - hármasok által alkotott listára bontja úgy, hogy a listába bekerülő elemet mindig az elérhető linkek közül a legszigorúbb késleltetéssel rendelkező határozza meg. A végrehajtási sorrend kialakításáért felelős algoritmust az 1. pszeudo kód reprezentálja.

A függvény első lépésben létrehozza azokat a halmazokat, melyekben tárolni fogja a szolgáltatás gráf bejárása során megjegyezni kívánt változók értékeit. Kiinduló pontként felvesszük szolgáltatás gráf első SAP-ját, és betesszük a már meglátogatott csomópontok halmazába, továbbá a SAP-hoz hozzákapcsolt linkeket beteszi az elérhető linkek halmazába. A gráf bejárása úgy történik, hogy a már felvett csomópontok élei közül kiválasztjuk a legszigorúbb késleltetéssel rendelkező, korábban még nem bejárt élt. A kapott él lesz a virtuális link (VLink). A VLink végén lévő csomópont - amelyik már korábban vizsgálva volt - lesz az előző komponens ($previous\_element$), a másik felén lévő csomópont pedig az aktuális komponens ($actual\_element$).

A végrehajtási sorrend kialakításáért felelős algoritmus lépésszáma felülről becsülhető: $O(|E_s|^2)$. A szolgáltatás gráf éleit egyszer járjuk be és előfordulhat, hogy

az egyes élek kiválasztásánál az összes virtuális link szerepel az elérhető linkek halmazában. Erre példa egy csillag topológia, mely egy SAP-pal rendelkezik, és ez a SAP a középpontja a gráfnak.

Algorithm 1 Végrehajtási sorrend kialakítása

```
1: procedure ORDERSUBCHAINS(SG)
2:     mapped_vlinks ← ∅
3:     mapped_vnodes ← ∅
4:     return_list ← ∅
5:     mapped_vnodes.insert(ϱ_s[0])
6:     available_vlinks ← ϱ_s[0].connected_links
7:     while length(mapped_vlinks) ≠ |E_s| do
8:         min_delay_vlink ← GETSTRICTESTDELAYLINK(available_vlinks)
9:         if min_delay_link.node1 ∈mapped_vnodes then
10:            previous_element ← min_delay_link.node1
11:            actual_element ← min_delay_link.node2
12:        else
13:            previous_element ← min_delay_link.node2
14:            actual_element ← min_delay_link.node1
15:        mapped_vnodes.insert(actual_element)
16:        mapped_vlinks.insert(min_delay_link)
17:        min_delay_vlink.is_visited = True
18:        return_list.insert(previous_element,min_delay_link,actual_element)
19:        available_vlinks.insert(actual_element.connected_links)
20:    return return_list
```

3.3. Szolgáltatás gráf leképzése az erőforrás gráfra

Egy szolgáltatás gráf felbontása és sorba rendezése után, az erőforrások leképzése következik a fizikai hálózatra. A leképzés fő metódusa a MAP függvény, mely futásának a leírását a 2. pszeudo kód reprezentálja. A MAP függvény először inicializálja a segédváltozókat, melyek a futás közben szükségesek, majd az 1. pszeudo kódban ismertetett függvény meghívása után megkapja a szolgáltatás gráf végrehajtási

sorrendjét. A kapott listán iterál végig az algoritmusunk.

Minden egyes iterációnál először megvizsgáljuk, hogy az aktuális virtuális link szélein található elemek közül mind a kettőt vizsgáltuk-e már korábban, azaz létezik-e már egy leképzés valamelyik fizikai csomópont és a virtuális elem között. Ez a leképzés egy SAP esetében egyértelmű, ugyanis az SG-ben lévő SAP-ok megfeleltethetőek az RG-ben lévő SAP-oknak ($\varrho_s \subseteq \varrho_r$). Egy VNF esetében a leképzés azt jelenti, hogy a VNF-hez hozzárendelünk egy fizikai szervert, amelyen elhelyezzük a VNF-et a DARK futása végén, ha a szolgáltatás minden része sikeresen elhelyezhető. Ha ez a feltétel teljesül, akkor csak a a virtuális linket kell elhelyezni fizikai linkek egy adott sorozatára. Egy VLink leképzése úgy történik, hogy a szélein található SAP-ok vagy VNF-ek erőforrás gráfbeli megfelelőjük között megkeressük a késleltetés szerinti legrövidebb utat, és az út minden fizikai élére lehelyezzük a virtuális linket. A legrövidebb út megkereséséhez Dijkstra algoritmust használunk az erőforrás gráfon.

Ha a jelenlegi VLink egyik végpontja még nincs benne a vizsgált csomópontok halmazában, akkor az algoritmusunk megvizsgálja, hogy az aktuális elem VNF vagy SAP. Ha VNF, akkor a MAPVNF függvény használatával megpróbáljuk elhelyezni a fizikai topológián. A MAPVNF metóduson belül a program megkeresi azokat a fizikai csomópontokat, melyek rendelkeznek annyi szabad erőforrással, amennyi elég a VNF számára, majd ezen csomópontok halmazát tovább szűkíti a hálózati követelmények kompatibilitásának vizsgálatával. Ez azt jelenti, hogy megvizsgálja, hogy az aktuális iterációban vett előző elemhez rendelt fizikai szerver és a jelenlegi kompatibilis szerverek között található-e olyan útvonal, mely mind a késleltetési követelményeket, mind a sávszélesség által megszabott követelményeket teljesíti. Amennyiben egy szerver nem felel meg valamelyik követelménynek, úgy az algoritmus kiveszi a kompatibilis csomópontok sorozatából.

Ha a szűrést követően legalább egy szerver kompatibilis, akkor a program sorba rendezi bizonyos szempont szerint a csomópontokat, és kiválasztja a sorrendezési szempont szerinti legmegfelelőbb csomópontot. Ha megvan a kiválasztott szerver, akkor leképezzük a virtuális linket is, a korábban bemutatott módon. Amennyiben sikeres volt a virtuális hálózati funkció és a VLink elhelyezése, az algoritmus tovább lép a következő iterációra.

Algorithm 2 Szolgáltatás gráf leképzése az erőforrás gráfra

```
1: procedure MAP(SG,RG)
2:     running ← copy(RG)
3:     mapped_vnodes ← ∅
4:     rollback_level = 0
5:     map_list ←ORDERSUBCHAINS(SG)
6:     mapped_vnodes.insert(ϱ_s[0])
7:     for all (previous_element,min_delay_link, actual_element) ∈map_list do
8:         if min_delay_link.node1 and min_delay_link.node2 ∈mapped_vnodes then
9:             success ← MAPVIRTUALLINK(min_delay_link)
10:        else
11:            if actual_element ∈ V_s and actual_element∉ ϱ_s then
12:                success←MAPVNF(previous_element,min_delay_link, actual_element)
13:            else                                  ▷ This means actual_element is a SAP
14:                success ← MAPVIRTUALLINKTOSAP(previous_element, min_delay_link, actual_element)
15:            if ¬ success then
16:                if rollback_level >= rollback_allowed then
17:                    success← MIGRATINGEDGE2CORE(corable_vnfs, previous_element,actual_element)
18:                    rollback_level = 0
19:                else
20:                    success← ROLLBACK(previous_element, min_delay_link,actual_element)
21:                    rollback_level+ = 1
22:        if success then
23:            mapped_vnodes.insert(actual_element)
    done
```

Ha az aktuális elem egy SAP, akkor a program a VLink elhelyezése előtt megha-

tározza az előző elem és a SAP közötti legrövidebb (legkisebb késleltetéssel rendelkező) utat. Amenyibben ez kielégíti a közöttük húzódó virtuális link által megkövetelt késleltetési és sávszélességi feltételt, úgy leképezzük a virtuális linket a korábban bemutatott módon.

Előfordul olyan eset, hogy az előző lépések valamelyike nem fut le sikeresen. Például nem található kompatibilis fizikai szerver, vagy a virtuális link követelményei nem teljesülnek. Ilyen esetekben először az algoritmus visszalép egy előző állapotra, és azt megváltoztatva próbálja újra elhelyezni a sikertelen komponenst. Ezt a visszalépést és korábbi állapot megváltoztatását a ROLLBACK metódus hajtja végre. Ezt csak bizonyos mélységig engedélyezzük, ugyanis ellenkező esetben hosszú és bonyolult szolgáltatások esetén a futási időt túlságosan megnövelné, amit nem engedhetünk meg. Amenyiben még van lehetőség a ROLLBACK végrehajtására, úgy a program megkeresi azt az állapotot, melyben a mostani iterációban lévő előző elemet elhelyeztük, és a kiválasztott fizikai csomópontot megváltoztatjuk úgy, hogy a virtuális elemhez tartozó kompatibilis szerverek listájából a következőt választjuk. Majd a visszalépett állapottól folytatva próbáljuk újra a szolgáltatás gráf leképzését a már újonnan kiválasztott fizikai szerverrel.

Abban az esetben, ha már nem lehetséges tovább futtatni a ROLLBACK függvényt, az algoritmus megpróbál migrálni egy vagy több hálózati funkciót, ezzel lehetővé téve az aktuális komponens elhelyezését. A migrálás folyamatának bemutatása a 3.4 alfejezetben látható.

3.4. VNF migrálás fog-ból cloud-ba

Az migrálási funkcióért, vagyis a már leképzett VNF-ek mozgatásáért a 3. pszeudo kódban leírt MIGRATINGEDGE2CORE metódus felel. Ez a függvény három paramétert kap bemenetként, melyek az alábbiak:

- *coreable_vnfs* - Azon VNF-ek listája, melyek már egy ködben lévő szerveren futnak, de követelményeik alapján a publikus felhőben is futtathatóak lennének.
- *actual_vnf* - Éppen leképzendő VNF.
- *previous_vnf* - Az előzőleg leképzett VNF.

Algorithm 3 VNF migrálás fog-ból cloud-ba

```
1: procedure MIGRATINGEDGE2CORE(corable_vnfs, previous_vnf, actual_vnf)
2:     compatible_fogs ← ∅
3:     fog_list ← GETFOGSFROMVNFS(corable_vnfs)
4:     for fog ∈ fog_list do
5:         if fog == previous_vnf.fog then
6:             compatible_fogs.insert(fog)
7:         else
8:             vlink ← GETVLINK(previous_vnf, actual_vnf)
9:             phy_path ← GETPATH(previous_fog, fog)
10:            if GETBW(phy_path) ≥ vlink.bw and
                  GETDELAYPATH(phy_path) ≤ vlink.delay then
11:                compatible_fogs.insert(fog)
12:    corable_vnfs ← DELUNCOMPVNFS(corable_vnfs,compatible_fogs)
13:    possible_nodes ← GETPHYNODESFROMVNFS(corable_vnfs)
14:    migration_list ← ordered empty list
15:    for node ∈ possible_nodes do
16:        vnf_list ← ordered empty list
17:        for vnf ∈ node.corable_vnfs do
18:            if ISMIGRABLE(vnf) then
19:                vnf.mig_cost ← GETMIGCOST(vnf)
20:                migration_list.add(vnf)
21:        migration_list ← EXPANDMIGLIST(node.corable_vnf_list)
22:    for mig_opt ∈ migration_list do
23:        DOMIGRATING(mig_opt)
24:        if ISMIGWASSUC(mig_opt) then
25:            return True
26:        else
27:            RESTOREMIG(mig_opt)
28:    return False
```

A függvény három részre bontható. Az első harmada (11. sorig) az olyan ködök kiválasztásával foglalkozik, melyekből sikeresen mozgathatunk VNF-eket úgy, hogy az újonnan leképzendő virtuális funkció összeköttetés követelményei (késleltetés és sávszélesség) ne sérüljenek. A második rész (12. és 21. sorok között) a megfelelő ködökből való lehetséges migrálási lehetőségeket határozza meg. Végeredményképpen egy listák listája objektumot ad, hiszen előfordulhat olyan eset is, hogy egy fizikai szerverről akár több VNF-et kell elmozgatni, hogy az aktuálisat képesek legyünk leképezni. Végül a függvény harmadik része (22. sortól) a migrálási opciókat próbálja ki sorban és amint képes volt egy sikeres mozgatásra igaz feltétellel tér vissza.

3.4.1. Kompatibilis ködök kiválasztása

A GETFOGSFROMVNFS metódus adja meg számunkra, hogy mely ködök tartalmaznak a felhőben is megvalósítható VNF-eket, hiszen csak ezeket van értelme a továbbiakban vizsgálnunk a migrálhatóság szempontjából. Ha ezek között az is szerepel, ahol az előzőleg leképzett VNF található, akkor az biztos, hogy megfelelő lesz számunkra, hiszen így az aktuális VNF és az előző VNF közötti késleltetési és sávszélességi igény szinte biztos, hogy teljesül. Ezt a vizsgálatot a kódban az ötödik és a hatodik sor végzi el.

A többi köd esetén azt ellenőrizzük, hogy a fog és a gerinchálózat (1.ábrán a Core Network) közötti összeköttetés megfelel-e a követelményeknek. A nyolcadik sorban látható *VLink* változó tartalmazza azt a virtuális összeköttetést és követelményeit, amely az aktuális és az előző VNF között található. A *phy_path*-ban tároljuk el azon fizikai linkek sorozatát, mely az előző VNF-et tartalmazó és az aktuálisan vizsgált köd között találhatók. Ha ez az útvonal megfelel a link követelményeinek, akkor elmentjük a vizsgált fog-ot a kompatibilisek közé.

3.4.2. Migrálási lehetőségek összegyűjtése

Első lépésként a felhőben is megvalósítható VNF-ek listáját csökkentjük annak érdekében, hogy a nem kompatibilis ködök szerverei ne szerepeljenek benne. Erről a DELUNCOMPVNFS függvény gondoskodik. Ezek után a *possible_nodes* listába elmentjük azon szervereket amelyek tartalmaznak migrálható funkciókat. Az olyan VNF-eket, melyeket elmozgatva a helyükre kerülhetne az aktuális hálózati funkció,

a migrálási költség kiszámolása után elmentjük a végeredményt adó *migration_list*-be. E feltétel teljesülését az ISMIGRABLE metódus végzi. Fontos megemlítenünk, hogy ez a lista a migrálási költségek szerint rendezve van, vagyis minden beszúrás is rendezetten történik.

Eddig a VNF-eket csak egyesével vizsgáltuk, azonban előfordulhat olyan eset is, hogy egyszerre több hálózati funkció elmozdításával, az azokat tartalmazó fizikai szerver is megfelelő lehet az aktuálisan leképzendő VNF számára. Emiatt végül a szerveren lévő migrálható VNF-eket párosával, hármasával és így tovább növekedve is célszerű lehet ellenőrizni. Ez lényegében azt jelenti, hogy a felhő kompatibilis VNF-ek halmazának az összes részhalmazát kellene meghatározni és ezeket a részhalmazokat vizsgálni, hogy megfelelnek-e a követelményeknek. A futási időt figyelembe véve mi csak a listában szomszédos elemeket - mint részhalmazokat - vizsgáltuk. Amennyiben ezen részhalmazok megfelelnek a feltételeknek, akkor hozzáadjuk a *migration_list* változóhoz. Az ilyen részhalmazok keresését az EXPANDMIGLIST metódus végzi el. Az egy szerverre vonatkozó *migration_list* összeállítása $O(|V_s|(|V_s|+1)/2)$ lépésszámot követel, ahol $|V_s|$ a szerveren futtatott VNF-ek száma. Végeredményképpen megkaptuk a lehetséges migrálási kombinációk költség szerinti rendezett listáját.

3.4.3. Migrálások végrehajtása

Végül a migrálási listán egyre beljebb - drágább migrálások felé - haladva folyamatosan próbálkozunk a VNF-ek mozgatásával. Ezt végzi a DOMIGRATING függvény. Minden lépés során az ISMIGRATING metódus segítségével ellenőrizzük a migrálás sikerességét. Ez azt jelenti, hogy az átmozgatott VNF(ek)-hez tartozó virtuális linkek követelményeit hasonlítjuk össze az azokat megvalósító fizikai linkek tulajdonságával. Ha a migrálás sikeres volt, akkor a 3. algoritmus visszatérési értéke igaz lesz. Ellenkező esetben visszaállítjuk a migrálás előtti állapotra a szerverek és linkek tulajdonságait (a RESTOREMIG függvény segítségével) és próbálkozunk a következő migrálási lehetőséggel.

4. A DARK hatékonyságának mérése

Ebben a fejezetben mutatjuk be az elkészített DARK algoritmusunk teljesítményére vonatkozóan kapott eredményeinket. Mivel a fog computing a kiépítést illetően még gyerekcipőben jár, így tényleges hálózat nem állt rendelkezésünkre a tesztelés során. Emiatt létre kellett hoznunk a saját szimulátorunkat ahol a teszteket futtathattuk. Először különböző topológiákat kellett létrehoznunk, pontosabban olyan modelleket, melyek a fizikai hálózatot írták le. A következő kihívás a különböző típusú követelményekkel rendelkező, de mégis realisztikus szolgáltatásokat leíró kérések generálása volt.

Az erőforrásgráf és a szolgáltatás láncok elkészítése után az algoritmusunk által adott megoldásokat hasonlítottuk össze az optimális megoldás eredményével. A különböző esetekben, a megvalósított kérések által leképzett CPU-k számát és az üzemeltető költségeit vizsgáltuk. Fontos megjegyezni, hogy az algoritmusunk jelen állpotban csak egy típusú migrálásra képes: a ködökben található szerverekről helyez át futó funkciókat a felhőbe. Az szimuláció során használt költségeket a (9),(10),(11) képletek reprezentálják. Az olyan jelölések jelentéseit, melyek megtalálhatóak ezekben a formális leírásokban, de nem szerepelnek a 1. táblázatban, a 2. táblázat tartalmazza.

2. táblázat. Matematikai jelölések kiegészítése

Jelölés	**Jelentés**
m_i	Migrálási költség
α	Migrálási együttható
$CPU_i \subset \vec{r_i}$	i. NF által megkövetelt CPU-k száma

$$F(\vec{r_i}) = CPU_i \tag{9}$$

$$m_i = \begin{cases} 1, & \text{ha } x_u^i \leftarrow x_j^i,\ u = \gamma. \\ 0, & \text{különben.} \end{cases} \tag{10}$$

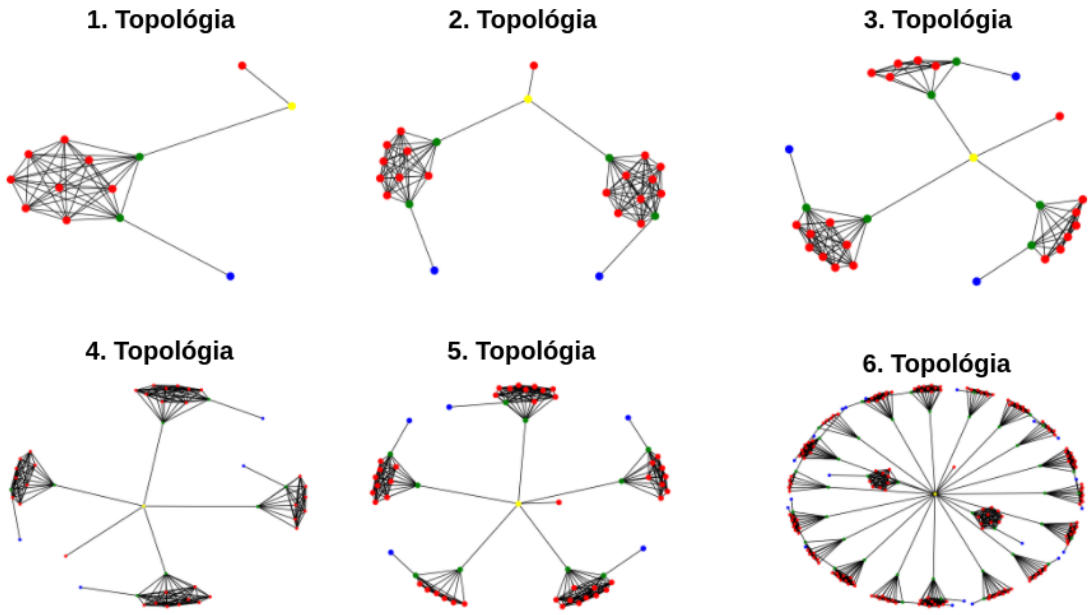
$$c_u^i = \begin{cases} 0, & \text{ha } u \in V_r, u \neq \gamma. \\ F(\vec{r_i}) + m_i * \alpha, & \text{különben.} \end{cases} \tag{11}$$

Dolgozatunk során a 9. képlet alapján számoltuk a leképzések költségeit, ahol úgy vettük, hogy a felhőbe elhelyezett VNF-ek erőforrás foglalási költsége megegyezik az általuk használt CPU-k számával. A migrálási költséget pedig minden esetben konstans értékkel számoltuk, mely 1, amenyibben az adott VNF-et átmigráltuk a felhőbe, különben 0 (10). Egy virtuális hálózati funkció teljes költsége a szimuláció végén az üzemeltetőnek fizetett költség (9) összeadva a migrálási költség (10) és a migrálási együttható (mely az egyes szimulációk alkalmával szintén konstans) szorzatával (11). A részletes eredményeket a 4.3. fejezetben mutatjuk be.

4.1. Hálózati topológiák generálása

Munkánk során összesen hat darab topológiát generáltunk annak érdekében, hogy a kapott leképzési eredményeket több esetben is összehasonlíthassuk. Ahogy már azt az 1.1 fejezetben ismertettük, többféle réteg van jelen egy tipikus köd hálózatban. Legalul helyezkednek el az IoT-ben ismert okos eszközök. Ezek olyan szenzorok, beágyazott rendszerek vagy egyéb intelligens eszközök, melyek vezetékkel vagy vezeték nélkül csatlakoznak a Fog réteghez. Ebben a rétegben egy vagy több úgynevezett köd, vagy angolul fog helyezkedik el, amelyen belül találhatóak az egy adott fizikai területért felelős szerverek. Ez felfogható úgy is, mint a központi felhő adatközpont szélekre kiszervezett része. Ennek célja, hogy a ködben lévő szerverek és a lefedett területen lévő okos eszközök között a lehető legkisebb késleltetést biztosítsanak. Ennek köszönhetően a kívánt szolgáltatás azon részeit melyeknek fontos, hogy a végberendezés közelében legyenek (például kritérium a kicsi késleltetés) a végberendezés tartózkodási helyéhez tartozó ködben valósíthajuk meg.

Egy hasonlóan kiépített rendszert elképzelve készítettük el saját topológia modelljeinket, melyeknek grafikus ábrázolásuk az 5. ábrán látható. A kék csomópontok jelentik a topológia SAP-jait, vagyis a ki és belépési pontokat. Ezeken keresztül tudnak csatlakozni az eszközök a hálózatra. Minden SAP egy ködhöz csatlakozik. Egy ködön belül véletlenszerűen kiválasztva öt és tíz közötti szerver (ábrán piros csomópontok) helyezkedik el, illetve két darab router (ábrán zöld csomópontok). Az



5. ábra. Elkészített topológiák

ezek közötti linkeket úgy helyeztük el, hogy az egy ködön belüli csomópontok teljes gráfot alkossanak. A fog-okon belül feltételeztük, hogy a sávszélesség nem lehet szűk keresztmetszet, mivel itt néhány szerverről van csak szó amit többféle módszerrel is hatékonyan össze lehet kötni (például leaf-spine, fat-tree, full-mesh és egyéb architektúrák).

Minden egyes köd csatlakozik a gerinchálózathoz, amit az ábrán a sárga csomópont jelez. Ezen keresztül érhetjük el a felhő adatközpontot, ami alatt olyan publikus felhőrendszert értünk mint például az AWS, Azure, Upcloud vagy a DigitalOcean. Mivel ezen felhőszolgáltatók esetében hatalmas adatközpontokról beszélünk, így úgy feltételeztük, hogy a modellünkben megvalósított felhő csomópontunkban végtelen processzor, memória és tárhely helyezkedik el. Az egyes szerverekhez és linkekhez a 3. és a 4. táblázatokban szereplő adatok alapján választottuk ki a számítási és hálózati tulajdonságok értékeit.

3. táblázat. Modellezett szerverek erőforrásai

	CPU [db]	RAM [GB]	Háttértár [GB]
Ködben található szerverek	24 - 30	64 - 256	5120 - 81920
Felhő	999999	999999	999999

4. táblázat. Modellezett összeköttetések hálózati erőforrásai

	Sávszélesség [Gbps]	Késleltetés [ms]
Köd és gerinchálózat közti link	40	10 - 15
Gerinchálózat és felhő közti link	100	50 - 100

4.2. Szolgáltatás igények generálása

Annak érdekében, hogy teljeskörű teszteket hajthassunk végre, szükség volt a topológiákra leképzendő szolgáltatás kérések generálására is. Ezért minden egyes topológiához több kéréssorozatot gyártottunk, hogy az egyes működési hibákat statisztikai alapokon is képesek legyünk kiszűrni. A létrehozott szolgáltatás igények a következőképpen néztek ki. Minden kérés tartalmazott:

- A topológiában szereplő, véletlenszerűen kiválasztott SAP-ot.
- Egy vagy több a fog réteg karakterisztikájának megfelelő VNF-et
- Egy vagy több a cloud réteg karakterisztikájának megfelelő VNF-et

Mivel a fog computing egy teljesen új megközelítés, így konkrét adatokkal nem rendelkezünk a valós életből vett példákon keresztül az egyes VNF-ek erőforrás és hálózati követelményeit illetően. Azonban az elképzelés szerint a köd rétegben olyan VNF-ek fognak elhelyezkedni, melyeknek legfontosabb ismérve a gyors reagálás, vagyis kicsi késleltetést engednek meg a felhasználó és a megvalósított hálózati funkció között. Ezzel ellentétben a cloud-ban olyan VNF-eket célszerű futtatni, melyek magas sávszélességet, nagy tárhelyet és magas számítási kapacitást igényelnek. Ide kerülhetnek például az analitika, menedzsment és üzleti intelligenciát megvalósító alkalmazások. A generált szolgáltatás komponenseinek tipikus számítási és hálózati követelményei a 5. és az 6. táblázatban szereplő adatok alapján lettek elkészítve.

5. táblázat. Generált szolgáltatás kérések számítási erőforrás követelményei

	CPU [db]	RAM [GB]	Háttértár [GB]
Köd VNF	1 - 4	1 - 8	10 - 100
Felhő VNF	4 - 10	4 - 16	100 - 5000

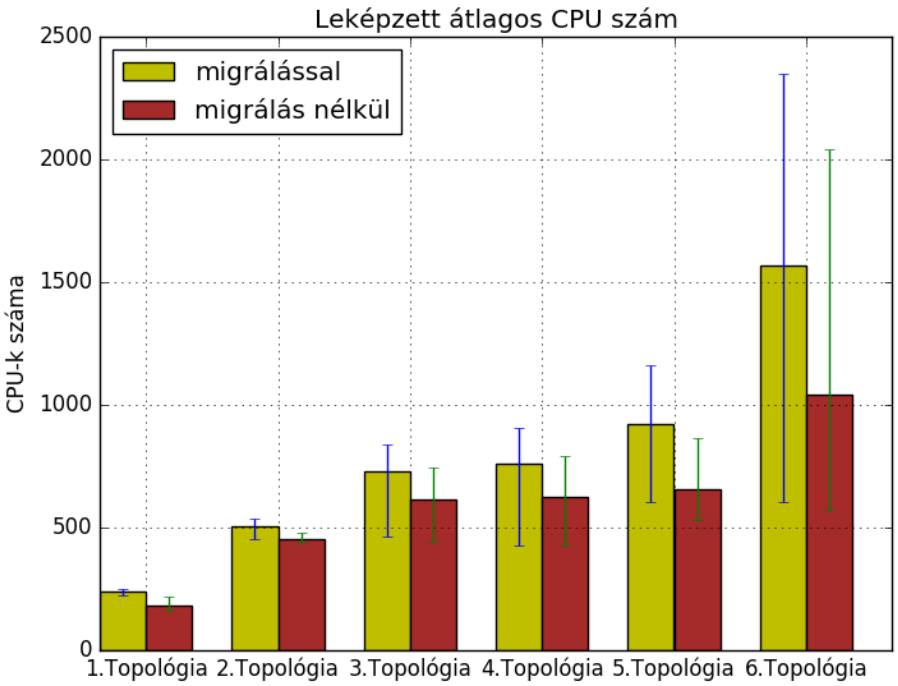
6. táblázat. Generált szolgáltatás kérések hálózati erőforrás követelményei

	Késleltetés [ms]	Sávszélesség [Mbps]
Köd link	5 - 20	100 - 1000
Felhő link	50 - 1000	1000 - 5000

4.3. Szimulációs eredmények

A fentebb bemutatott minden egyes topológiánkra több kéréssorozatot generáltunk, hogy egy átfogó képet kaphassunk a DARK algoritmusunk működéséről. A szimuláció során a kéréssorozatban szereplő igényeket egymás után küldtük be az erőforrás-vezénylő algoritmusunknak egészen addig, amíg el nem értük az első olyan kérést, amelyet az algoritmus már nem tudott leképezni az erőforrás gráfra. A létrehozott programunkat kétféle működési móddal használtuk, a migrálási funkció (vagyis a már leképzett VNF-ek mozgatása) tiltásával és engedélyezésével.

Könnyen belátható, hogy a migrálási funkció amiatt szükséges, hogy költségtakarékosan tudjuk a kívánt kéréseket leképezni. Ha ezt kikapcsoljuk, az algoritmusnak két lehetősége van. Egyik, hogy azon VNF-eket, amelyeket a felhőbe is leképezhetünk, oda is helyezi el. Ez egy működő megoldás, ellenben nem költséghatékony, mivel a harmadik fél által üzemeltett felhő adatközpontba tett virtuális funkciókért fizetnünk kell. Másik lehetőség, hogy ezeknek a VNF-eknek a felhő helyett egy olyan ködben keresünk helyet, amely megfelel a felé támasztott számítási és hálózati követelményeknek. Ekkor ugyan sikeresen spóroltunk a felhő költségein, azonban elfoglaltuk az erőforrásokat egy olyan esetleges jövőbeli kérés elől, melynek egy vagy több VNF-jét is csak az adott fog-ban lehetne megvalósítani, például egy késleltetés követelmény miatt.

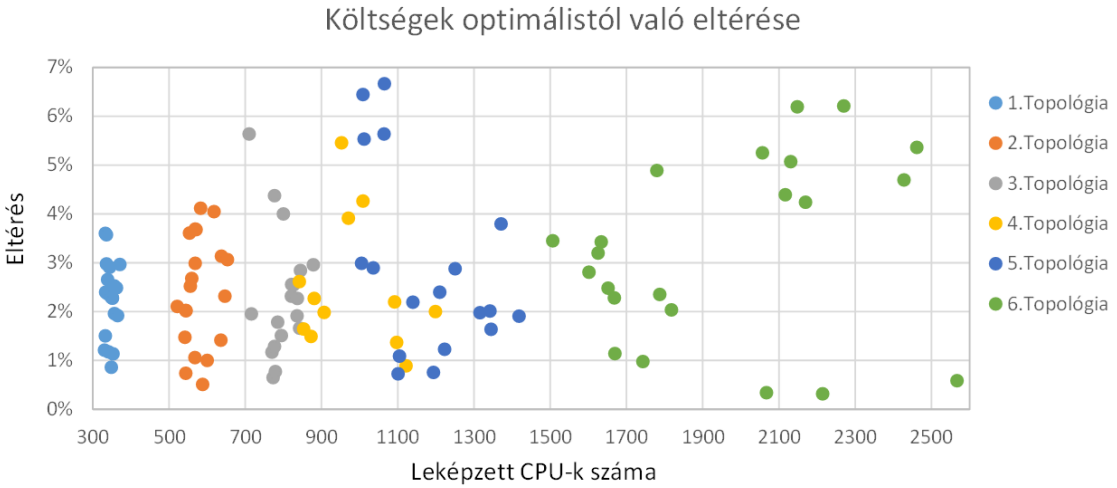


6. ábra. Leképzett szolgáltatások átlagos CPU száma különböző szcenáriókra

A 6. ábrán láthatjuk, az algoritmusunk által leképzett átlagos virtuális CPU (VNF-ek által igényelt processzor) számot migrálás engedélyezésével és nélküle: a hat topológiára az átlag eredmények (bal oldali oszlop a migrálásos, a jobb oldali a migrálás nélküli minden esetben) mellett az oszlopok közepén látható kék és zöld hibasávok mutatják a kéréssorozatok közötti minimális és maximális leképzett CPU számot. Fontos megemlítenünk, hogy a mért adatokon meghatároztuk az alsó és felső percentiliseket (5% - 95%) és az átlagszámításba, illetve a minimum és maximum meghatározásába, csak az ezek közötti adatokat használtuk referencia értékekként.

A továbbiakban kíváncsiak voltunk, hogy az elkészített algoritmusunk milyen mértékben képes az optimális megoldást megközelíteni a beérkező kérések leképzésekor. Ahhoz, hogy erre a kérdésre választ kapjunk, a hivatkozott [13] cikkben is használt egészértékű programozás alapú algoritmus eredményeit használtuk fel összehasonlítási alapként. Bemenetként ugyanazokat a kéréssorozatokat használtuk, mint a saját algoritmusunk esetén is, és a kapott optimális eredményeket vetettük

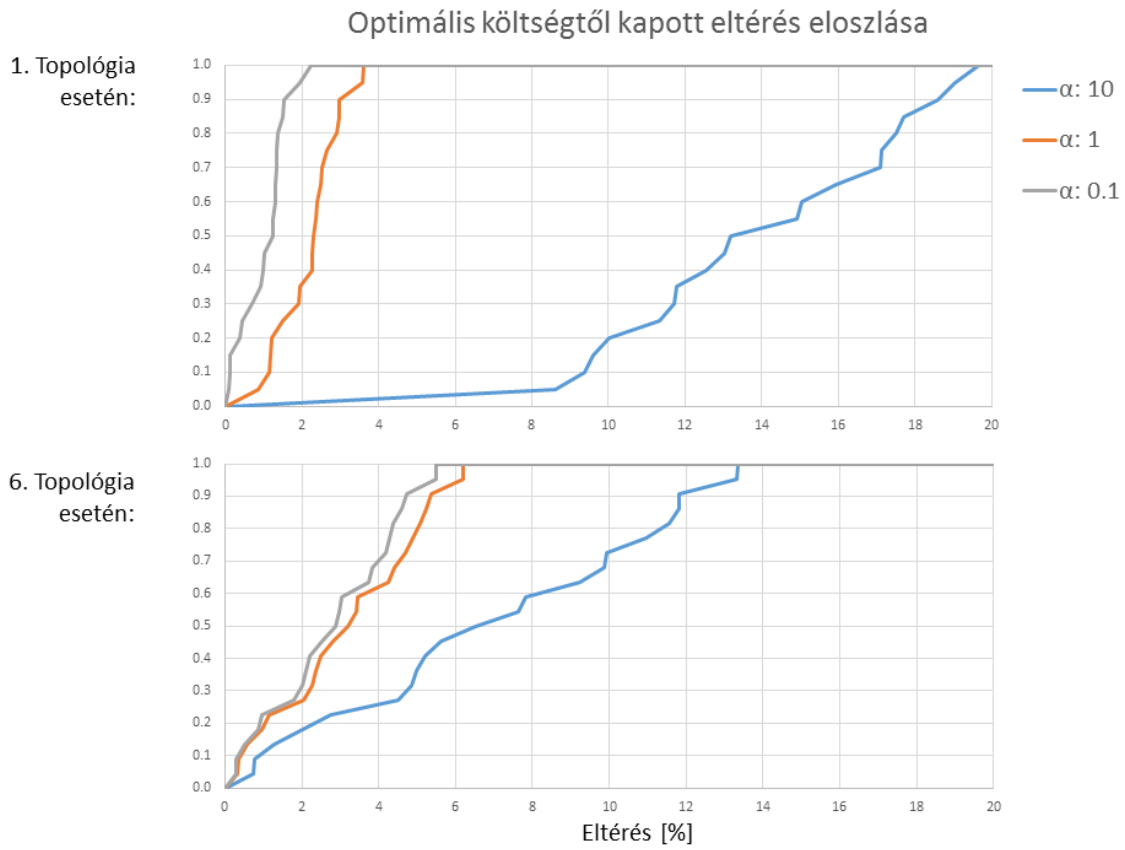
össze a DARK megoldásaival. Az összehasonlítást a költségek szempontjából tettük meg. Optimális megoldás esetén ezt az értéket úgy kaptuk meg, hogy megvizsgáltuk a publikus felhőben futó VNF-ek által használt CPU-k számát, mivel a tudományos dolgozatunk során azt feltételeztük, hogy ez a paraméter határozza meg a harmadik fél által futtatott VNF-ek árát.



7. ábra. Optimumtól való eltérés

A kapott különbségek a 7. ábrán láthatóak. Az optimális megoldástól való százalékos eltérést jelenítettük meg különböző topológiák esetén, az összesen leképzett virtuális CPU-k számának függvényében. A diagrammon minden pont egy kérésssorozatra kapott DARK vs. ILP eredményt jelent. Kiemelendő, hogy az algoritmusunk által adott költségek maximum körülbelül 7%-kal tértek el az optimális megoldástól. Továbbá az is szembetűnő, hogy a különböző topológiákon mekkora hatása van a véletlenszerűen generált kéréssorozatoknak a leképzett VNF-ek összesített CPU használatát illetően. Amíg például egy és két köd esetén (világoskék és narancssárga pontok) ez az érték számottevően nem változik, addig húsz köd esetén (ábrán zöld szín) körülbelül ezer CPU különbség található a legrosszabb és a legjobb eset között. Végül a legfontosabb jellemző amit megállapíthatunk az az, hogy skálázódás szempontjából is helyesen működik a megalkotott algoritmusunk, hiszen ahogy növekednek a ködök - és ezáltal a virtualizáló szerverek - száma, továbbá növekedik az elhelyezett virtuális szolgáltatások száma, az optimális költségtől való eltérés

jelentősen nem változik a DARK algoritmust használva.



8. ábra. Optimumtól való eltérés

A 8. ábrán az algoritmusunk által adott költség kimenet eloszlását láthatjuk a topológiák szerinti két szélsőséges esetben (egy és húsz köd). Az X tengelyen az általunk fejlesztett DARK algoritmus kimenetének költsége és az optimális algoritmus által adott eredmény költsége közötti eltérést jelenítjük meg százalékosan. Fontos különbség a két program között, hogy míg az optimális esetben a költséget csak a felhőben futtatott CPU-k száma alapján számoljuk, addig a DARK algoritmus esetén figyelembe kell venni a már leképzett VNF-ek migrálásából adódó költségeket is, amit a (11) képletben az $m_i\alpha$ értéke fejez ki.

Érdekes kérdés, hogy hogyan változik az optimális költségtől való eltérés eloszlása az α paraméter függvényében. Ha túlságosan nagyra vesszük, akkor jelentősen megnövekedhetnek a költségeink, azonban $\alpha = 1$ és $\alpha = 0.1$ esetén nincsen jelentős, csak

körülbelül 1%-os különbség. Ebből arra következtethetünk, hogy ha a migrálással kapcsolatos költségek rendkívül magasak (egy nagyságrenddel költségesebbek, mint a felhőben való futtatás), akkor hatékonyabb lehet az a működés, ahol a migrálás helyett egyenesen a felhőben hozzuk létre a hálózati funkciókat. Így elkerüljük a jövőben adódó lehetséges migrálások által adott plusz költségeket.

5. Konklúzió

A dolgozatunk során megismerkedtünk egy jövőbeli heterogén köd-felhő környezetben felmerülő problémával, mely az olyan szolgáltatók számára lesz fontos, akik új köd-alapú számítástechnikai rendszerben kívánnak szolgáltatásokat nyújtani. A probléma mind technológiai, mind pénzügyi szempontból releváns, hiszen a költségeket próbálja minimalizálni olyan szolgáltatások nyújtása kapcsán, melyek a legújabb technológiákat felhasználva működnek. Létrehoztunk egy olyan polinomiális lépésszámú algoritmust, mely a fentebb bevezetett probléma optimális megoldásától kis mértékben tér el. A szimulációink során megállapítottuk, hogy a megoldásunk jól skálázódik, hiszen nagy számú lerakott virtuális CPU és több köd esetén is megtartja az optimálishoz közeli eredményeket.

Hivatkozások

[1] N. Panwar, S. Sharma, and A. K. Singh, „A Survey on 5G," *Phys. Commun.*, vol. 18, no. P2, pp. 64–84, Mar. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.phycom.2015.10.006

[2] Q. L. Shanhe Yi, Cheng Li, „A survey of fog computing: Concepts, applications and issues," in *Mobildata '15, ACM.* ACM, 2015, pp. 37–42.

[3] G. Brown, „Mobil edge computing use cases and deployment options," Tech. Rep., 2016.03.01. [Online]. Available: https://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000642-en.pdf

[4] Z. B. Pavel Mach, „Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[5] C. C. Byers, „Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 14–20, 2017.

[6] L. R.-M. Luis M. Vaquero, „Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[7] R. B. Amir Vahid Dastjerdi, „Fog computing: Helping the internet of things realize its potential," *IEEE Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[8] J. Z. S. A. Flavio Bonomi, Rodolfo Milito, „Fog computing and its role in the internet of things," *MCC '12*, pp. 13–16, 2012.

[9] O. Consortium, „Openfog reference architecture for fog computing," Tech. Rep., 2017.02.08. [Online]. Available: https://www.openfogconsortium.org/wp-content/uploads/OpenFog_Reference_Architecture_2_09_17-FINAL.pdf

[10] ETSI, „White Paper: Network Functions Virtualisation (NFV)," 2013. [Online]. Available: http://portal.etsi.org/nfv/nfv_white_paper2.pdf

[11] ETSI GS NFV-PER 001, Dec 2014, *Network Functions Virtualisation (NFV); Architectural Framework*, Mentve:http://www.etsi.org/deliver/etsi_gs/NFV-PER/001_099/001/01.01.02_60/gs_NFV-PER001v010102p.pdf, ETSI, 2014 Dec.

[12] J. Halpern and C. Pignataro, „Service Function Chaining (SFC) Architecture," IETF RFC 7665, Oct. 2015.

[13] B. Németh, B. Sonkoly, M. Rost, and S. Schmid, „Efficient Service Graph Embedding: A Practical Approach," in *Second IEEE International Workshop on Orchestration for Software Defined Infrastructures (O4SDI @ IEEE NFV-SDN 2016)*, Nov 2016.

[14] M. T. B. H. d. M. Andreas Fischer, Juan Felipe Botero and X. Hesselbach, „Virtual network embedding: A survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[15] E. Amaldi *et al.*, „On the computational complexity of the virtual network embedding problem," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 213 – 220, 2016, INOC 2015 – 7th International Network Optimization Conference.