Kovács Ákos

Felhő alapú infrastruktúrák kiegészítő szolgáltatásainak vizsgálata

doktori értekezés

Témavezető:

Prof. habil Lencse Gábor

Távközlési Tanszék, Széchenyi István Egyetem

Széchenyi István Egyetem

Infrastrukturális Rendszerek Modellezése és Fejlesztése

Multidiszciplináris Műszaki Tudományi Doktori Iskola

Tartalomjegyzék

1 Bevezetés 4

1.1.1 Igény szerinti önkiszolgálás 6

1.1.2 Nagyteljesítményű hálózat 6

1.1.3 Erőforrás készletek 6

1.1.4 Rugalmasság 7

1.1.5 Mérhető adatok 7

1.2 A kutatási téma és annak időszerűsége, a kutatási téma fontossága 7

1.3 TCP/IP 8

2 Multi-Path technológiák 10

2.1 Multi-Path TCP (MPTCP) 11

2.1.1 Kapcsolatfelépítés 12

2.1.2 Adatátvitel 12

2.1.3 Kapcsolatbontás 13

2.2 MPT 13

2.3 MPT és MPTCP aggregációs képességének összehasonlítása 16

2.3.1 MPT Mérések 17

2.3.2 MPT mérések (100Mbit/s) 18

2.3.3 MPTCP mérések (100Mbit/s) 24

2.3.4 Mérések 1000Mbit/s sebességen 31

2.3.5 Az átviteli sebességek modellezése 37

2.4 1. Téziscsoport 39

3 Konténer technológiák 41

3.1 Docker 43

3.1.1 Docker Engine 43

3.1.2 Docker konténer ... 44
3.1.3 Docker Hub ... 46
3.2 Singularity ... 46
4 Konténerek összehasonlítása ... 48
4.1 Mérési összeállítás ... 48
4.1.1 Teszt infrastruktúra ... 49
4.2 Konténerek létrehozása ... 50
4.2.1 LXC ... 50
4.2.2 Docker ... 51
4.2.3 Singularity ... 51
4.3 Mérések ... 53
4.3.1 Leggyakoribb konténer hálózati üzemmódok ... 56
4.4 Hoszt mód mérési eredmények ... 59
4.5 2. Téziscsoport ... 62
5 MultiPath használata konténer technológiákkal ... 63
5.1 mérési összeállítás és telepítés ... 63
5.2 Mérések bemutatása ... 64
5.3 Mérési eredmények ... 65
5.3.1 Tesztmérések ... 65
5.3.2 Konténerizált MPT ... 65
5.3.3 Konténerizált MPTCP ... 67
5.4 3. Téziscsoport ... 69
6 Összefoglalás, jövőbeni kutatási tervek ... 70
6.1 Mérési módszertant érintő kutatások ... 70
6.2 Eltérő sebességű linkek aggregációja ... 71

6.3 Aggregációs vizsgálatok nagyobb CPU magszám mellett 72

7 Köszönetnyilvánítás 73

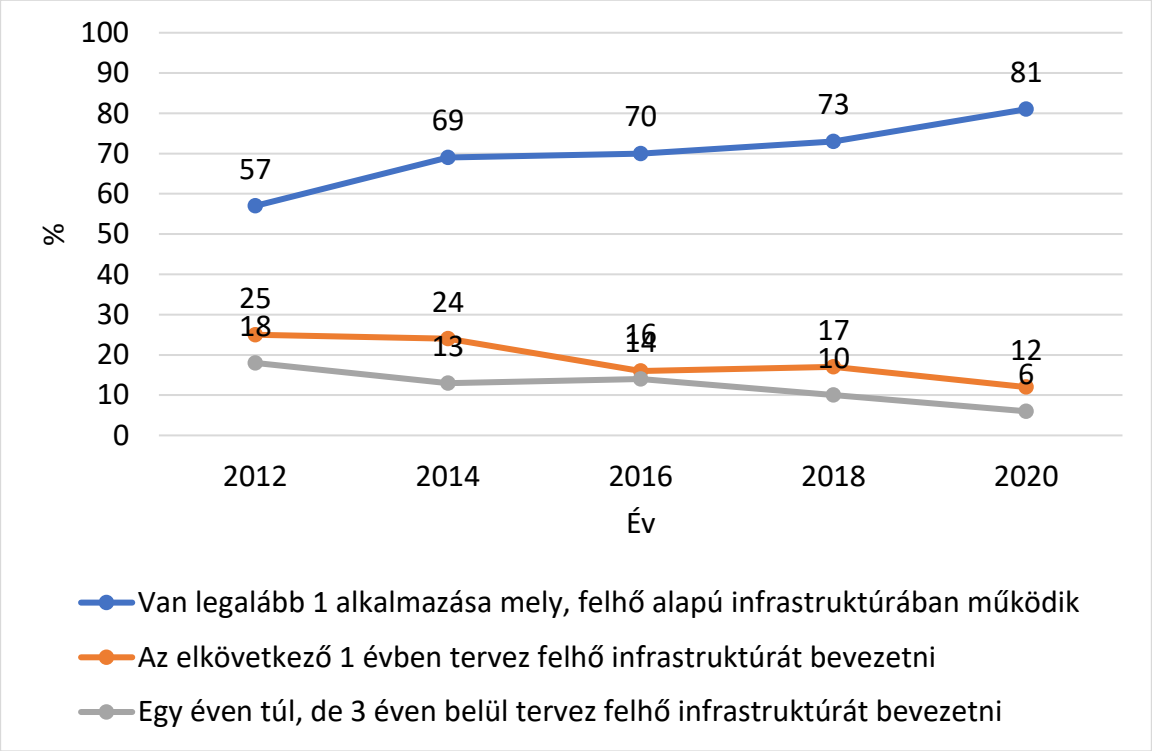
Ábrajegyzék 74

Irodalomjegyzék 76

1 Bevezetés

A felhő alapú infrastruktúra napjaink legtöbb informatikai rendszerében már elterjedt kifejezés. A számítási felhő az informatikában korunk egyik meghatározó fogalma, mely szakít az eddig megszokott nézőponttal, és előrevetíti a fizikai erőforrásoktól való eltávolodást. Felhőt használva nem tudhatjuk, hogy az általunk használt számítási teljesítmény, illetve tárhely milyen fizikai eszközön, illetve hol helyezkedik el.

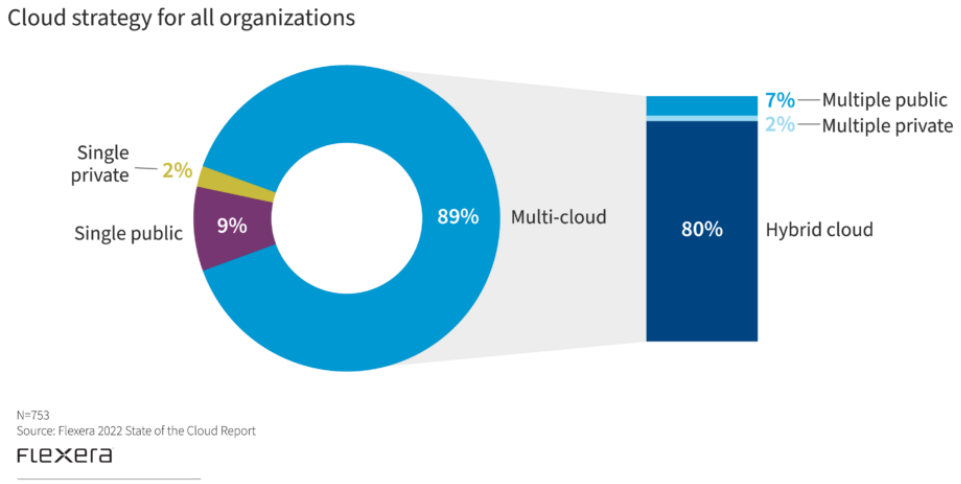
Az IGD 2020-as Cloud Computing Survey szerint a nagyvállalatok több mint 80%-a kiszervezte már valamilyen infrastruktúráját vagy alkalmazását felhő infrastruktúrára, vagyis van olyan rendszere, amely felett nincs teljes kontrollja. A résztvevő cégek mintegy 9%-a csak felhő alapú szolgáltatásokkal működteti IT infrastruktúráját. [1]



1. ábra. Felhő alapú infrastruktúrák eloszlása [1]

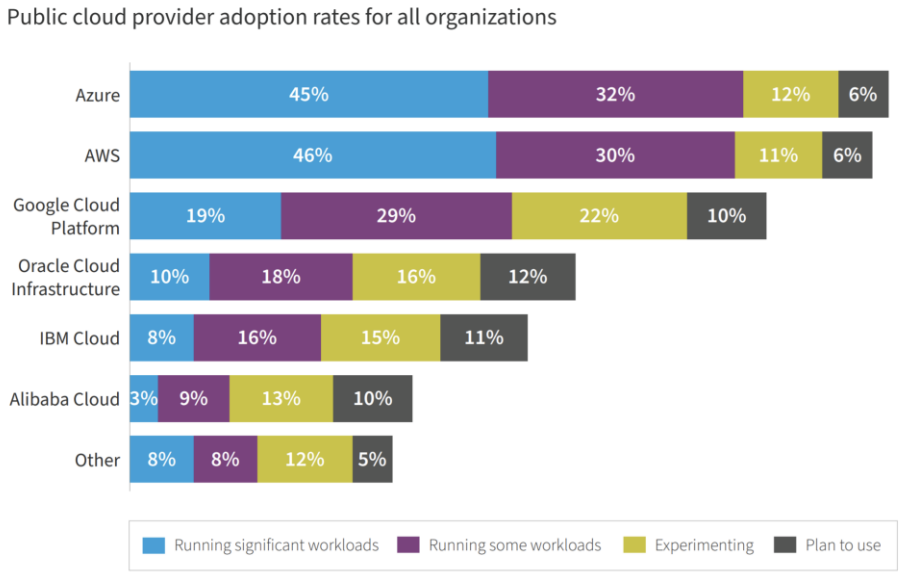
A Flexera legfrissebb kutatásából kiderül, hogy továbbra is a felhő számítási rendszerek dominálnak azon belül is a multi-cloud vagy hibrid megoldások mely során a vállalatok

használnak privát és publikus felhő szolgáltatásokat IT infrastruktúrájuk működtetésére. [2]



2. ábra. Vállalati felhő stratégák alakulása 2022-ben [2]

A felmérés alapján az egyes felhő szolgáltatók különböző megoldásai egyre inkább elfogadottak, a megkérdezett nagyvállalatok mintegy 70-80% már komoly erőforrásokat szervezett ki saját IT infrastruktúrájából a felhőbe. Jelen felmérésből kiderül, hogy a Microsoft Azure és az Amazon AWS a két meghatározó vállalat ebben a piaci szegmesben, majd őket követi a Google Cloud Platfrom, az Oracle Cloud Infrastructure valamint az IBM Cloud. [2]



3. ábra Felhő szolgáltatók használata [2]

Az NIST (National Institute of Standard and Technology) körülírta – még ha csak vázlatosan is – a felhő alapú rendszerek leglényegesebb tulajdonságait: [3]

- igény szerinti önkiszolgálás (On-demand self service)
- nagyteljesítményű hálózat (Broad network access)
- erőforrás készletekre épülnek (Resource pool)
- teljes és gyors rugalmasság (Rapid elasticity)
- mérhető adatokat biztosítanak a szolgáltatás igénybevételéről (Measured Service)

A „leglényegesebb” tulajdonság arra utal, hogy bármely felhőszolgáltatást vesszük igénybe, ezek a tulajdonságok várhatóan igazak lesznek rá.

1.1.1 Igény szerinti önkiszolgálás

Amennyiben egy ismert szolgáltatótól veszünk igénybe ilyen szolgáltatást, magától értetődőnek vesszük az önkiszolgálást. Ennek lényege, hogy az adott feladatnak megfelelő erőforrásokat megadva „válogathatjuk” össze az általunk használni kívánt eszközök teljesítményét. Az ilyen modellt leegyszerűsítve akár egy Google fiók létrehozásához is hasonlíthatjuk, hiszen a szolgáltató csak az igénylés módját írja elő, de a sikeres regisztráció után gyakorlatilag szabadon használhatjuk a fiókhoz tartozó erőforrásokat, természetesen a szolgáltató által megszabott határok között. [3]

1.1.2 Nagyteljesítményű hálózat

A nagyteljesítményű hálózat egyszerre jelenthet megfelelő sávszélességet, a fogyasztóhoz közeli szolgáltatást, valamint akár a hálózati eszközök sokszínűségéről is árulkodhat. [3]

1.1.3 Erőforrás készletek

A klasszikus formában vett erőforráskészletek a felhő infrastruktúrában nem használhatóak, hiszen az erőforrások itt nem konkrét processzor, memória, vagy merevlemez modulokban mérhetőek, hanem kapacitásokban. Tehát a felhőben számítási teljesítményt, memóriamennyiséget, és tárhelyet vehetünk igénybe, attól függetlenül,

milyen tényleges hardvert használunk. Az erőforráskészletek tényleges nagysága illetve összeállítása a felhasználó számára teljesen transzparens. Gyakorlatilag a felhő, mint fogalom is ezt hivatott szemléltetni. A felhő azt jelképezni, hogy az informatikai architektúra a fogyasztó számára teljesen lényegtelen. [3]

1.1.4 Rugalmasság

A rugalmasság már nagyban épít az előző tulajdonságra, hiszen az önmagát kiszolgáló fogyasztó az adott határokon belül korlátlanul igényelhet és mondhat vissza erőforrásokat. Ennél fogva a rugalmasság a fogyasztó szempontjából a skálázhatóságot jelenti, mint például néhány kattintással megoldható egy egyetemen az újonnan felvett 2000 hallgató levelezésének ellátása, vagyis a fogyasztó a váratlan erőforrás-igényeket pillanatokon belül kielégítheti. [3]

1.1.5 Mérhető adatok

Egy vállalaton belüli informatikai osztály nehezen elképzelhető, hogy profitorientált legyen, ugyanakkor, ha az informatikai költségek nem érzékelhetőek a fogyasztók részéről, akkor az elvárások és az igények mindig is nagyok lesznek, de amint számszerűsítik, a fogyasztó is csak annyi erőforrást vesz igénybe, amennyire tényleg szüksége van. [3]

1.2 A kutatási téma és annak időszerűsége, a kutatási téma fontossága

A felhő alapú infrastruktúra egyik legmeghatározóbb eleme a hálózat. Ezen rendszer segítségével férhetünk távolról hozzá az általunk kíván erőforrásokhoz. A hálózat, mely az egyes autonóm informatikai eszközök, mint pl.: switchek, routerek, szerverek, tűzfalak között teremt kapcsolatot. A jelenleg eladott informatikai eszközök szinte mindegyikében több hálózati interfészt találunk, figyeljük meg a legtöbb notebookot, melyben WLAN és vezetékes hálózati kapcsolódás, esetleg mobil kommunikációra alkalmas modem is található, vagy gondoljunk a nagyteljesítményű

szerverekre, melyekben alaplapra szerelve legalább 2, de általában több hálózati interfészt is találunk. Nem kell azonban speciális eszközökre gondolunk, vegyük például a mindenki zsebében ott lapuló mobiltelefont, melyben WLAN és mobil kommunikációra egyszerre alkalmas modemeket építenek be. Ezeket mind-mind csak külön tudjuk használni az egyik legelterjedtebb átviteli protokoll miatt.

A kor trendjei azt mutatják, hogy az adatátviteli sebesség növekedése megállíthatatlan, mindenki gyorsabban szeretné a tartalmakat letölteni. Ezek az igények nem csak a végfelhasználói eszközökben, de a kiszolgáló hálózatban és a Core elemek között is egyre nagyobb kihívást jelentenek mind a gyártóknak, mind az üzemeltetőknek.

Ezen informatikai rendszerek legmeghatározóbb protokollja a TCP/IP. Ez a megbízható protokoll írja le, milyen módon szállítsunk adatot egyik végponttól a másikig biztosítva az átvitt információ integritását.

1.3 TCP/IP

A TCP/IP protokollt több mint negyven éve publikálták először. [4] Ez a protokoll jelenti az ARPA-NET alapjait, valamint ez a protokoll azóta is az Internet teljes információátvitelének nagy részét képezi.

A TCP protokoll a kapcsolatfelépítés (3 utas kézfogás) után folyamatos adatfolyamokban szállítja oktettenként az adatot. A protokoll képes az adatok visszaállítására amennyiben az sérült, elveszett, duplikált, valamint rossz sorrendben érkezett meg a vevő oldalra.

Minden oktett egy sorszámot (Sequence number) kap az átvitel során, és egy nyugtát (acknowledgment) vár, hogy egy bizonyos sorszámig a csomagok hibátlanul érkeztek meg. Ha ez a meghatározott ideig nem érkezik meg (timeout), akkor a csomagokat újra küldi. A vevő oldalon a sorszám alapján ellenőrizhetjük, hogy a csomagok a megfelelő sorrendben érkeznek meg, vagy, hogy nincs köztük duplikált csomag.

A hibás csomagok kiszűrésére a küldő fél egy ellenőrző összeget (checksum) is küld az adott csomagról, mely alapján a fogadó fél ellenőrizheti a csomagok hibátlanságát.

A TCP egy kapcsolat orientált protokoll, vagyis amíg adatot küldünk át, a kapcsolat és a hozzá tartozó erőforrások le vannak foglalva, amennyiben a megfelelő ideig nem érkezik újabb csomag, a TCP bontja a kapcsolatot (4 utas kézfogás).

Az informatikai rendszerek evolúciójának is nevezett felhő alapú infrastruktúra fejlődéséhez elengedhetetlen az információátvitel fejlődése is. A mai modern eszközökben egyszerre több hálózati kapcsolatot biztosító interfész van. Gondoljunk csak az okostelefonokra, ahol Wi-Fi, és mobilinternet kapcsolat is rendelkezésre áll vagy a modern szerverekre melyek legalább két hálózati interfésszel rendelkeznek.

Ezek kiaknázására a klasszikus TCP/IP használata során nincs mód, mivel egy hálózati kapcsolatot 5 paraméter határoz meg. Cél és forrás IP cím, cél és forrás port szám, valamint a kommunikációs protokoll (TCP vagy UDP). [5] Ez az öt paraméter együtt jelent egy adatátvitelt. Természetesen több ilyen is létezhet két végpont között, de ezek közül csak maximum 3 lehet ugyanaz. Annak érdekében, hogy egy adott TCP folyamhoz tartozó csomagot más útvonalon küldjünk el a fogadó félhez, úgy a forrás IP címet (vagy portszámot) meg kell változtatni. [6], [7] Ez a TCP folyam töréséhez vezet, hisz a más IP címről érkezett adatokat nem tud demultiplexálni. Ahhoz, hogy az egy adott adatátvitelre több interfész vonali sebességét ki tudjuk használni a klasszikus TCP/IP protokollt le kell cserélni egy olyan 3-4 OSI rétegben működő protokollra mely biztosítja a redundáns Multi-Path adatátvitelt.

2 Multi-Path technológiák

A Multi-Path technológia lényege, hogy a két kommunikáló fél között több különböző útvonalat határozunk meg, melyeken párhuzamosan hozhatunk létre kommunikációs csatornákat. Ezzel természetesen nemcsak az adatátviteli sebesség növekedhet drasztikusan, hanem a redundanciát is növelhetjük. A Multi-Path technológiákat számos kutató tanulmányozta az elmúlt évek során, melyek mögött több motiváció is mutatkozik. Vegyük csak alapul a szokványos IT eszközöket (okostelefonok, tabletek, notebookok) melyek legtöbbjei több hálózati interfésszel rendelkeznek (Ethernet, Wi-Fi, 3G, 4G, 5G) melyek közül a TCP/IP protokoll sajátosságai miatt csak egyet lehet egy adott kommunikációra használni. A másik fontos felhasználási terület az adatközpontokban való felhasználás, a modern szerverekben, melyek a felhő infrastruktúrát alkotják, általában több hálózati interfész van, melyeket felhasználva a hálózati kihasználtság optimalizálható, valamint még hibatűrőbbé tehető az infrastruktúra. Amennyiben az összes hálózati interfész egyidejűleg használható, a hálózatok közötti váltás is gördülékenyebb, így a felhasználói élmény is javulhat.

A többutas megoldás nem újkeletű, az IT infrastruktúrákban már bizonyított eljárás, melyet használnak adatkapcsolati szinten két eszköz közötti direkt összeköttetéseknél, mint például a széles körben elterjedt Link Aggregation Control Protocol (LACP). Ez két egymással „szomszédos” hálózati eszköz között valósítja meg a többutas kommunikációt oly módon, hogy az egyes linkek sebességét különböző szervezések mentén aggregálja. Így nemcsak redundanciát visz az adott átvitelbe, de annak sebessége is közel lineárisan növekedhet az átvitelhez hozzáadott linkek függvényében. Adatközpontokban már alkalmazzák a PS (Packet Scattering) technológiát. Az adatközpontot röviden tanulmányozták a [8]-ban, és itt tárgyalták a további teszteket a [9]-ben. A Packet Scatter (PS) mögött meghúzódó ötlet az, hogy az ECMP (Equal-cost multi-path routing) switchek válasszanak egyet az aktuálisan aktív és routolható portok közül csomagonkénti (per packet), nem pedig folyamonkénti (flows) alapon, mint ahogy az itt található Valiant Load Balancing [10]. Így meg lehet osztani a forgalmat a lehető legegyenletesebben a két végpont között felhasználva az összes útvonalat. A több útvonal használata, és azok esetleges különbözősége miatt

nagyon valószínű, hogy a csomagok újra rendezése szükségesség válik. Ezáltal egy robosztus "Fast Retransmit" algoritmust kell használni a nem sorban érkező csomagok újra rendezéséhez. Azzal érveltek, hogy ha a forgalmi terhelés egyenlő a szerverek között, és egy adatközpont egységes hálózati topológiával rendelkezik, mint például a FatTree [11] vagy a VL2 [12], akkor a PS tökéletes terheléselosztást ér el egy adatközponton belül, és kiküszöböli a torlódást az azonos rétegek között [8]. Bár a csomagonkénti switchelés nem okoz hotspotokat a core hálózatokban és adatközpontokban, az a hálózati forgalom mely ECMP-n keresztül van szétosztva, folyamonként még mindig okozhat torlódásokat, hiszen osztozhatnak ugyanazon fizikai hálózati interfészen.

További széles körben elterjedt többutas kommunikáció a SAN (Storage Area Network) hálózatokban szerver és Storage rendszerek között. Léteznek még olyan Multi-Path megoldások, melyek a hálózati topológia felsőbb szintjén vagy ad-hoc hálózatokban valósítanak meg Multi-Path kommunikációt (DSR, MSR). [13]

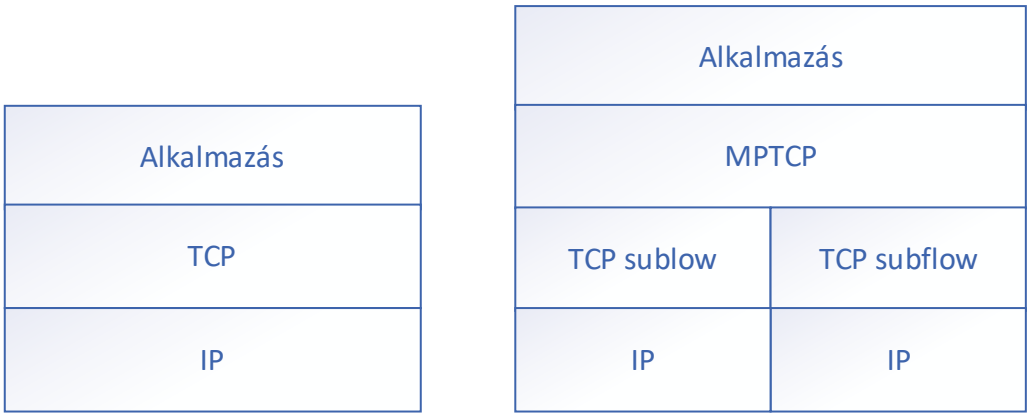
A cél az, hogy olyan Multi-Path technológiát használjunk, mely során a küldő és a fogadó fél a TCP/IP korlátait átlépve több hálózati interfész és hálózati hozzáférés segítségével megnövelje adatátviteli kapacitását, úgy, hogy a résztvevő felek nem azonos hálózatban találhatók.

2.1 Multi-Path TCP (MPTCP)

A Multi-Path TCP egy nemrég kiadott RFC [14] szerinti TCP kiegészítés, mely használatával a TCP adatfolyamot TCP-subflow adatfolyamokra bontjuk, melyeket aztán külön hálózati útvonalakon továbbítunk a kliens felé. Ezeket a TCP subflowkat a kernel külön TCP kapcsolatként kezel, így áthidalja a szokványos TCP korlátait. Ezek a subflowk egy-egy külön TCP kapcsolatként működnek.

Az MPTCP megalkotásánál a fejlesztők ügyeltek a visszafele kompatibilitásra, illetve arra az esetre is, ha a kommunikációban résztvevő felek valamelyike nem rendelkezik a TCP ezen kiterjesztésével, vagy csak egyetlen hálózati interfésszel

rendelkezik. Beállítása során szükséges a megfelelő kernel módosításokat elvégezni, valamint a routing táblában is módosításokat kell eszközölni (lásd beállítások).



4. ábra. TCP és MPTCP felépítése

2.1.1 Kapcsolatfelépítés

A szokványos TCP kapcsolatfelépítés abban módosul, hogy a 3 utas kézfogásban a SYN és SYN/ACK egy MP_CAPABLE opciót is tartalmaznak. Ez az opció különböző célokat szolgálhat. Kezdetben felméri, hogy a fogadó hoszt MPTCP kompatibilis-e, valamint információt cserélhetnek a kommunikációban résztvevő felek a későbbi TCP subflowk létrehozásához, authentikálásához.

Ha a kommunikációban mindkét hoszt MPTCP kompatibilis, a MP_CAPABLE opcióval a felek kulcsot cserélnek, melyekkel a későbbi subflow-kat azonosítják. A subflow-k kapcsolatfelépítésére ugyanazt a 3 utas kézfogást használják, mint a szokványos TCP kapcsolat felépítésre, annyi kiegészítéssel, hogy itt egy úgynevezett MP_JOIN opciót tartalmaznak a SYN, SYN/ACK, ACK csomagok. [14]

2.1.2 Adatátvitel

Azért, hogy az MPTCP biztosítsa a megbízható folyamatos adatátvitelt a subflow-kon – melyek bármikor eltűnhetnek, majd újra megjelenhetnek a hálózat változásával – az MPTCP 64 bites sorszámot (DSN – Data Sequence Number) használ.

Azért, hogy minden adatot elláthasson sorszámmal, az MPTCP minden egyes subflow-nak egy 32 bites egyedi sorszámteret biztosít. Ezzel, valamint a szokványos TCP fejrészben található sorszámmal az MPTCP képes az elveszett csomagok újra küldésére akár másik TCP subflow felhasználásával. [14]

Az MPTCP-ben minden subflow osztozik a fogadó pufferen (receive buffer), valamint ugyanazt a csúszó ablakot hirdetik. Ez esetben a nyugtázás (acknowledgment) is kétszintű, hiszen egyszer a TCP subflowk adatátvitelét (adat szinten), másodszor pedig a subflowkat összefogó Multi-Path TCP adatátvitelét (kapcsolat szinten) is nyugtázni kell.

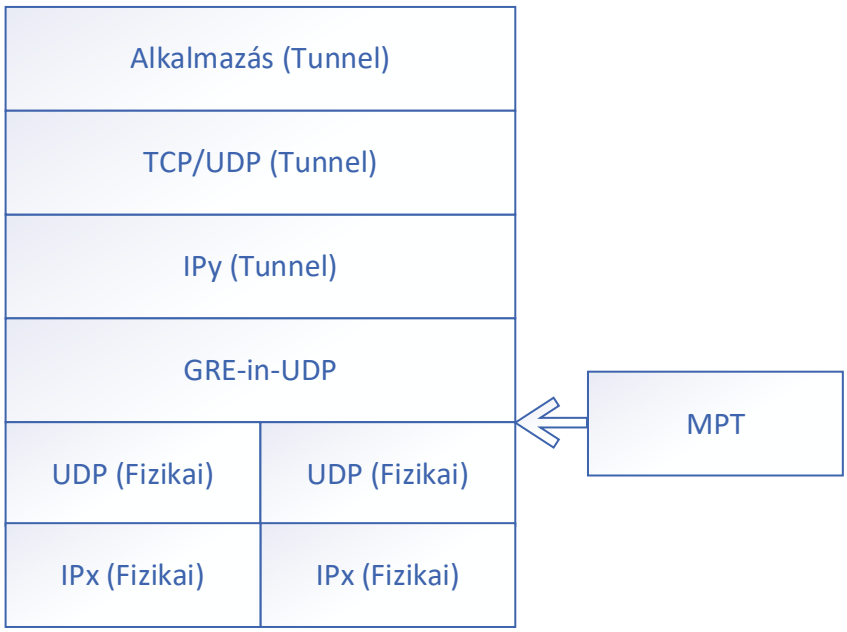
2.1.3 Kapcsolatbontás

Ha egy Subflow a kapcsolatát bontani kívánja, úgy a jól megszokott TCP kapcsolatbontást (4 utas kézfogás) alkalmazza. Az MPTCP másik megoldása az úgynevezett Fast Close, mely során az utolsó ACK flag mellett egy MP_FASTCLOSE is aktív, ezzel a két fél egy RST bites csomaggal válaszolva azonnal megszünteti a TCP kapcsolatot. [14]

2.2 MPT

Az MPT a Debreceni Egyetemen fejlesztett hálózati rétegbeli Multi-Path Library [15] mely működése közben egy tunnel interfészt hoz létre a különböző hálózati interfészeket és útvonalakat elfedve a GRE-in-UDP [16] protokoll használatával.

Az MPT-t lehetőség van routerként is használva site-to-site Multi-Path átjáróként használni. Mivel a fizikai hálózati interfész IP verziója, valamint a tunnel IP verziója különbözhet, az MPT-t akár IPv6 áttérést segítő technológiaként is kezelhetjük. [17]



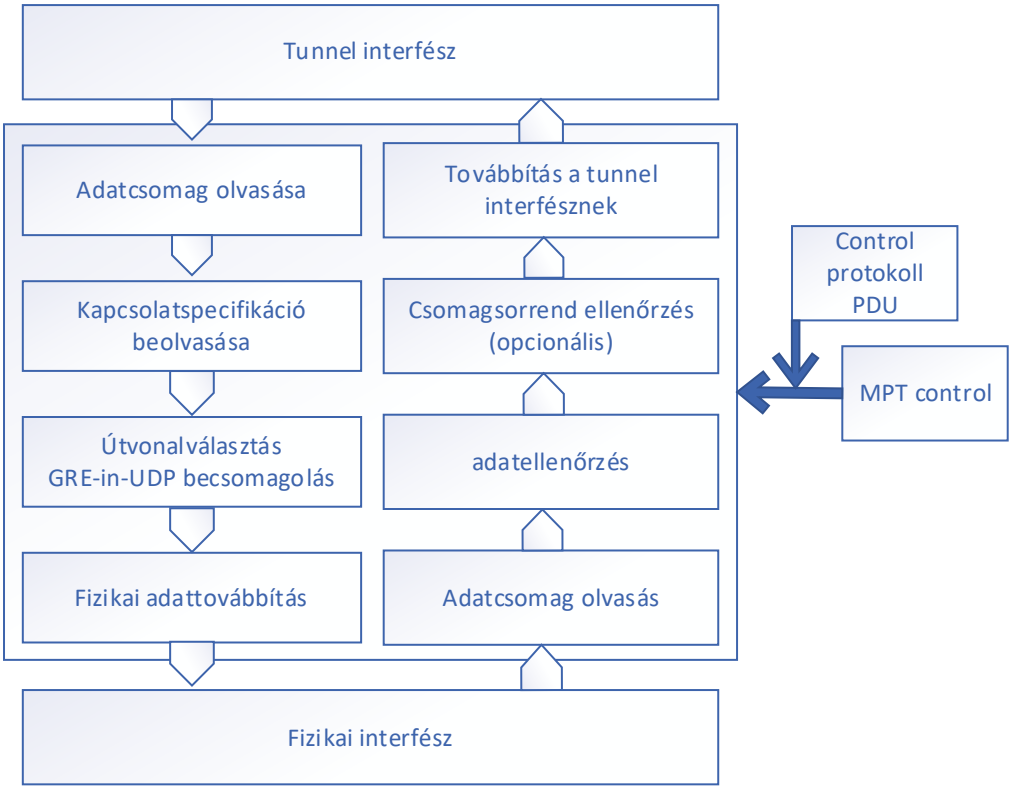
5. ábra. MPT architektúra (x, y ∈ {4, 6})

Az MPTCP-vel ellentétben az MPT egy user space alkalmazás. Elindítva, beolvassa a konfigurációs állományát, majd létrehozza ezek alapján a tunnel interfészt. Beállítása könnyebb, nincs szükség új kernel telepítésére, valamint a routing táblák módosítására.

Az MPT tunnelből érkező csomagokat az MPT alkalmazás az aktív interfészeken párhuzamosan szállítja, melyek között az MPT teremt kapcsolatot. Amennyiben küldeni szeretnék adatokat az MPT tunnelen keresztül, úgy az MPT kiválasztja a megfelelő útvonalakat, majd becsomagolja egy UDP csomagba, és a GRE tunnelen keresztül elküldi egy másik MPT szervernek, mely aztán kicsomagolja, és újra alkotja az eredeti csomagot.

Path IP IPv4 vagy IPv6	Path UDP (port 4754)	GRE-in-UDP	Tunnel IP IPv4 vagy IPv6	Tunnel TCP/ UDP	Application Data

6. ábra. Az MPT kommunikáció beágyazása



7. ábra. MPT működésének folyamata [15]

Mivel a különböző útvonalak más-más jellemzőkkel bírhatnak, így az egyes útvonalakhoz tartozó késleltetések is változhatnak. A MPT képes a csomagok megfelelő sorrendjének visszaállítására. A fogadó egy puffert használ, hogy tárolja az érkező (nem rendezett) csomagokat, majd a GRE sorszám alapján biztosítja a megfelelő sorrendet a fogadó tunnel interfésznek. [18]

Az MPT főleg két konfigurációs állományból áll:

- Alapvető információk az MPT szerverről
- Interfészek és kapcsolatok specifikálása

Alapvető információk közé tartozik a „general” szekción belül a GRE tunnel IP címe (lokális IPv4 és/vagy IPv6), timeout beállítások, a létrehozandó tunnel interfész neve, valamint az MTU (Maximum Transmission Unit). Lásd 2.3.2.

A kapcsolatok beállításai alatt az egyes Path-ok beállításait értjük, melyeknél szintén az IP címeket, GRE UDP portjait (lokális és távoli) és egyéb beállításokat kell megadni. Szintén lásd: 2.3.2.

Egy MPT kapcsolatot kétféleképpen indíthatunk. Ha az MPT szerver elindul, a beállított konfigurációs állományokat beolvasva és feldolgozva hozza létre a kapcsolatot a másik féllel. Természetesen ehhez szükséges a másik fél megfelelő beállítása és futtatása is, különben nem áll össze tunnel interfész. Ugyanakkor az MPT kliens mely az `mptsrv` különböző paramétereinek megváltoztatásáért felel, lokálisan képes akár különböző útvonalak ki-be kapcsolására is. A másik lehetőség, hegy ezen MPT klienssel hozzuk létre a kapcsolatot a megfelelő paraméterezéssel.

```
mpt address {add|del} IPADDRESS/PREFIX dev INTERFACE
```

A fenti példával adhatunk hozzá IPv4/v6 címet egy adott interfészhez.

Az alapvető beállítások a „connections” szekcióban helyezkednek el, melynél megadhatjuk a kapcsolatunk nevét (amennyiben több ilyen kapcsolatunk van, a könnyebb azonosítás érdekében). Adhatunk SEND és RECEIVE jogosultságot, melyek segítségével MPT szerverünk képes küldeni és fogadni a kapcsolathoz tartozó frissítéseket. [18]

Az MPT-ről számos kutatást publikáltak a közelmúltban. FTP adatfolyamokat sebességének tesztjét végezték el Cisco routerek segítségével relatív alacsony sebességen [19]. Emellett tesztelték az egyes hálózati forgalom priorizálási metódusokat [20]. Tesztelték már emulált WAN szimulátor segítségével, mely során olyan hálózati hibák terhelték az átvitelt, mint a jitter vagy a csomagvesztés [21]. MPT-t több helyen is megemlítik, mint az IPv4 és IPv6 integrálási technológiát IoT rendszerekben. [22]

2.3 MPT és MPTCP aggregációs képességének összehasonlítása

Jelen fejezetet a „Comparing the Aggregation Capability of the MPT Communications Library and Multipath TCP” [6] és az „Evaluation of the Aggregation Capability of the MPT Network Layer Multipath Communication Library and Multipath TCP” [23] cikkekben közölt mérési eredményeim alapján állítottam össze.

A méréseim célja annak a vizsgálata, hogy az MPT illetve az MPTCP hogyan képes egyre növekvő számú átviteli csatorna kapacitását összegezni. A rendelkezésemre álló eszközök lehetővé tették, hogy az átviteli csatornák számát illetően 1-től 12-ig végezzem el a vizsgálatot.

2.3.1 MPT Mérések

MPT esetén további vizsgálati szempont volt, hogy hogyan viselkedik a csatornakapacitás összegzése a használt IP verziószámától függően. Mivel mind a tunnel, mind az elemi csatornák szintjén mindkét verzió használatára lehetőségem volt, ezért kétszer két, azaz négy mérési sorozatot végeztem el.

A mérésekhez 2db Dell Precision 490 workstationt használtam.

Dell Precision 490 Workstation főbb paraméterei:

a) DELL 0GU083 motherboard, Intel 5000X chipset

b) 2xIntel Xeon 5140 2.33GHz dual core processzor

c) 8x2GB 533MHz DDR2 RAM (quad channel)

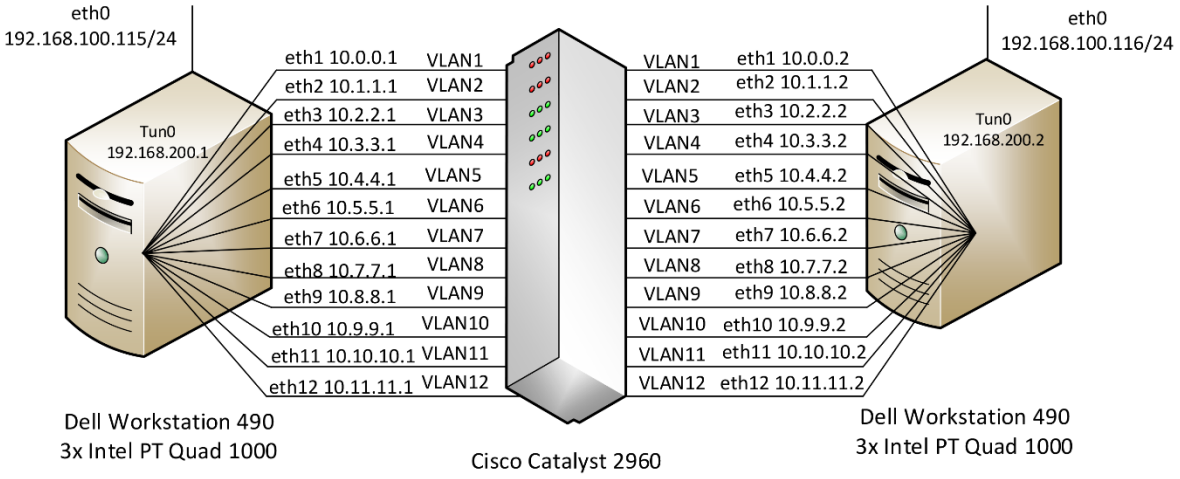
d) Broadcom NetXtreme BCM5752 Gigabit Ethernet controller (PCI Express, integrált)

e) Debian Jessie 8.3, Kernel verzió 3.16.7 amd64, gcc verzió 4.9.2-2

f) 3db Intel PT Quad hálózati interfész (3x4 port Gigabit Ethernet)

Mindkét számítógép 13 interfésszel rendelkezett, melyből 12-t a mérésekhez, egyet pedig a vezérléshez használtam fel. 100Mbit/s-os mérésekhez egy Cisco Catalyst 2960-as switchet alkalmaztam, hogy az átviteli sebességeket 100Mbit/s-ra korlátozzam. A Cisco konfigurációját jelen disszertációban nem részletezem, a mérési interfészeket páronként külön VLAN-ba helyeztem.

A tesztekhez az MPT GRE-2015-10-23-64bit-es verzióját használtam, mint a korábbi publikációmban [24], míg az MPTCP-ből a v0.92-t.



8. ábra. Multi-Path mérési összeállítás

2.3.2 MPT mérések (100Mbit/s)

A mérőkörnyezet összeállítása során először a hálózati infrastruktúrát, majd az interfészeket állítottam be oly módon, hogy minden interfész saját hálózati szegmensbe került. Ez után következett az MPT beállítása. Képes voltam különböző IP verziójú protokollok alkalmazására mind a logikai mind a fizikai eszközökön, mert explicit módon megadható, hogy melyik verziójú IP protokollt alkalmazzuk a két szinten.

Az MPT beállításához először a logikai hálózati interfészt állítottam be az `interface.conf` módosításával.

```
[general]
tunnel_num    = 1
; Accept remote new connection request
accept_remote = 1
cmdport_local = 60456
cmd_timeout = 25

[tun_0]
name      = tun0
mtu       = 1440
ipv4_addr = 10.0.0.2/24
ipv6_addr = fd00:de:200::2/64
```

1. kódrészlet: Az MPT logikai interfész beállítása IPv4 és IPv6 részére

Majd a fizikai interfészek beállítása következett a `connection` mappában található `conf` file segítségével.

```
;###################### Multipath Connection Information: ##############
[connection]
name                = MPT_connection_Ipv6
permission          = 3
ip_ver              = 6
ip_local            = fd00:de:200::2
local_port          = 23456
ip_remote           = fd00:de:200::1
remote_port         = 23456
remote_cmd_port     = 60456
path_count          = 12
network_count       = 0
status              = 0
reorder_window      = 900
max_buffdelay_msec = 300
auth_type           = 0
;########################## PATHS ############################
[path_0]
interface_name   = eth1
ip_ver           = 6
private_ipaddr    = fd00:de:201::2
remote_ipaddr    = fd00:de:201::1
keepalive_time   = 5
dead_time        = 11
weight_out       = 10
cmd_default      = 1
status           = 0
[path_1]
interface_name   = eth2
ip_ver           = 6
private_ipaddr    = fd00:de:202::2
remote_ipaddr    = fd00:de:202::1
keepalive_time   = 5
dead_time        = 11
weight_out       = 10
status           = 0
```

2. *kódrészlet: Fizikai interfészek definiálása (A terjedelem miatt csak az első 2 PATH beállítása látható)*

A mérés során az iparban de-facto sztenderdként elfogadott iperf mérőszoftvert alkalmaztam. Egy egyszerű szkriptet írva, minden sikeres mérés után újra konfiguráltam a hálózatot oly módon, hogy az eggyel több aktív hálózati interfészt alkalmazzon.

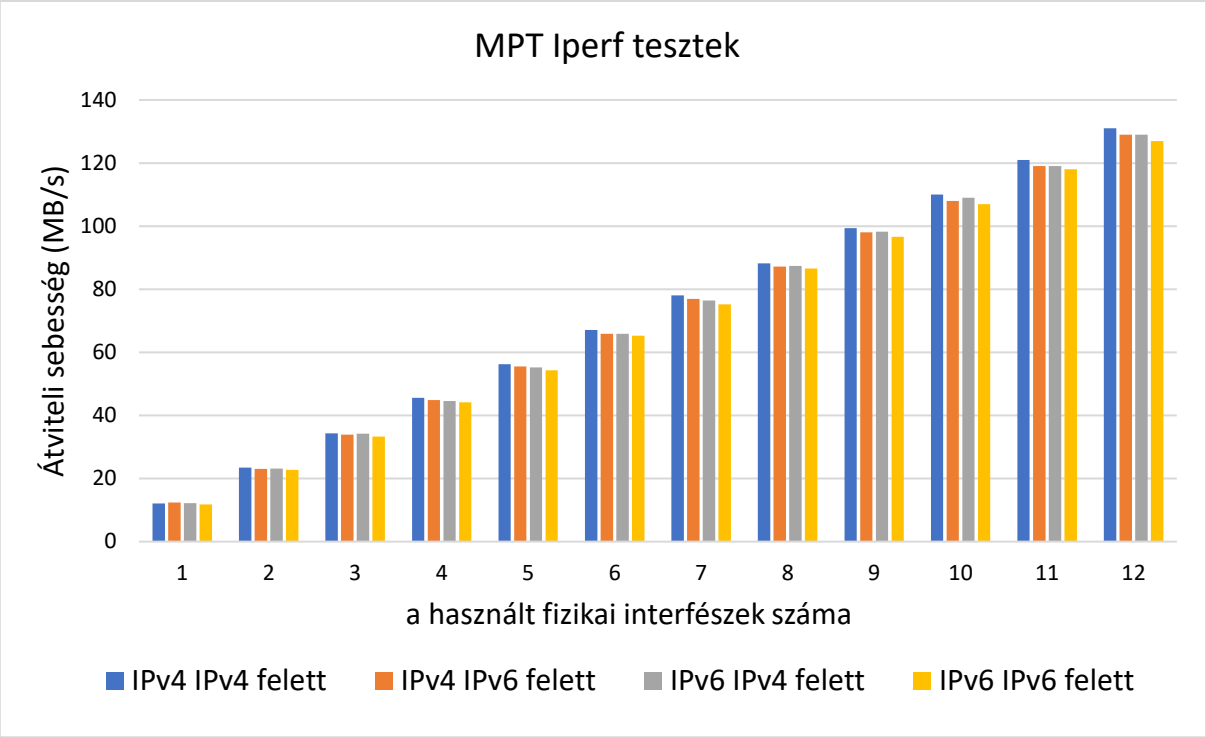
További vizsgálatként az iperf mérések mellett egy http letöltést is lefuttattam, mely során egy 8GB-os fájlt töltöttem le. Ahhoz, hogy kiküszöböljem a HDD/SSD elérésének esetleges sebességcsökkentő hatását, a szerver gépen RAMDISK-re helyeztem a 8GB-os random tartalmú fájlt, míg a kliensen a célnak a `/dev/null`-t adtam meg.

A méréseket a következő script segítségével automatizáltam:

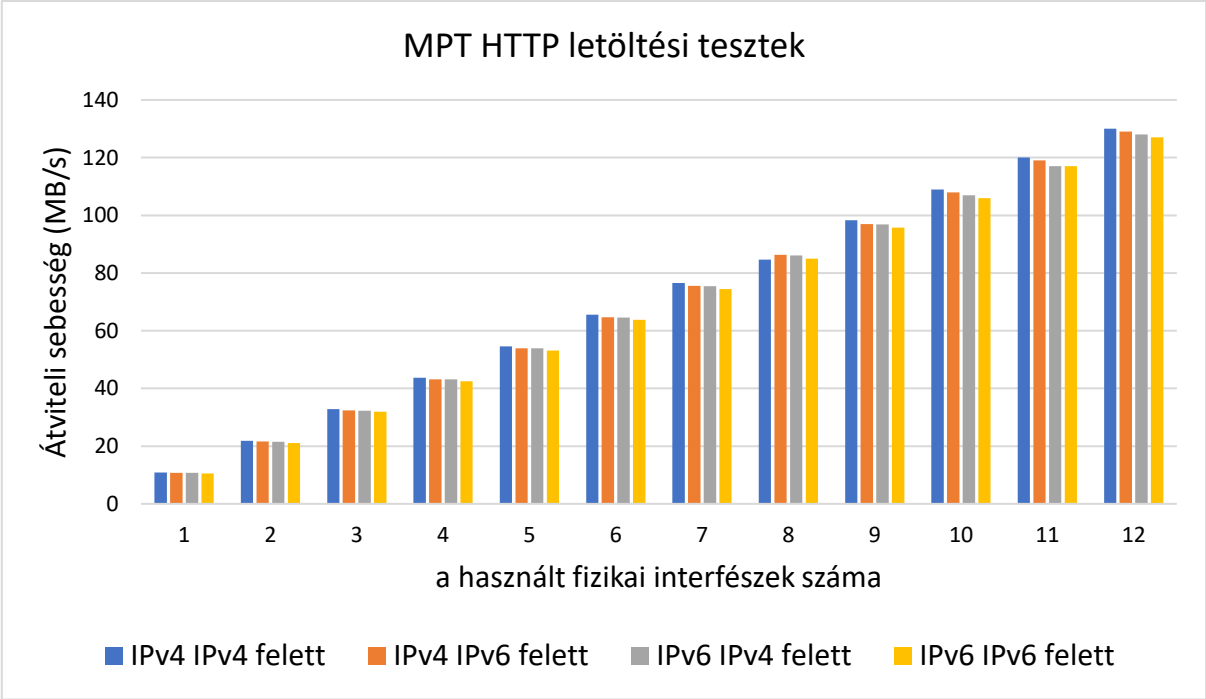
```
#!/bin/bash
# MPT ellenorzes
MPTSRV=`ps aux | grep mptsrv | wc -l`
if [ "$MPTSRV" = 1 ]; then
   echo "Az mptsrv nem fut!"
exit 1
fi
echo "Starting time : `date` "

CMD1="iperf -V -c 10.0.0.1 -t 60 -i 2 -f M"
CMD2=”wget -O /dev/n8GiB”

echo "Disabling devices"
ifconfig eth2 down
ifconfig eth3 down
ifconfig eth4 down
ifconfig eth5 down
ifconfig eth6 down
ifconfig eth7 down
ifconfig eth8 down
ifconfig eth9 down
ifconfig eth10 down
ifconfig eth11 down
ifconfig eth12 down
echo "done"

########################## 1 NIC ##################################
```

```
      echo "Active device(s): eth1"
      echo "Waiting for 30 seconds for mptsrv handle NICs"
      sleep 30
      echo "Running iperf TCP test"
      mkdir iperf
      eval $CMD1 >> iperf/1nic
      echo "Running wget TCP test"
      mkdir wget
      eval $CMD2 >> wget/1nic
      echo "Done"
      ##################################################################
      ……….
      ########################### 12 NIC ##################################
      ifconfig eth12 up
      echo "Active device(s): eth1 eth2 eth3 eth4 eth5 eth6 eth7 eth8 eth9
eth10 eth11 eth12"
      echo "Waiting for 30 seconds for mptsrv handle NICs"
      sleep 30
      echo "Running iperf TCP test"
            eval $CMD1 >> iperf/12nic
      echo "running wget TCP test"
      eval $CMD2 >> wget/12nic
      echo "Done"
      ###################################################################
      echo "#######################################################"
      echo "######################## ALL DONE!!######################"
      echo "#######################################################"
```

3. kódrészlet: Az MPT mérőszkript



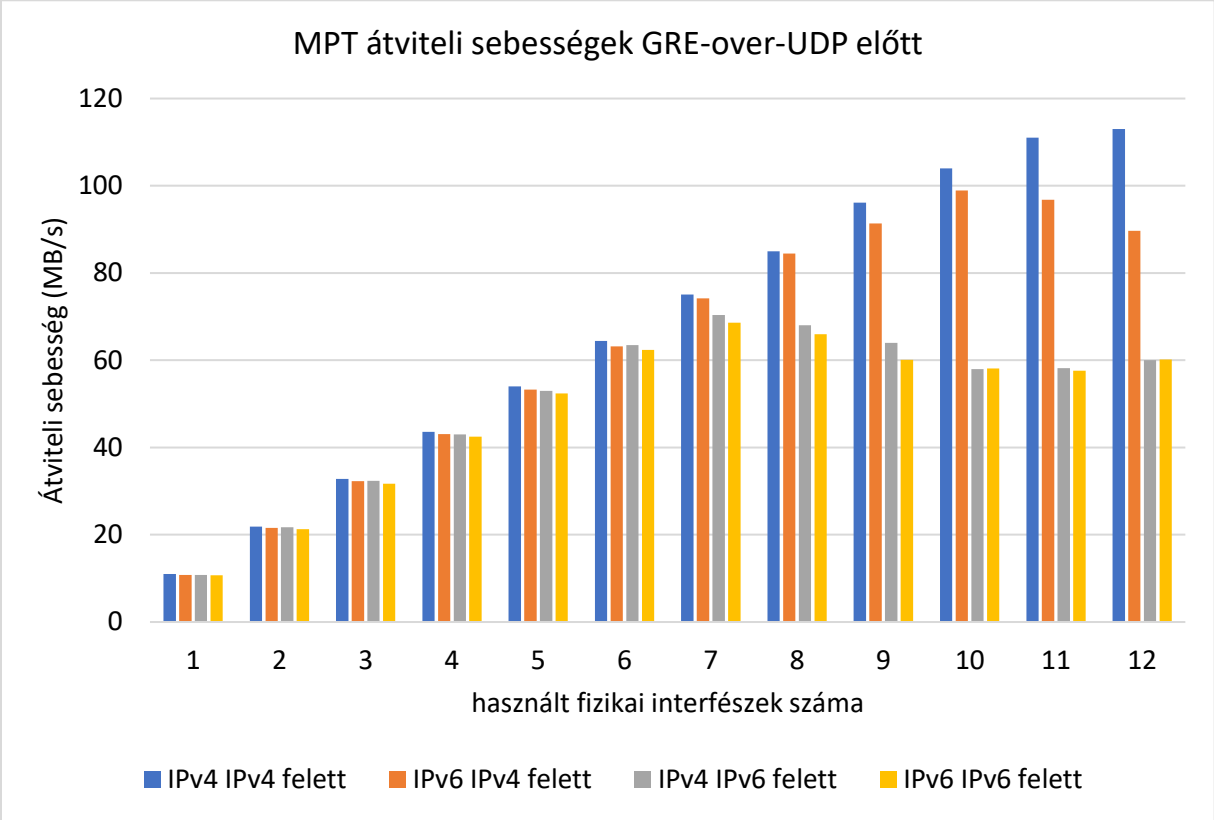
9. ábra. MPT iperf mérési eredmények (100Mbit/s)



10. ábra. HTTP letöltés mérési eredmények (100Mbit/s)

A mérési eredményeket a 9. és 10. ábra mutatja. Az MPT használatakor az összesített átviteli sebességnövekedés majdnem ideális. Az átvitel majdnem lineárisan skálázódott egészen 12 fizikai eszközig. Az Ipv4 és IPv6 protokoll minden lehetséges

permutációját tesztelve látható, hogy a közel azonos eredményt kapunk. A minimális eltérés az IPv6 címhosszúságának használatából adódik. Ez szignifikáns javulást mutat az előző mérési eredményeinkhez képest, ahol még egy régebbi verziót használtam a mérésekhez, mely nem a GRE tunnel protokollon alapult. [24]



11. ábra. MPT átviteli sebességek GRE-over-UDP előtt

Jól látható, hogy a régebbi megoldásnál, mely még nem GRE-over-UDP protokollt használ, az átviteli kapacitások elmaradnak a későbbi mérési eredményektől. Kifejezetten akkor mutatnak rosszabb eredményt, amikor az IPv6-ot mint a fizikai interfészek IP protokollját használjuk. Feltehetően az IPv6 128 bites címeinek hosszúságából ered ezen eredmény, szemben a IPv4 32 bites címhosszúságával. Emellet optimalizálták a CPU kihasználtságot is a jobb eredmények érdekében, mely szembetűnő sebességnövekedéshez vezetett. A GRE over UDP protokoll bevezetése tehát minden szempontból előnyösnek bizonyult. [24]

2.3.3 MPTCP mérések (100Mbit/s)

Mivel az MPT-vel szemben az MPTCP nem használ újabb IP réteget, az MPTCP esetén csak egy döntési pont volt, vagy IPv4 vagy IPv6-ot használhattam. Így ebben az esetben csak két mérési sorozatot végeztem: egyet IPv4-gyel, egyet IPv6-tal.

Az MPCTP teszthez ugyanazt az infrastruktúrát, de néhány beállítást megváltoztatva használtam. Elsőként egy a vanilla kerneltől eltérő kernelt kellett fordítanom, mely támogatja az MPTCP-t. Alapvetően a fejlesztők által elérhető if-up.d szkript használatával a beállítások automatikusan végbemennek, de a méréseim során nem alkalmazhatóak, mert a folyamatos interfész beállítások után nem minden esetben futnak le ezek a szkriptek, így ezt a mérő-szkriptembe integrálva használtam.

Az MPTCP lényegében minden egyes fizikai interfészhez rendel egy routing táblát, vagyis minden interfésznek megvan a maga alapértelmezett átjárója is. Így lehet az átviteleket két végpont között párhuzamosítani. Végül a MPTCP 3 utas kézfogása speciális kiegészítéseket kapott, mely során a két kommunikáló eszköz meggyőződik, hogy mindkét fél képes MPTCP kommunikációra így TCP subflow létrehozására. Példaként az eth1 interfész routing bejegyzésinek létrehozása:

```
ip rule add from 10.1.1.2 table 1

ip route add 10.1.1.0/24 dev eth1 link table 1

ip route add default via 10.1.1.1 dev eth1 table 1
```

Ahhoz, hogy a eth0-s kontrol interfészünket, melyet a mérések indítására használunk, ne vonjuk be a mérésekbe (ezzel torzítva az eredményeket), egy egyszerű IPtables szabállyal letiltottam az SSH-tól különböző bejövő kommunikációt az eth0 interfészen.

```
iptables -A INPUT -i eth0 -p tcp --dport !22  -j DROP
```

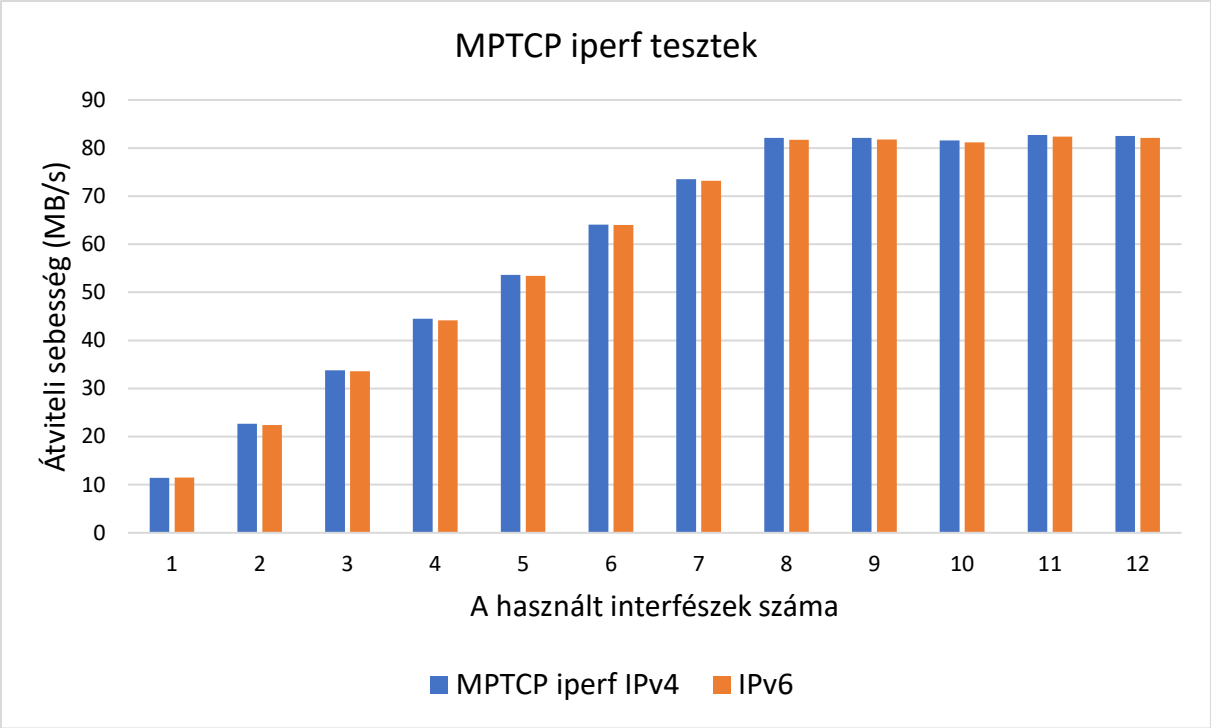
A mérő szkriptem majdnem ugyanaz maradhatott, annyi módosítással, hogy mivel az MPTCP nem hoz létre pluszban logikai interfészt is, így mérésre az első mérésben használt fizikai interfész IP címét állítottam be, illetve minden újabb interfész

hozzáadásánál az adott interfészhez tartozó routing beállításokat is a mérőszkript végezte.

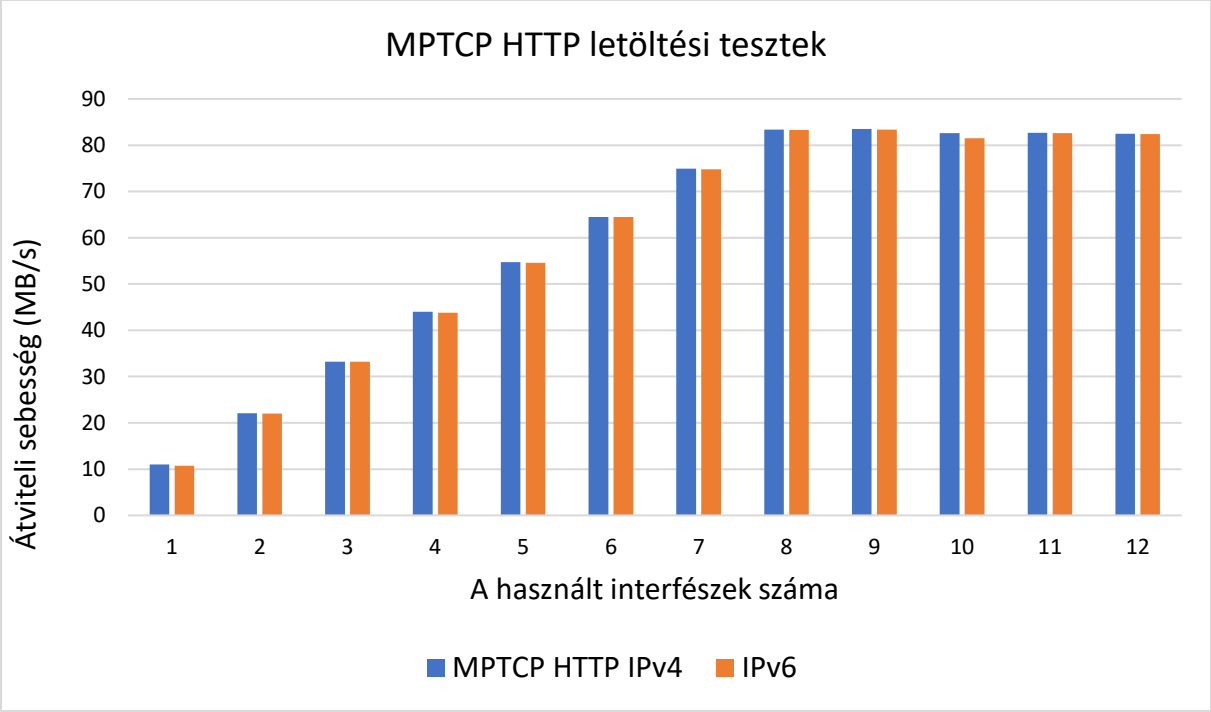
```
#!/bin/bash
#MPTSRV=`ps aux | grep mptsrv | wc -l`
#echo $MPTSRV
#if [ "$MPTSRV" = 1 ]; then
#echo "az mptsrv nem fut!"
#exit 1
#fi

./cpu.sh &

echo "starting time : `date` "
echo "starting time : `date` " > time

CMD1="iperf -V -c 10.0.1.1 -t 60 -i 2 -f M"
CMD2="wget -O /d0.1/8GiB"

echo "disabling devices"
ifconfig eth2 down
ifconfig eth3 down
ifconfig eth4 down
ifconfig eth5 down
ifconfig eth6 down
ifconfig eth7 down
ifconfig eth8 down
ifconfig eth9 down
ifconfig eth10 down
ifconfig eth11 down
ifconfig eth12 down
echo "done"

########################## 1 NIC ##################################

echo "active device : eth1"
ip rule add from 10.1.1.2 table 1
ip route add 10.1.1.0/24 dev eth1 link table 1
ip route add default via 10.1.1.1 dev eth1 table 1
```

```
echo "waiting for wait 15 sec for mptrv handle nics"
sleep 30
echo "running iperf TCP tests"
mkdir iperf

    eval $CMD1 >> iperf/1nic

echo "iperf TCP done"
#####################################################################

########################### 2 NIC ##################################
ifconfig eth2 up
echo "active device : eth1 eth2"
ip rule add from 10.1.2.2 table 1
ip route add 10.1.2.0/24 dev eth2 link table 1
ip route add default via 10.1.2.1 dev eth2 table 1

echo "waiting for wait 15 sec for mptrv handle nics"
sleep 30
echo "running iperf TCP tests"

    eval $CMD1 >> iperf/2nic

echo "iperf TCP done"
#####################################################################
```

4. kódrészlet: Az MPTCP mérőszkript



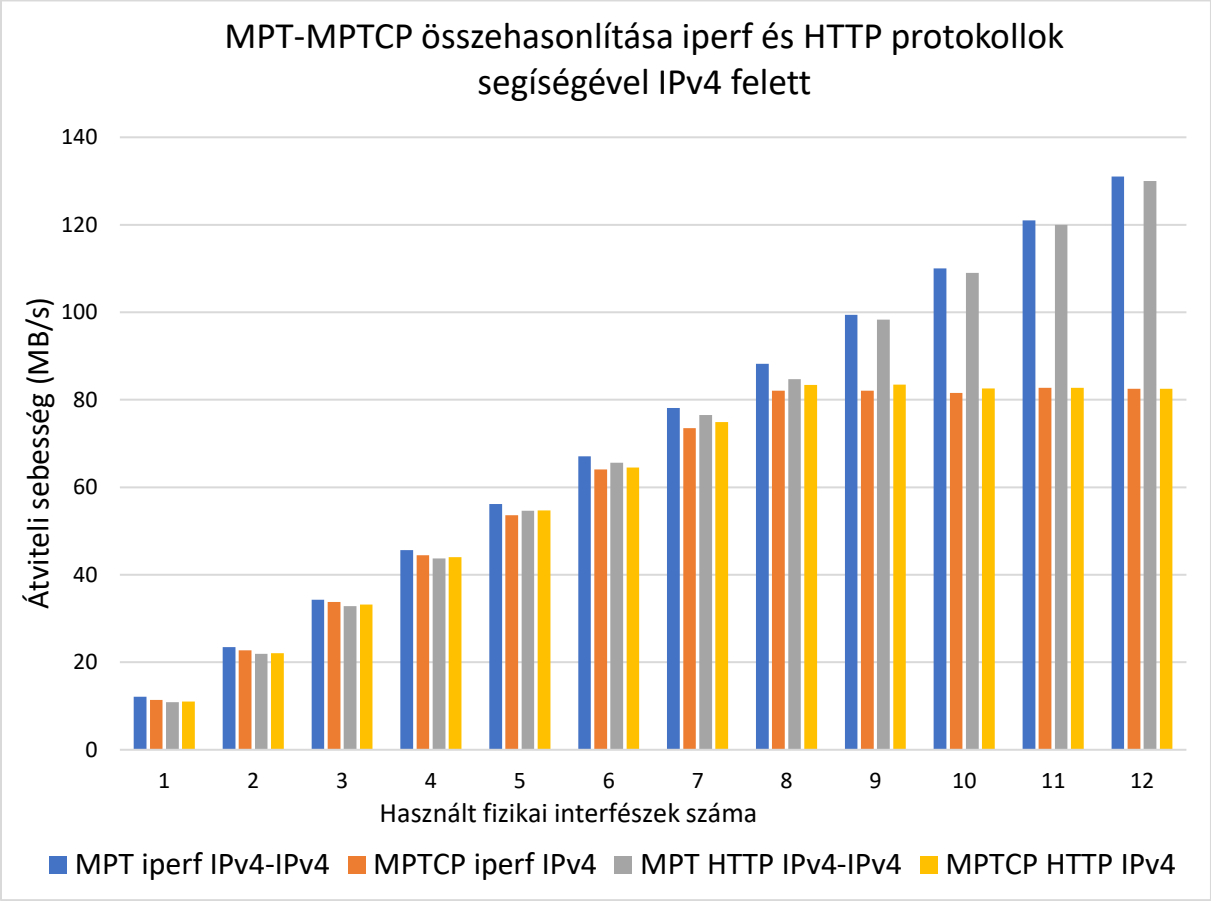
12. ábra. Az MPTCP iperf tesztek



13. ábra. MPTCP HTTP letöltési tesztek

A tesztekből jól látszik, hogy amint a 8. interfész is használatba kerül, úgy nincs további növekedés az átviteli sebességben. Ez ügyben felvettem a kapcsolatot a fejlesztőkkel, akik megerősítették, hogy nincs konfigurációs hiba, az MPTCP maximum 8 interfészt

tud egyszerre aggregálni. [25] Tehát az alábbi grafikon tökéletesen szemlélteti az MPT és MPTCP aggregációs képességei közötti legnagyobb különbséget:



14. ábra. Az MPT és MPTCP átviteli aggregációs képességinek összehasonlítása

Jól látható az ábrán az MPTCP 8 hálózati interfészes korlátja, az iperf és HTTP tesztek közel azonos eredményt mutatnak.

2.3.3.1 CPU kihasználtság

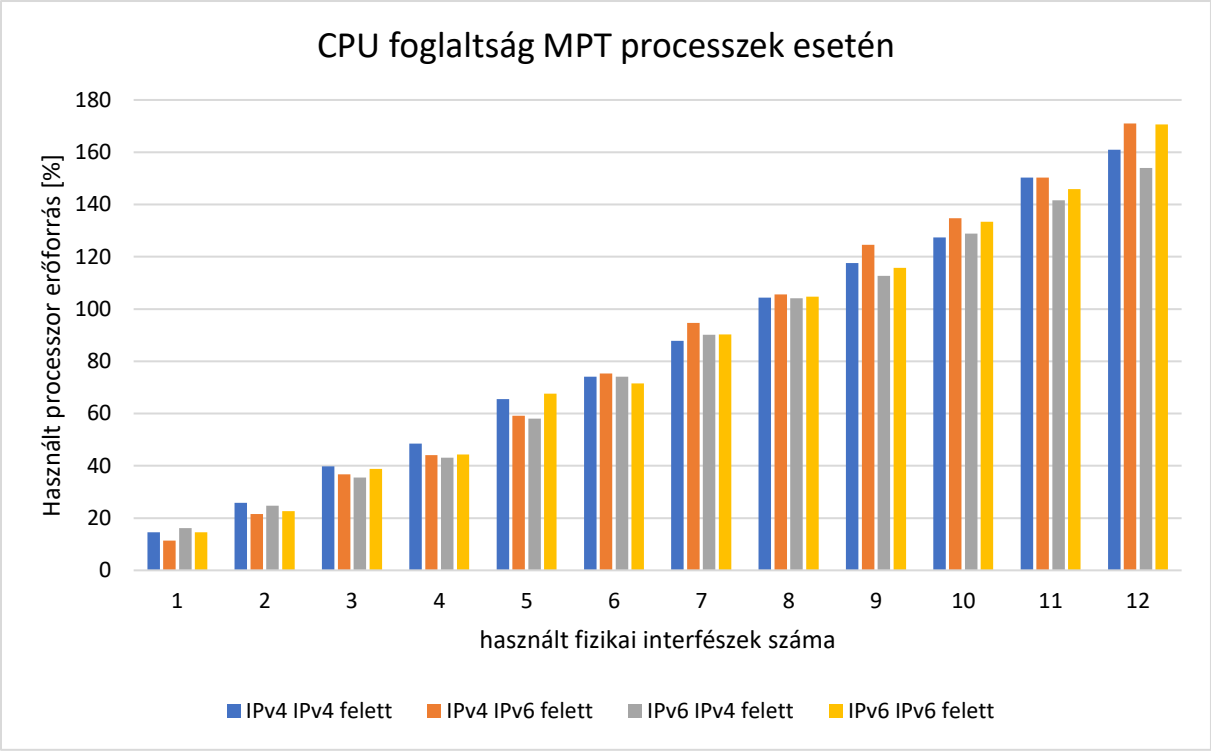
Ahhoz, hogy a két technológia közötti különbséget teljes egészében összehasonlítsuk, más paramétereket is meg kellett vizsgálnunk. A következő méréseim során megvizsgáltam, hogy a két technológia mekkora CPU erőforrást használ fel működés közben.

Ehhez a mérő szkriptemet úgy módosítottam, hogy egy másik egyszerű bash szkript monitorozta és naplózta 10 másodpercenként az éppen aktuális processzor

terheltséget, Mivel a mérések között 30 másodperc türelmi idő van, így a mérési eredményekből könnyen megállapítható volt mikor futott a mérés és mikor várt a következő mérésre a szkriptem. Ez a terhelés mely az MPT esetén az `mptsrv` processze, míg MPCTP esetén a szimplán a kernel terhelése okozott, mely az iperf processz terhelésében jelentkezett. A processzorterhelés mérő szkriptjét az átviteli mérő szkript inicializálásakor indítom párhuzamosan, mely figyeli, hogy a „szülő" szkript fut-e, illetve egyszerre méri a lokális és a távoli gép processzorterheltségét egy megfelelően paraméterezett `top` parancs segítségével. Ezeket aztán fájlként elmentettem, melyet később könnyen feldolgozhattam. A mérés végeztével az átviteli mérőszkript `killall` parancsot használva megszakítja a CPU terheltséget mérő szkriptem futását.

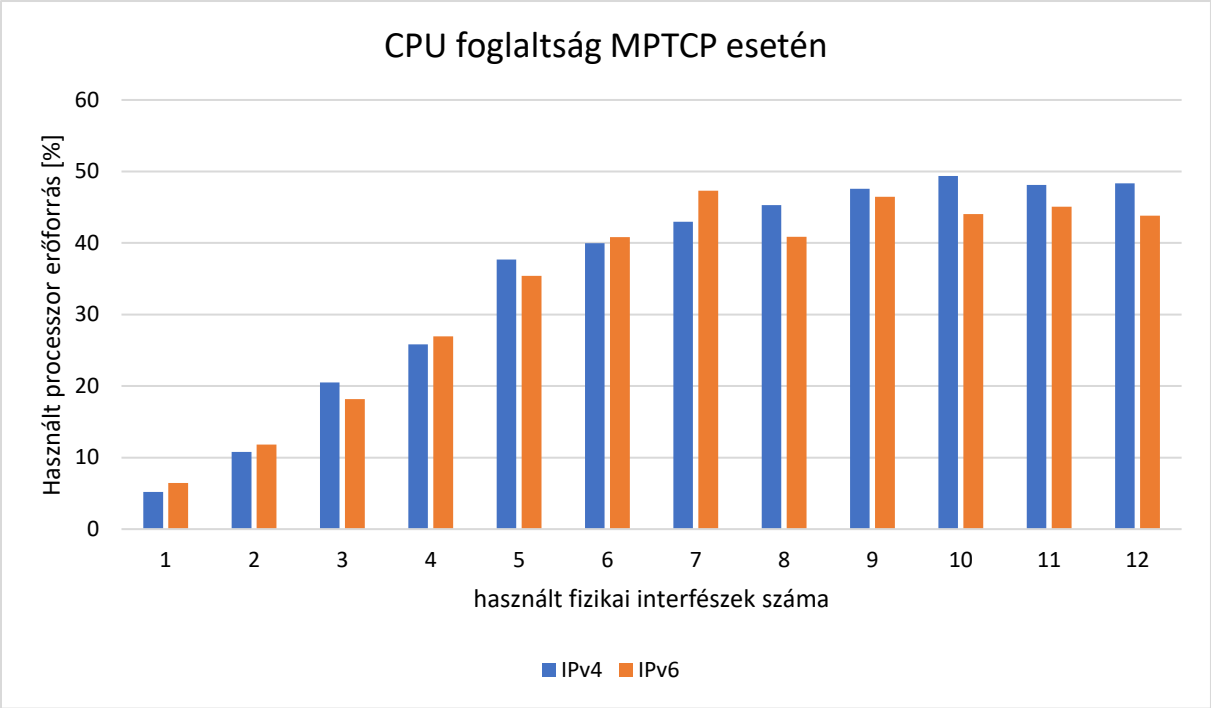
```
#!/bin/bash


for i in `seq 1 10000`
do
meres=`ps aux | grep meres |wc -l`
if [ "$meres" -eq 1 ];then
     top -b -n 1 | grep mptsrv | awk '{print $9}' >> cpu_usage_client
     ssh 192.168.100.140 top -b -n 1 | grep mptsrv | awk '{print $9}' >>
cpu_usage_server
else
     exit 0
fi
sleep 10
done
```

5. *kódrészlet: A CPU kihasználtság mérőszkriptje MPT esetén*



15. ábra. CPU foglaltság MPT processzek esetén

A mérések során kiderült, hogy az MPT majdnem 2 teljes processzormagot lefoglal, mikor több mint 10 interfészt használunk adatátvitelre. A mérések során mind a két kommunikációban résztvevő oldal hasonló mérési eredményeket produkált. (A rendszerünkben a teljes processzor terhelést 400%-nak vesszünk mivel 4 processzormag volt mind a kliens mind a szerver gépben.)



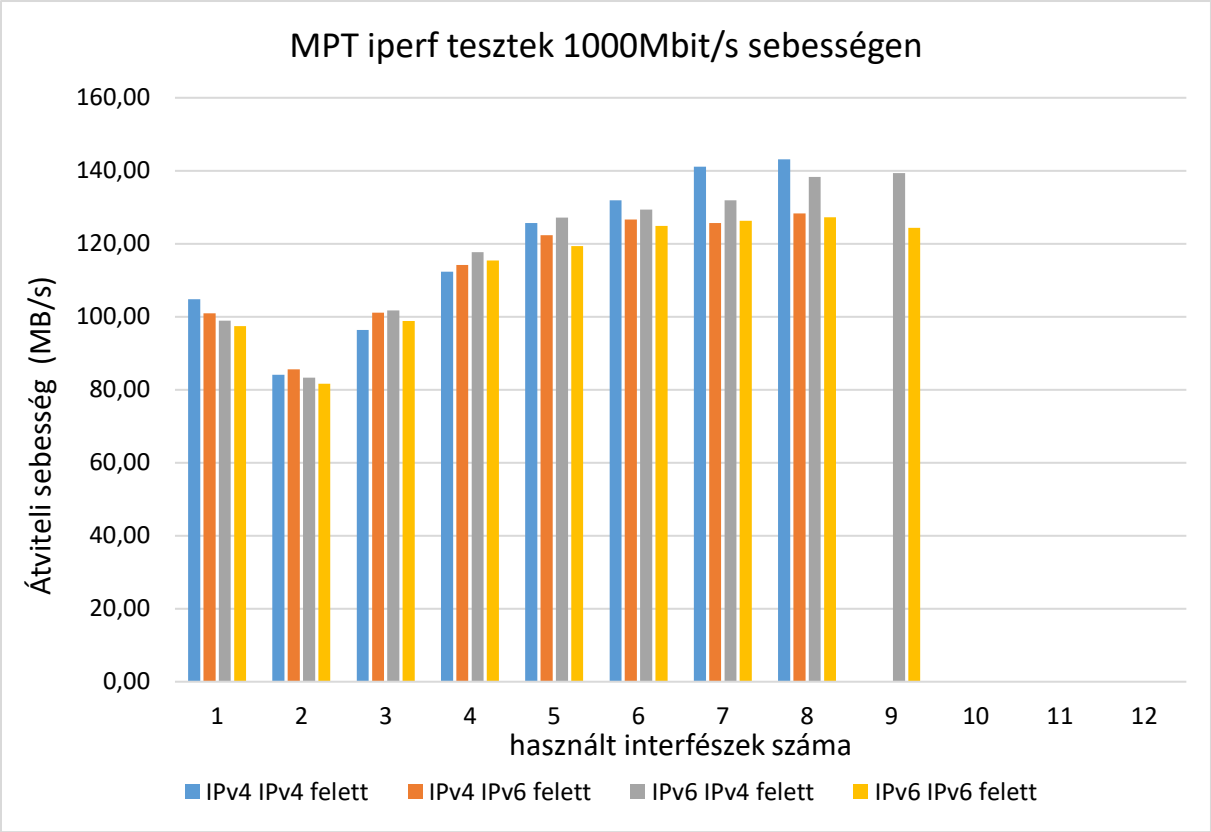
16. ábra. CPU foglaltság MPTCP esetén

Ahogy az a mérési eredményeinkből is láthatjuk, az MPCTP sokkal kevésbé CPU intenzív az átvitel során. Vizsgálataim során a MPT-hez viszonyítva közel fele annyi processzort használt. Itt ütközik ki a MPTCP kernel space előnye a user space-szel szemben, illetve az, hogy az MPTCP-nek nem szükséges GRE tunnelt fenntartani a kommunkiációhoz. Ugyanakkor természetesen 8-nál több fizikai interfészt nem képes aggregálni.

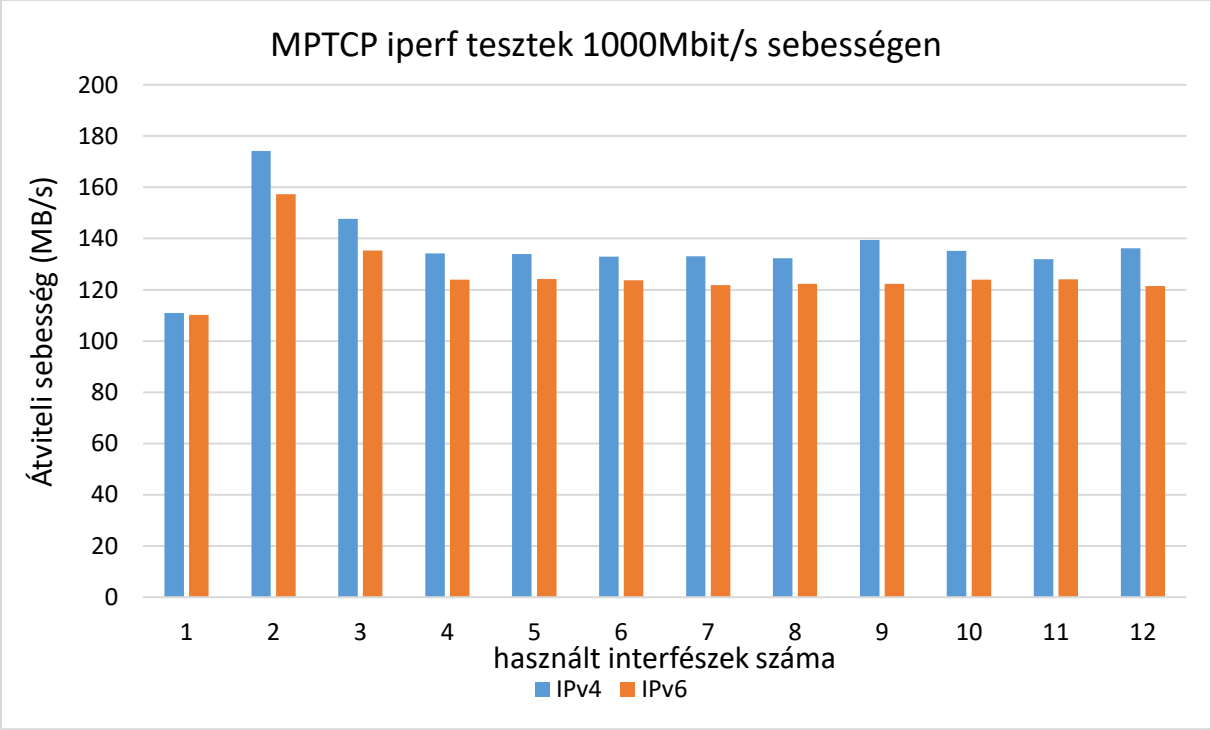
2.3.4 Mérések 1000Mbit/s sebességen

A 100Mbit/s sebességen végrehajtott sikeres mérések után a rendszert 1000Mbit/s sebességű méréskre állítottam be úgy, hogy a Cisco switch-et, mely a sebességek fix 100Mbit/s sebességre korlátozását végezte kivettem a rendszerből, és direkt összeköttetést biztosítottam a fizikai interfészek között. Mivel 100Mbit/s sebességen az iperf és a HTTP tesztek szinte azonos végeredményt mutattak, 1000Mbit/s sebességen csak az iperf teszteket végeztem el.

Minden egyes interfészt külön is leteszteltem, hogy biztosítsam minden interfész képes a közel 1000Mbit/s sebességre, majd újra futtattam. Ezek eredményei a következők:



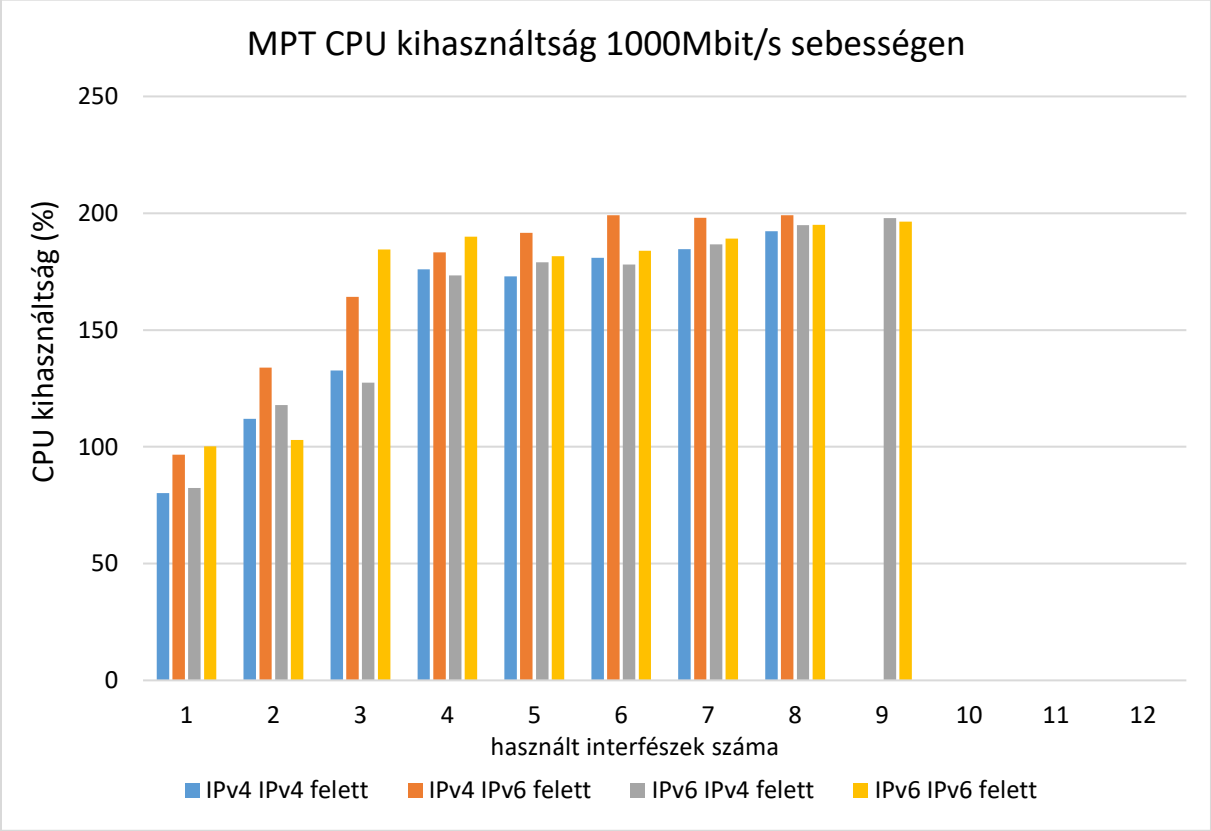
17. ábra. MPT iperf tesztek 1 Gbit/s sebességen



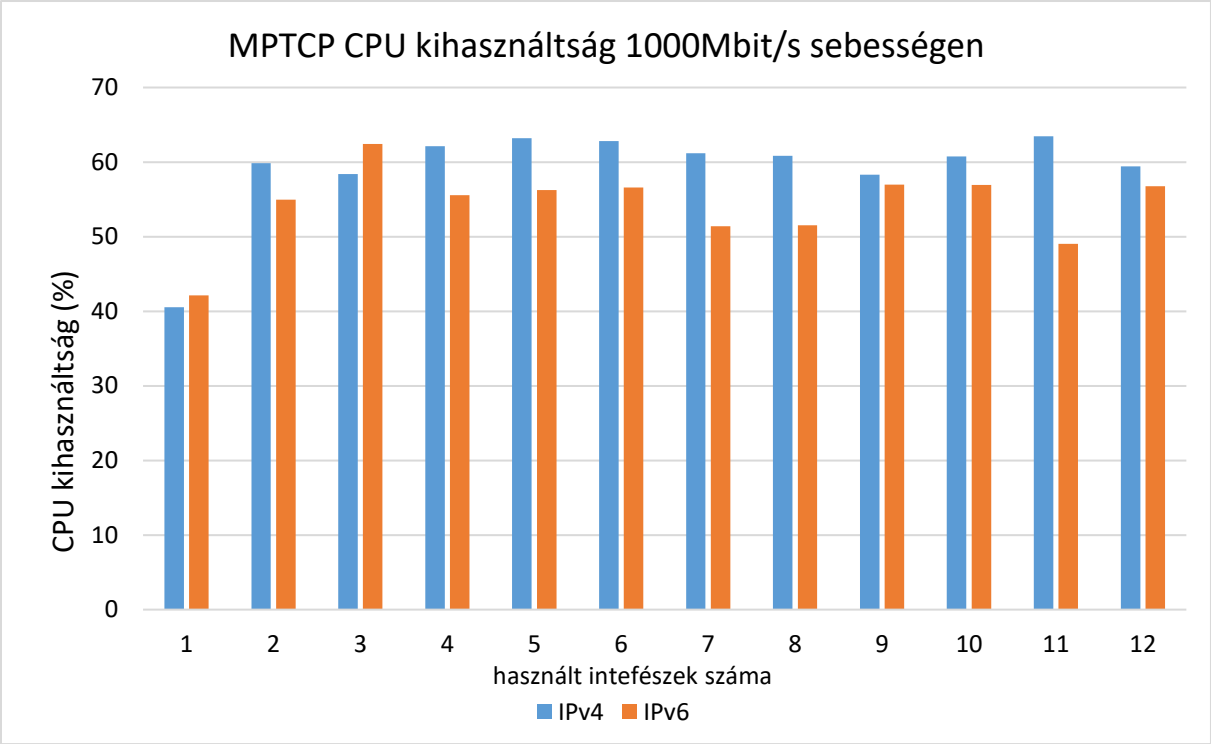
18. ábra. MPTCP iperf tesztek 1 Gbit/s sebességen

Mint az látható, az MPT nem képes kihasználni az 1000Mbit/s sebességű kapcsolatokat. Még ha csak a 2. fizikai interfészt adjuk hozzá az átvitelhez, akkor is messze alulmúlja a várt 2000Mbit/s sebességet. Ez betudható a mérőrendszer elavult processzorteljesítményének, erősebb hardverrel jobb eredmények is produkálhatóak. [26] Sőt, 2 fizikai interfész esetén még az 1000Mbit/s sebességet sem tudja elérni. Miután a 9. interfészt is hozzáadtam az méréshez, az MPT teljesen összeomlott. Az iperf 0.06 Mbit/s adatátviteli sebességet mért, ami teljesen elhanyagolható. A logokban eközben számtalan hibaüzenetet generált, melyek jelezték, hogy a GRE tunnel eldobálja a csomagokat, mert vagy nem jó sorrendben érkeztek, vagy duplikáltak lettek.

Amennyiben az MPT elveszti a 9. fizikai interfész használatát, úgy a legnagyobb előnyét is elveszti az MPTCP-vel szemben. Az MPTCP mérési eredményeiből is látszik, hogy gyorsan elérte teljesítőképessége határát, így ezeket a méréseket is megismételtem a processzorterheltség monitorozása mellett.



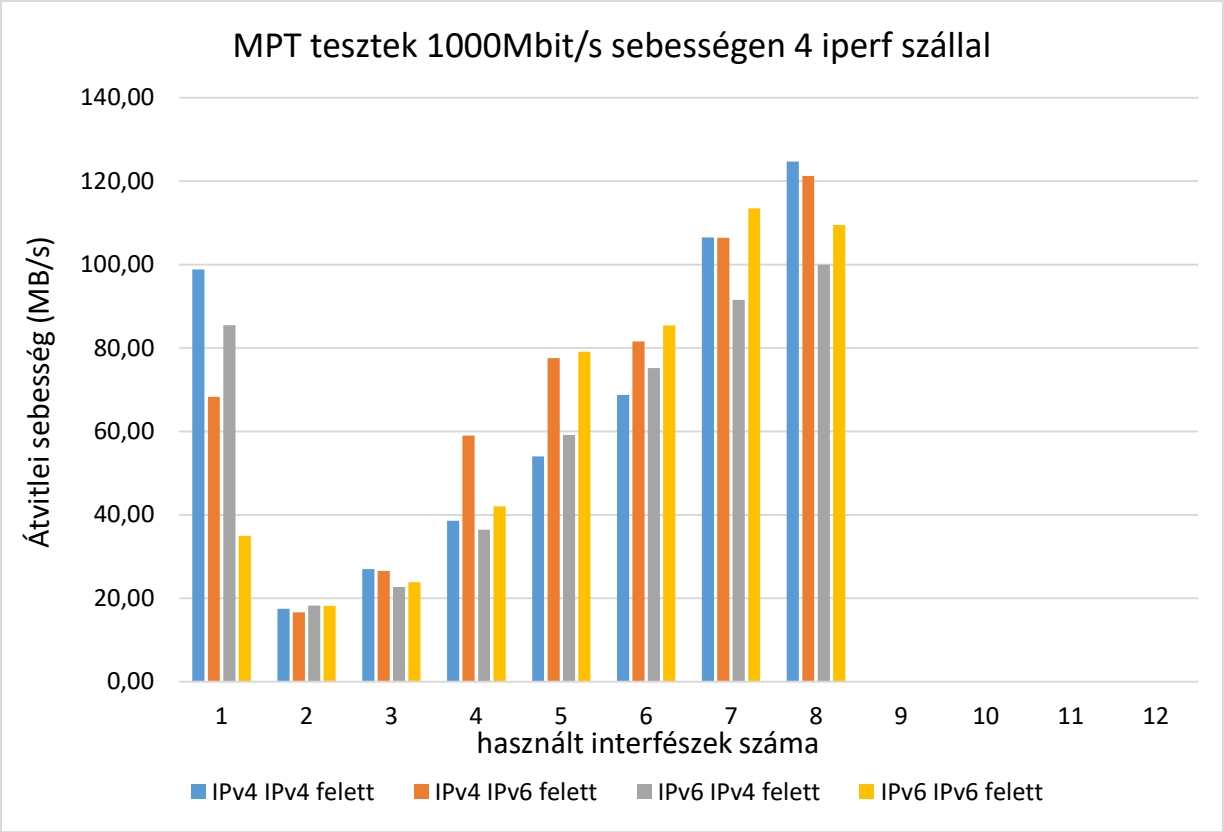
19. ábra. MPT CPU kihasználtság 1 Gbit/s sebességen



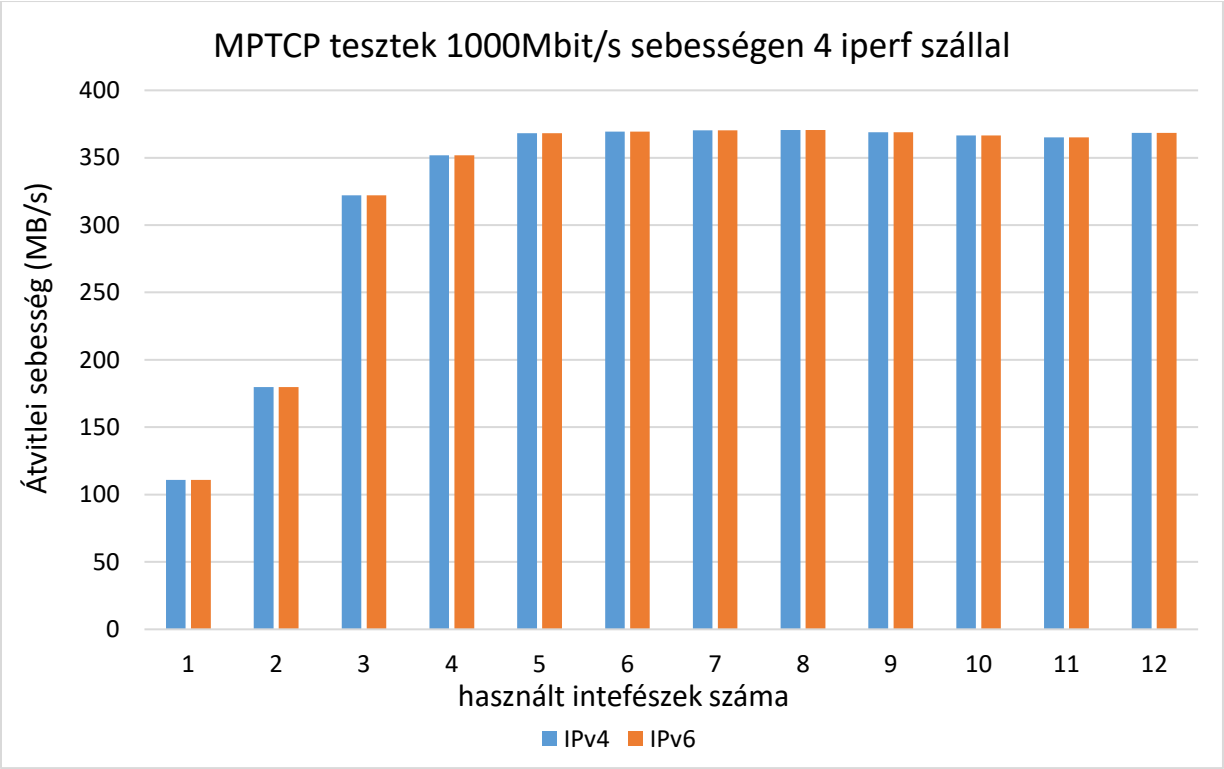
20. ábra. MPTCP CPU kihasználtság 1Gbit/s sebességen

Jól látható, hogy az MPT gyorsan felhasznál 2 teljes processzor magot, mely egyenértékű a 100Mbit/s sebességen mért 12 fizikai interfésznél használt processzormagokkal. Ez valamilyen implementációs hibára utal. Ha az MPT nem képes több processzormag egyidejű használatára, úgy komoly falakba ütközhet nagyobb sebességeknél is. Mivel az MPT nem párhuzamosítható megfelelően, így hiába van szabad kapacitás a rendszerben, nem képes azt kihasználni. Úgy gondolom, ezen a téren kell erősíteni az MPT implantációját, hisz a modern processzorok általában nem órajelben, hanem a processzormagok számában mutatnak erős fejlődést.

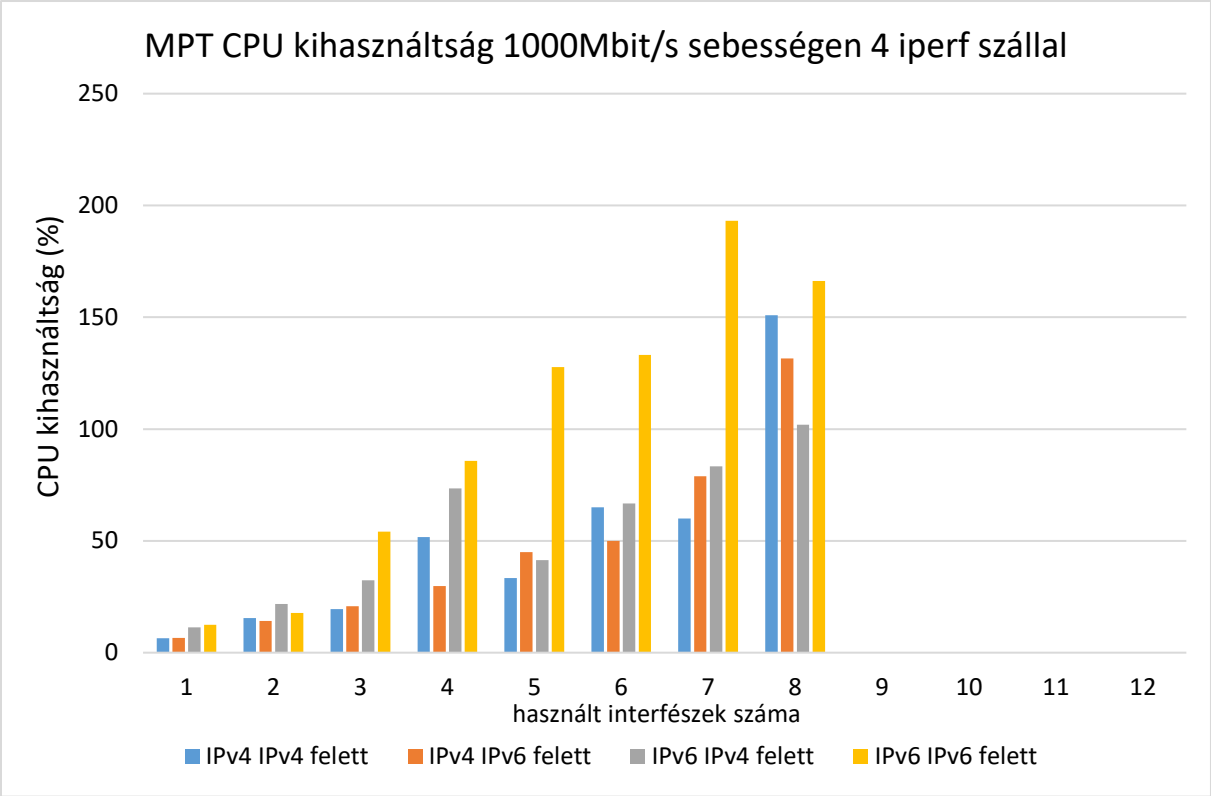
Az MPTCP is visszafogott teljesítményt mutatott, így a teszteket újra lefuttattam immáron az iperf-et 4 párhuzamos szálon futtatva.



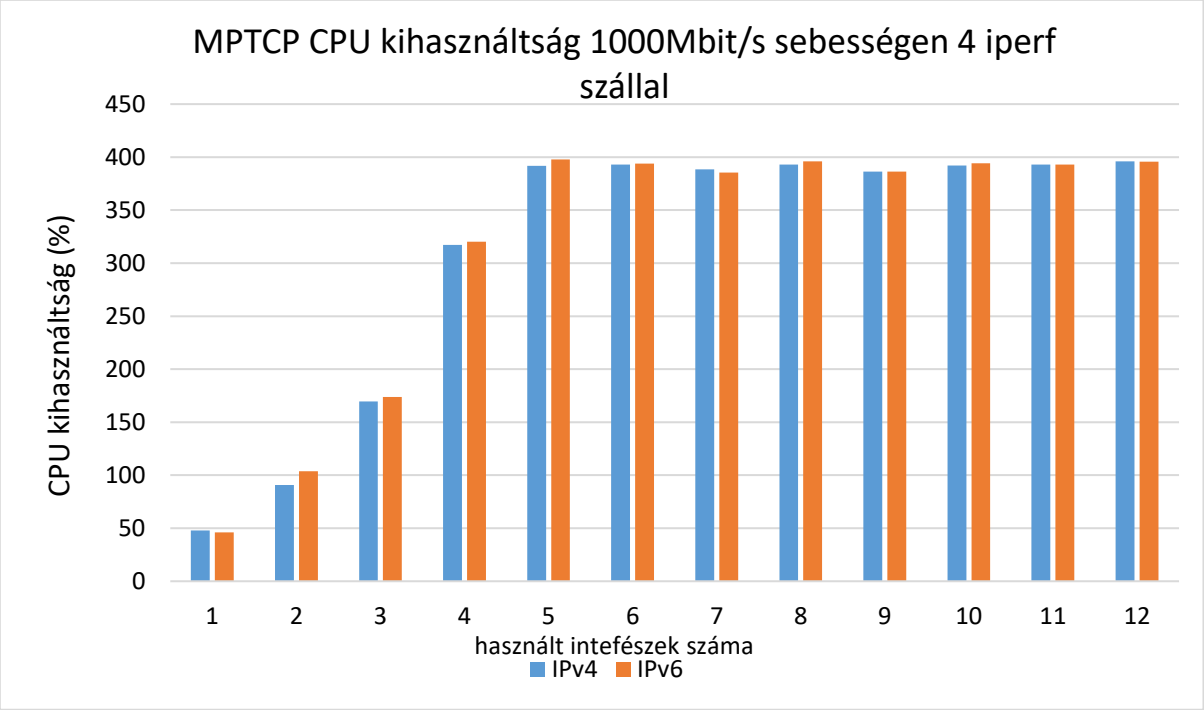
21. ábra. MPT tesztek 1Gbit/s sebességen 4 iperf szállal



22. ábra. MPTCP tesztek 1Gbit/s sebességen 4 iperf szállal



23. ábra. MPT CPU kihasználtság 1Gbit/s sebességen 4 iperf szállal



24. ábra. MPTCP CPU kihasználtság 1Gbit/s sebességen 4 iperf szállal

Látható, hogy mikor az MPT processzei, valamint a 4 iperf processz is az erőforrásokért küzd, úgy a MPT teljesítmény még inkább aggasztó. Amint elkezdünk több interfészt hozzáadni, teljesen kiszámíthatatlan átviteli mutatókat produkál. Ez mutatja, hogy az MPT-nek komoly teljesítménybeli problémái vannak, amint relatív nagysebességű környezetben használjuk. Hozzá kell tenni, hogy a méréseimhez használt hardverek viszonylag elavult technológiának számítanak, egy mai Intel Scalable processzornak, mely akár 40 fizikai processzormaggal is rendelkezhet [27] nem akadály ez a terhelés.

Ugyanakkor az MPTCP képes mind a 4 elérhető processzor mag kiaknázására 4 párhuzamos iperf segítségével, így az összes processzort kihasználva a 4 fizikai interfészt kihasználva a teoretikus 370MB/s sebességet is megközelíti. Itt a 24. ábrán jól látható, hogy a processzor volt a kritikus erőforrás, amíg volt szabad processzorteljesítmény, addig az MPCTP folyamatosan tudta hozni az elvárt eredményeket.

2.3.5 Az átviteli sebességek modellezése

2.3.5.1 Modellek 100Mbit/s adatátviteli sebességre

Az MPT sebessége és aggregációs képesége a teljes elérhető tartományban közel lineáris volt, míg az MPTCP csak 8 fizikai interfészig tudta ezt teljesíteni implementációs limitációi miatt. Az MPT (1) és az MPTCP (2) átviteli sebesség valamint a fizikai interfészek száma alapján a következő egyszerű matematikai módon modellezhetők:

$$T(\mathrm{n}) = \mathrm{n} \cdot T(1) \tag{1}$$

$$T(n) = \begin{cases} n \cdot T(1), if\ n \leq 8 \\ 8 \cdot T(1), if\ n > 8 \end{cases} \tag{2}$$

Ezekben az esetekben az átviteli sebességet nem limitálta a rendelkezésre álló processzorteljesítmény.

2.3.5.2 Modellek 1000Mbit/s adatátviteli sebességre

Az MPT átviteli sebességét és aggregációs képességeit iperf-et használva 1 és 4 szálon mértem meg. Látható, hogy az átviteli sebesség azonnal visszaesik amint hozzáadunk egy párhuzamos útvonalat. Ezek után további interfészeket hozzáadva folyamatos növekedésbe kezd egészen addig, míg a processzorteljesítmény el nem fogy, mely után az MPT összeomlik.

Az MPT 4 iperf szálon mérve ismét drasztikus teljesítménycsökkentést produkál. 1 fizikai interfészt még teljesen ki tud aknázni, ám utána folyamatosan küzd. Az MPT (3) matematikai modelljéhez lineáris regressziót alkalmaztam.

$$T(n) = \begin{cases} T(1), & if\ N = 1 \\ \alpha \cdot \mathrm{T}(1) + n \cdot \beta \cdot \mathrm{T}(1), & if\ 2 \leq n < 8 \\ 0, & if\ n \geq 9 \end{cases} \tag{3}$$

Az MPTCP átviteli függvénye teljesen máshogy alakult 1000Mbit/s sebességen. Az MPTCP képes az 1000Mbit kiaknázására egészen 4 fizikai interfészig, mely során folyamatos lineáris gyorsulást eredményez, ezek után azonban konstanssá válik bármennyi interfészt is adunk hozzá. Az MPCTP (4) teljes sebességre 4 párhuzamos iperf szállal volt képes mely során hozta az elvárt teoretikus sebességet.

$$T(n) = \begin{cases} n \cdot T(1), & if\ n \leq 4 \\ 4 \cdot T(1), & if\ n > 4 \end{cases} \tag{4}$$

2.4 1. Téziscsoport

1. Összeállítottam egy tesztrendszert, melyben az egyes MultiPath technológiák aggregációs képességinek vizsgálatát végeztem el 12 db hálózati interfész felhasználásával. Mind az MPT mind pedig az MPTCP alkalmas sok hálózati interfész összekapcsolására, melynek segítségével az átviteli kapacitás jelentősen növelhető.
    a. Az általam összeállított tesztrendszerben 100Mbit/s kapcsolatok esetén az MPT 1-től 12 elemi átviteli út esetén képes az átvitelei utak kapacitását teljesen linárisan összegezni. Az átviteli utak és a tunnel IP verziója az MTP teljesítményére nincsen érdemi hatással. Az MPTCP csak 8 elemi átviteli útig képes az átvitelei utak kapacitását teljesen lineárisan összegezni. (Ennek oka tervezői döntés, az MPTCP nem képes 8-nál több elemi út kezelésére.)
2. Az MPT és MPTCP jelentős CPU terhelést okoz az átviteli utak kapacitásának bemutatott összegzése során. Mivel az MPT user space szinten valósul meg, így erőforrásigénye nagyobb, mint a kernel szinten üzemelő MPTCP-nek.
    a. Megmutattam, hogy a tesztrendszer processzorteljesítménye nem elég ahhoz, hogy ugyanezen aggregációs képességek Gigabit Ethernet kapcsolatok esetén is megvalósuljanak.
3. Méréseim alapján egy egyszerű matematikai modellt állítottam fel, melynek segítségével modellezhető az, hogy adott interfész szám és interfész sebesség mellett milyen átviteli kapacitásra számíthatunk az MPT és MPTCP implementálása után.

Téziseimet alátámasztó publikációk:

- G Lencse, Á. Kovács, Testing the channel aggregation capability of the MPT multipath communication library, *Proc. World Symposium on Computer Networks and Information Security*, 13-15, 2015
- G Lencse, Á. Kovács, Advanced Measurements of the Aggregation Capability of the MPT Network Layer Multipath Communication Library,

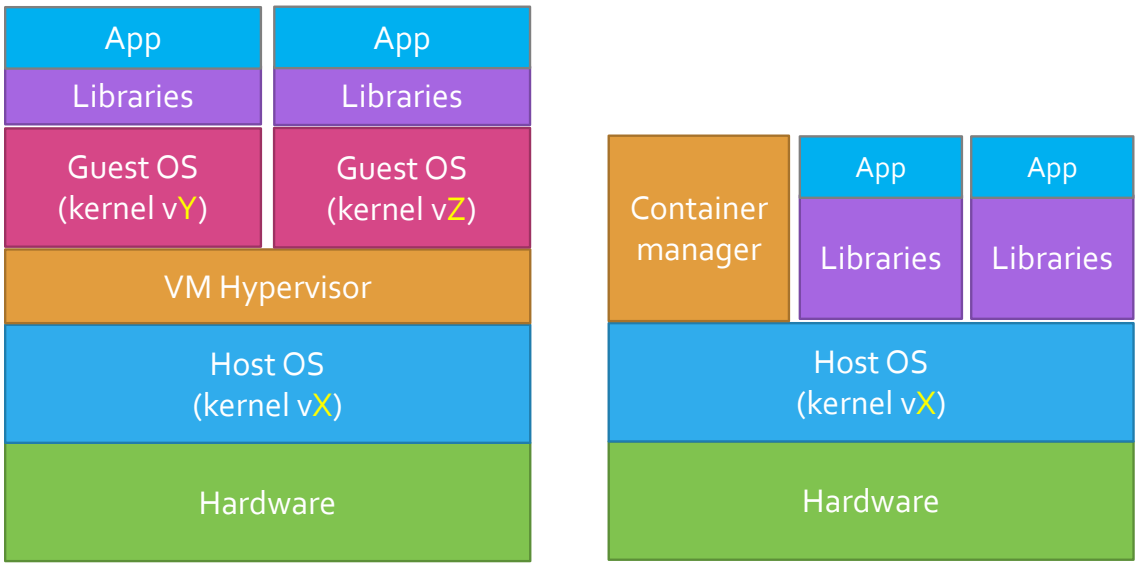
International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems, 2015, kötet 4, szám 2, pp:41-48

- Á. Kovács, Comparing the aggregation capability of the MPT communications library and multipath TCP, *In Proc. of 2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom),* pp: 157-162
- Á. Kovács, Evaluation of the Aggregation Capability of the MPT Network Layer Multipath Communication Library and Multipath TCP, *Acta Polytechnica Hungarica, 2019, Kötet 16, szám 6,* pp: 129-147

3 Konténer technológiák

Az elmúlt néhány évben a konténer technológia lett az egyik legelterjedtebb kifejezés a modern IT világában, gyakran a virtualizáció következő lépcsőfokának, evolúciós ugrásának is nevezik, bár közel sem egy újkeletű ötleten alapul.

A virtuális infrastruktúrák széles körben elterjedtek, rugalmasságuk valamint kezelhetőségük miatt. Jelenleg is az összes cloud erőforrás virtuális gépeken alapul, ugyanakkor a virtuális gépek viszonylag nagy többleterőforrást igényelnek, mivel minden egyes virtuális gép külön-külön kernel-el rendelkezik, melyek erőforrást vesznek el a ténylegesen futtatott applikációktól, szolgáltatásoktól. A hypervisor alapú virtualizációnál természetesen megvan az az előny, hogy bármilyen operációs rendszert futtathatunk rajtuk. A konténerek ezzel ellentétben a hoszt gép kernelét használva férnek hozzá az erőforrásokhoz, míg a erőforrások menedzsmentje, valamint a szeparáció más szinten valósul meg. Ugyanakkor olyan üzemeltetési megoldások, mint a live migration, a virtuális gépek kiváltsága marad, mert a konténerek mozgathatósága csak kikapcsolt vagy szüneteltett módon lehetséges. Másik oldalról viszont, mivel nincs minden konténernek saját kernele, így az elindításuk, migrálásuk másodpercek alatt végbemegy, így könnyen reagálhat a rendszer az esetleges erőforrások ugrásszerű igényváltozásaira. A két architektúra különbségét jól ábrázolja a következő blokkvázlat.



25. ábra. Hpyervisor kontra Konténer alapú virtualizáció

Az első Linux processz szeparáció 1979-ben mutatkozott be a chroot bevezetésével mely a Unix V7-es rendszerekben jelent meg. [28] Ennek segítségével egy olyan könyvtárszerkezetet hozhattunk létre, melybe egy processz minden függőségét bemásolva elszeparálhattuk a fájlrendszerünk többi részétől úgy, hogy ahhoz root jogosultág hiányában nem férhetett hozzá az adott processz. Gyakorlatilag egy új „/” könyvtárat definiáltunk számára. Ugyanakkor a zárt folyamat semmilyen más módon nem volt elszeparálva a többitől, így a memórián és a hálózaton keresztül ugyanúgy kihasználhatta a rendszer valamely sérülékenységét.

A 2000-es évek elején a FreeBSD Jail bevezetésével a szeparáció új foka jelent meg, mely során teljesen elszeparálhatóvá váltak a processzek egymástól, sőt a Linux bridge-en keresztül az egyes processzek saját IP címmel is rendelkeztek a szeparáció emelése érdekében. Ezen „börtön” már memóriában és processzorban is szeparált környezetnek számított, de összeállítása és kezelése kézzel zajlott. [28]

2004-ben a Solaris bevezette a Solaris Containers technológiát, mely a Jail technológiát a ZFS fájlrendszerrel kombinálva lehetőséget nyitott ezen elszeparált rendszerek snapshot-olására és klónozására. Ez nagyban segítette a kezelhetőség növekedését, valamint terjedését, de a ZFS 2004-ben még csak a SUN Solaris kiváltsága volt.

2006-ban a következő mérföldkőhöz érkezve a Google bemutatta a Process Containers technológiáját, melyet arra fejlesztettek, hogy egy processzt teljes egészében szeparáljon, és minden szinten limitálhasson a Linux Kernel (CPU, diszk I/O, memória, hálózat). Később a technológiát átnevezték Control Groups-ra (Cgroups) és bekerült a 2.6.24-es mainline kernelbe is. Ezzel megteremtve tulajdonképpen a modern konténerizációs technológiák alapját. [28]

2008-ban az LXC (LinuX Containers) az első teljesnek nevezhető konténer implementáció, mely a cgroupst és Linux Namespace-t használva saját managert alkalmazva megvalósította a konténer technológiák jelenleg ismert működését, úgy, hogy a Linux kernelben semmilyen módosításra nem volt szükség.

2013-ban jelent meg a Docker első verziója, mely a konténerek robbanásszerű elterjedését okozta. Ez is az LXC-n alapul, ám számos módosítást végeztek a konténer

menedzsment területén melyet később át is neveztek a saját libcontainer megoldásukra. Ez az ötlet adta az első teljes megoldást a konténerek létrehozására, menedzselésére valamint megosztására is a docker-hubon keresztül. [29]

3.1 Docker

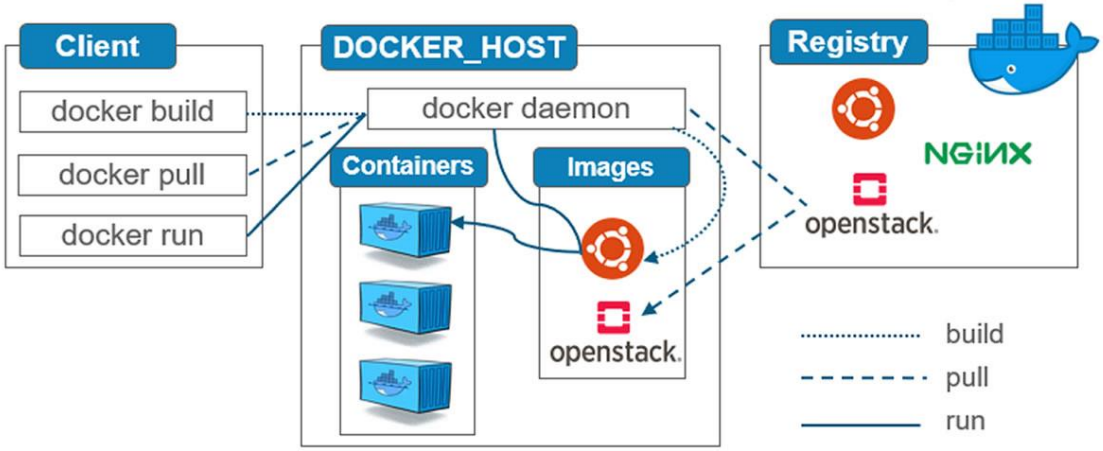
Az elmúlt évtizedben egyre inkább szükség volt egy olyan virtualizációs megoldásra, mely képes nagysűrűségű szolgáltatásokat futtatni, könnyen méretezhető és biztonságos. Erre kiválóan alkalmasak a konténer megoldások, melyek közül a legelterjedtebb a Docker ökoszisztéma. Ennek segítségével akár egy nagyságrenddel több virtualizált szolgáltatást futtathatunk ugyanazon hardveren, mint a hypevisor típusú virtualizációval. [30]

A Docker egy open-source-ként elérhető konténer technológia, mely képes elosztott alkalmazások telepítésére és üzemeltetésére. Olyan nagy cégek használják mint az Ebay, a Yelp, vagy a Spotify. Népszerűségét első körben a kiváló interfészelésének köszönheti, mely segítségével könnyen kezelhetőek és építhetőek konténerek. Másik nagy előnye, hogy a fejlesztők akár előre létrehozott konténereket is használhatnak, felgyorsítva ezzel a fejlesztés és tesztelés menetét; mindezt úgy, hogy a teszt és produktív környezetben ugyanazon szoftverbázis és konténer módosítás nélkül használható. [31] Végül de nem utolsó sorban a Docker több nagyobb third-party szolgáltatóval is együttműködve képes a konténerek menedzselésére külső eszközökkel is. A legelterjedtebb DevOps tool-ok, mint a Puppet, Ansible, és a Vagrant mind-mind támogatják a Docker konténereket. Ezek segítségével ilyen konténerek könnyen telepíthetőek és üzemeltethetőek a legtöbb Cloud szolgáltató platformján. A legtöbb orchestration tool, mint a Mesos, Shipyard és a Kubernetes is támogatja a Docker konténerek menedzselését.

3.1.1 Docker Engine

A Docker Engine a konténer alapú virtualizációnak egy „pehelykönnyű” és hordozható csomagolása. A Docker architektúrája, mely megegyezik az általános

konténer architektúrákkal, két fő komponensből áll, az egyik a docker-engine maga, míg a másik kiegészítő szolgáltatások és konténerek megosztására szolgáló docker-hub. A konténerek a Docker Daemon felett futnak, mely felelős a konténerek futtatásáért és menedzseléséért. A Docker kliens, a felhasználó által kiadott parancsokat egy RESTful API-n keresztül a Docker Daemon-nak továbbítja. Ezen felépítés lehetővé teszi, hogy a Docker engine és a Docker kliens akár más-más hoszton is lehet. [32]



26. ábra. Docker Arhitektúra [33]

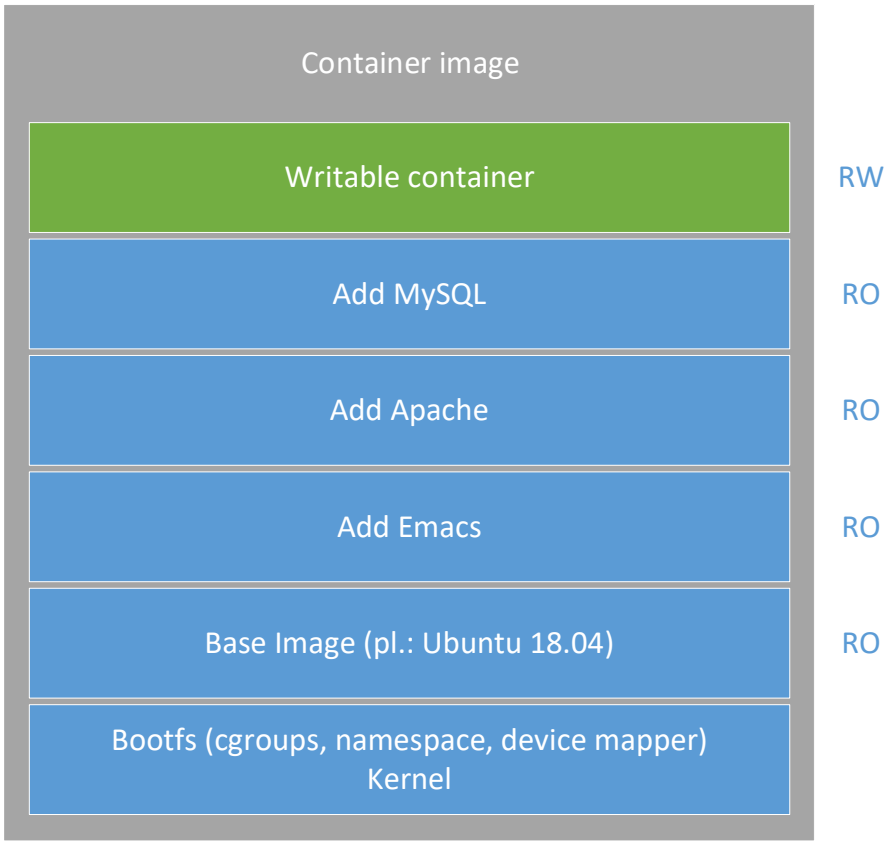
3.1.2 Docker konténer

Kezdetben a Docker az LXC-t használta konténer platformként, majd a 0.9-es verziótól kezdve a saját fejlesztésű `libcontainer` motort használja. A Docker két fő alappillére a cgroups és a namespace. [33], [34]

A cgroups felelős azért a mechanizmusért, mely a konténerben futó applikációnak megadja, hogy mely erőforrásokhoz és milyen mértékben férhet hozzá, és ezeket mérni is képes. A namespace elfedi az operációs rendszertől az erőforrások egy részét, azt az illúziót keltve, hogy a konténerben futó processzeknek saját, más processzekkel nem megosztott erőforrásuk van. Ezek a namespace-ek: Processz ID

(PID), hálózat, Inter-processz kommunikció (IPC), Mount (MNT) és Unix Timesharing System (UTS) [35]. Mindegyik különböző erőforrásokért felelős, például a hálózati Namespace felelős a hálózati erőforrásokért, mint az IP cím, routing táblák, melyek segítségével minden konténernek saját IP címe és network stackje lehet.

A Docker konténer másik nagy előnye a docker image-ek, vagy képfájlok felépítése. Minden konténernek mielőtt futtatnák, képfájlként már létezni kell a docker engine-ben. Ez a képfájl nem más, mint csak olvasható adatrétegek sorozata, melyeket egymásra helyezve adják a végleges képfájlt.



27. ábra. Adatrétegek a konténer image-ben.

Ez a felépítés nagyban megkönnyíti az image-ek és azok részeinek terjesztését, hisz minden ilyen réteg a base image-hez képest csak a változásokat tartalmazza. Ahhoz, hogy ezt egy imageként lehessen használni, a Docker egy úgynevezett UnionFS-t használ, mely támogatja a különböző rétegek egyként kezelését. [31], [34]

3.1.3 Docker Hub

A Docker Hub egy központi tükörszerver, mely a publikus és privát Docker image-eket és adatrétegeket tárolja. Itt lehetőségünk van az általunk generált és testreszabott image-eket és rétegeket megosztani. A Docker kliens segítségével lehetőségünk van mások által publikált docker konténereket keresni és letölteni. Továbbá lehetőségünk van a letöltött adatok integritásának ellenőrzésére is, mivel a Docker aláírja és verifikálja a Hubra feltöltött adatokat. [31]

3.2 Singularity

A hordozható környezetek az iparágak nem minden szegmensében jelentek meg azonnal. A virtualizációban élenjáró iparágak számára a konténertechnológiák által kínált megvalósítás és szolgáltatáskészlet összhangban van a vállalati mikro-szolgáltatási, virtualizáció és a webalapú felhőalkalmazások iránti igénnyel. A tudományos világ, és különösen a HPC közösségek számára ugyanaz a technológia nem minden esetben használható. A 2015 eleji reprodukálhatósági válság [36], [37], miatt ezek a kutatói közösségek különösen vágytak a konténeralapú alkalmazásokra, amelyek hosszú távú sikere a tudományos eredmények és a számítási környezetek reprodukálhatóságán múlik. Ezeknek a csoportoknak a Docker telepítése valamilyen HPC-környezetre indokolatlan szintű biztonsági kockázatot jelentene, mivel a Docker daemonnak mindenképpen privilegizált módon kell futnia az összes HPC elemen. Így gyorsan nyilvánvalóvá vált, hogy ez a két közösség, annak ellenére, hogy vannak közös vonásai, olyannyira különbözik egymástól, hogy egy azonos konténer megoldás semmilyen módon nem jöhetett létre.

Ennek ellenére a HPC közösség nem mondott le a konténerizáció előnyeiről. Néhány tudományos csoport mint a genetika [38], az idegtudomány [39] és mások [40] használják a konténerizációt, mint megoldást a tudományos reprodukálhatóságra.

Ahogy a kutatók elkezdték felfedezni a tudománynak is hasznos konténereket, megnőtt az igény, hogy ezeket a konténereket széles körben is használják. A kutatóintézetek által biztosított HPC-környezetek egyre több kérést kaptak, hogy

lehetővé tegyék a kutatóknak konténereik üzemeltetését. A közösség erre a cél érdekében tett erőfeszítések áradatával válaszolt, nevezetesen a nemzeti laboratóriumok részéről, köztük a Los Alamos National Lab (LANL) a CharlieClouddal [41], a National Energy Research Scientific Computing (NERSC) a Shifterrel. [42] A konténer alapú környezetek technológiai innovációja, a méretezhető és reprodukálható termékek iránti igény, a használhatóság preferálása, valamint az együttműködés szükségessége a laptopoktól a nagyméretű HPC-erőforrásokig mindenben meghatározza jelenlegi helyzetünket. Az adatfájlokat és a forráskód véletlenszerű bitjeit, azokat a formátumokat, amelyek korábban a tudományos munkafolyamatok cseréjénél az érdeklődésre számot tartó formátumok voltak, szoftverrel váltották fel, hogy ezeket manipulálják (SD, Software defined). A végfelhasználó számára a határvonalak elmosódnak a felhő és a helyi számítási környezet között: mindkettő olyan gép, amelyhez a felhasználónak csatlakoznia kell, és amely bizonyos mennyiségű memóriát és lemezterületet kínál. Ettől függetlenül nagy szükség van egy konténeres megoldásra, amely nem veszi figyelembe ezeket a részleteket, és zökkenőmentesen mozoghat a két opció között.

A Singularity konténer ökoszisztémát kifejezetten HPC környezete és a tudományos közösség igényeit szem előtt tartva fejlesztették. Segítségével a különböző tudományos folyamatok és szoftverek módosítás nélkül képesek általános HPC rendszereken futni. Legnagyobb előnye a tudományos közösségek által igényelt reprodukálhatóság, könnyű megosztás. Ennek segítségével képesek vagyunk olyan HPC alkalmazások konténerizálásra, melyek MPI és egy elosztott számítási rendszert használnak, valamint minden konténer az őt indító felhasználó jogosultságaival rendelkezik, és képes hozzáférni olyan speciális hardverekhez, mint az Infiniband valamint a GPU gyorsító kártyák. [43]

A Docker-el ellentétben a Singularity egy nagy image fájlt generál, mely alapértelmezetten Read-only. Nagy rugalmasságát adja, hogy képesek vagyunk nem csak image fájlba, hanem egy kijelölt könyvtárba „sandbox” módban is egy konténer alapját képező könyvtárstruktúrát generálni, ez a későbbi kisebb módosításokat segíti elő, valamint képes a Docker Hubról is letölteni adatrétegeket, amelyeket aztán egy image fájlba egyesít.

4 Konténerek összehasonlítása

Ahhoz, hogy az egyes feladatokra a legalkalmasabb eszközt válasszuk, szükséges az elérhető technológiáinak összehasonlítása. Ebben a fejezetben a jelenleg elérhető és népszerű konténer technológiák teljesítmény metrikáit hasonlítom össze natívan és Virtualizálva (KVM) futtatott tesztek segítségével.

Az irodalomban számos publikáció létezik már a különböző konténer megoldások összehasonlítására, mint például Apache Hadoop teljesítménytesztek [44], vagy éppen Wordpress és MySQL teljesítménytesztek [45]. Vagy akár a nagy HPC tesztelésekre is használt LINPACK benchmark tool, melynek segítségével állítják össze évente a legerősebb HPC-ket felsoroló listát a top500.org-on [46]. Ezek a mérések és összehasonlítások mind-mind hozzájárultak ahhoz, hogy tisztább képet kapjuk a konténer technológiák teljesítmény tesztjeiről, de a technológia fejlődésével újabb és újabb kihívókat kapnak a már meglévő technológiák, mint pl.: a Singularity konténer. Ugyanakkor a jelenleg rendelkezésre álló szakirodalomban nem található tisztán a hálózati teljesítmény tesztekre vonatkozó mérés.

4.1 Mérési összeállítás

Egy összetett IT infrastruktúrában 4 erőforrást lehet komolyabb tesztek alá vonni: CPU, Memória, Háttértár (I/O) valamint hálózati teljesítmény. Ezek közül a legfontosabbak a konténer technológia architektúrája miatt főleg a CPU valamint a hálózat. A háttértár tesztelésénél valószínűleg a háttértár sebességét mérnénk, így nem lenne különbség a natív valamint a konténer technológia között. A memória hozzáférést szintúgy a hoszt kernelén keresztül zajlik, így különbséget itt sem tapasztalnánk.

A processzor teljesítményt ugyanakkor érdemes megvizsgálni, hisz a hoszt kernelen keresztül a namespace-ek és cgroup rendszeren keresztüli processzorhívásokon lehetnek olyan esetek melyek a processzorhasználat teljesítménymutatóit befolyásolhatják. Ugyanez a helyzet a hálózattal. Mivel a konténer technológiák

alapesetben Linux Bridge rendszeren keresztül saját network stacken keresztül kapcsolódnak a hálózathoz, mindenképpen javasolt ezen a teljesítménytesztek futtatása.

4.1.1 Teszt infrastruktúra

A konténer technológiákat egy HUAWEI CH121 V3-as blade szerveren futtattam. (h1) Ennek a specifikációi a következők:

- 2x E5-2630 V3 @2,4GHz processzor
- 128GB DDR4 @2333MHz
- 1000GB NL-SAS tárhely
- 4x10Gb Broadcom NIC

Szoftver verziók:

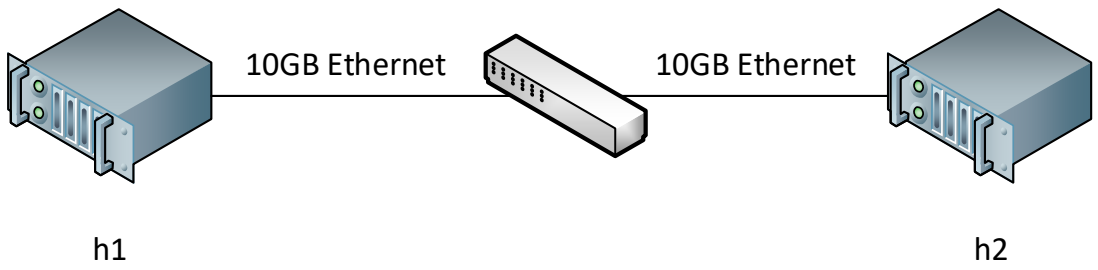
- OS: Ubuntu 16.10 (4.8 kernel verzió)
- Sysbench 0.4.12
- iperf 3.1.3

A hálózati mérésekhez egy másik szerverre is szükségem volt, melynek egy HUAWEI CH140 V3 használtam (h2) a következő specifikációkkal:

- 2x E5-2670 V3 @2,3GHz processzor
- 128GB DDR4 @2333MHz
- 1000GB NL-SAS tárhely
- 2x10Gb Broadcom NIC

Szoftver verziók:

- OS: Ubuntu 16.10 (4.8 kernel verzió)
- Sysbench 0.4.12
- iperf 3.1.3



28. ábra. Mérési összeállítás

Ezen két szervert egy HUAWEI CX310-es switch segítségével kötöttem egy hálózatba. Mivel ezek a szerverek blade rendszerűek, így nincs lehetőségem közvetlen kábeles összeköttetés létesítésére csak a Blade kereten belül található switchen keresztül.

A konténer technológiában lehetőségünk van definiálni a konténer base image-e segítségével a konténerben lévő OS verzióját, ezt minden tesztnél Ubuntu 16.04.1 LTS-re állítottam.

4.2 Konténerek létrehozása

A különböző konténer technológiáknál más-más úton lehet új konténereket létrehozni, vagy letölteni. [33]

4.2.1 LXC

Az LXC rendszerben az alapértelmezett mód az, hogy a LXC által támogatott konténerek base image-eit letölthetjük, de testre szabni, különböző szolgáltatásokat telepíteni magunknak kell. Egy iperf-et tartalmazó konténer létrehozása a következő képpen zajlott: [47]

```
$ lxc-create -t download -n ubuntu1 --dist ubuntu --release xenial -
-arch amd64
$ lxc-start --name ubuntu1 -daemon
$ lxc-attach -n ubuntu1
$ apt update && apt install iperf3
```

Ezek után már csak el kellett indítani a konténerben futtatott iperf3-at.

```
$ lxc start -n ubuntu1
$ lxc attach -n ubuntu1
(ubuntu1)$ iperf3 -s
```

4.2.2 Docker

A Docker konténer a legelterjedtebb konténer technológiának számít a nagy felhasználóbázisának és a Docker-Hub adta lehetőségeknek köszönhetően. A Docker-Hub segítségével a felhasználók megoszthatják egymással a saját testreszabott konténereiket, ugyanakkor lehetőségük van saját konténereket is létrehozni. Ehhez szintén definiálnunk kell milyen base image-el szeretnénk dolgozni, majd megadhatjuk milyen szoftverek és szolgáltatások települjenek konténerünkbe. Ehhez egy Dockerfile nevű fájlt kell létrehoznunk, mely egy recept a konténerhez. Természetesen van mód ilyen recept fájlok megosztására is, segítségükkel könnyen generálhatunk mások által definiált konténert, de saját magunk számára is testre szabhatjuk azokat. [33]

```
FROM ubuntu:16.04
WORKDIR /root/
RUN apt update
RUN apt install -y libssl-dev iperf net-tools
```

A fenti példában egy nagyon egyszerű dockerfile-t mutatok be, mely egy Ubuntu 16.04 base image-re épül, melybe iperf tool-t telepítünk. Ebből az image-ből aztán tetszőleges számú konténer kreálható.

4.2.3 Singularity

A Singularity-vel könnyű dolguk van, hiszen képes a saját leíró nyelvén konténert kreálni, de kompatibilis a Dockerfile ökoszisztémával, valamint akár Dockerhub-ról is tud előre definiált konténereket letölteni. Mivel a Singularity a Docker-rel ellentétben nem Read Only rétegeket használ, hanem egységes formátumú, akár tömörített image fájlt, így van lehetőségünk nem csak Read Only hanem írható konténereket, illetve könyvtárstruktúrát is létrehozni, melyek mérete növekedhet, később a benne létrehozott fájlrendszer tetszőlegesen módosítható, csomagok törölhetők és telepíthetők anélkül, hogy a konténert teljesen újra kelljen generálni. Ezt Sandbox módnak nevezzük. A

Singularity képes sandboxra kijelölt a könyvtárat használni teljes gyökérként (mint a chroot esetében) vagy ebből gyorsabban generálni image fájlt, hiszen a telepítéssel, szoftverek és alkalmazások fordításával már nem kell időt tölteni, csak a becsomagolással. A konténer definíciós fájljainak különböző dedikált részei vannak, melyek a következő szerepet töltik be:

```
Bootstrap # Itt adhatjuk meg milyen módon szeretnénk a konténer alapjául szolgáló rendszert összeállítani. Lehetőségünk van singularity vagy Docker hub-ról definiálni már meglévő konténert vagy sajátot generálni

From: # amennyiben valamilyen előre definiált konténert szeretnénk itt adhatjuk meg a disztribúciót és verziót.

     %labels # Itt adhatunk információt a Singularity konténerről, milyen szoftverek kerültek be, ezt később az inspect paranccal lehet lekérdezni.

     %files # Ebben a szekcióban van lehetőségünk a hoszt fájlrendszeréről fájlokat könyvtárakat bemásolni a konténerbe.

     %post # A telepítés főbb részei itt kapnak helyet, csomagok telepítése, szoftverek fordítása, fájlok módosítása valamint másolás és törlés is itt lehetséges.

     %environment # Itt beállítható milyen egyedi a hoszt-tól eltérő környezeti változók legyenek alapértelmezettek a konténerben.
```

Példa egy egyszerű Singularity kontainer receptre:

```
     Bootstrap: docker
     From: ubuntu:latest

     %post
     apt update
     apt install -y iperf3
```

4.3 Mérések

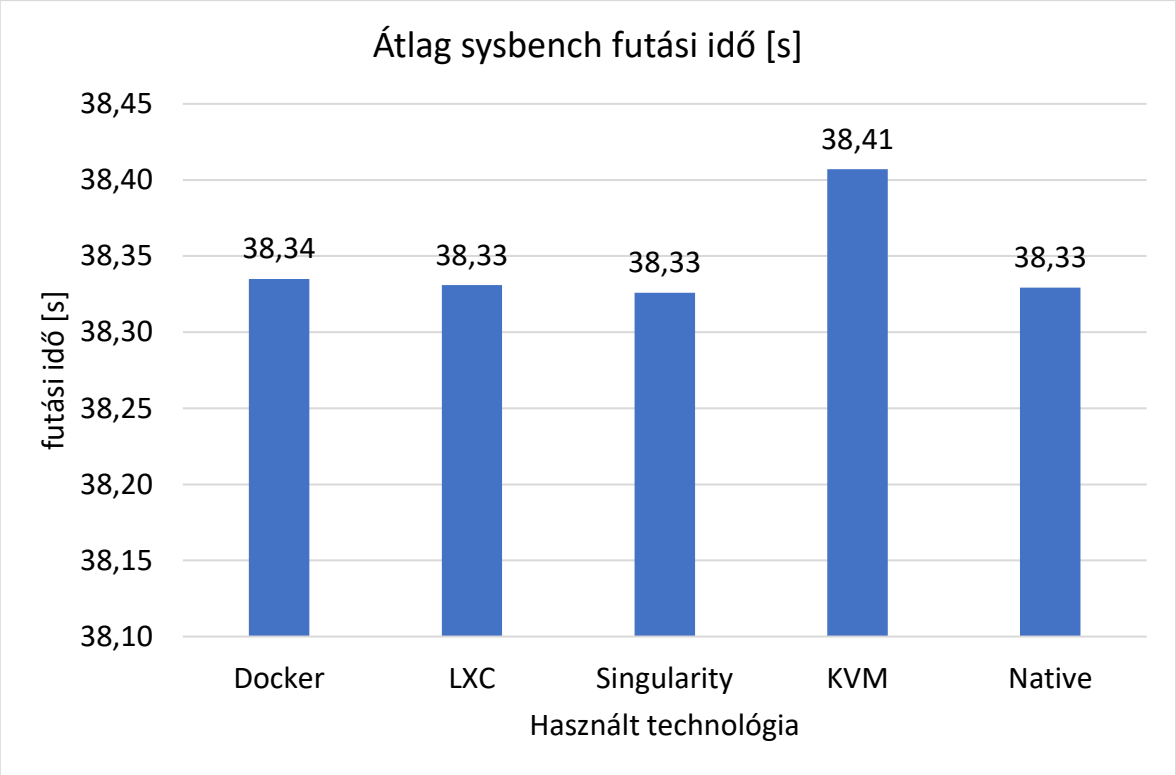
A CPU méréseknél először a natív, a KVM virtualizált valamint a különböző konténerek CPU teljesítményét mértem a sysbench szintetikus processzor tesztjével. Ennek során a szoftver prímszámokat keres a megadott maximális limitig a megadott magszámon. A virtuális gépnek 8 processzort adtam, és a sysbench-et is 8 szálon futtattam a következő paranccsal:

```
sysbench --num-threads=8 --test=cpu --cpu-max-prime=100000 run
```

Ezeket a méréseket 11 alkalommal végeztem el, majd átlagot valamint szórást számoltam. Az alábbi táblázat ezen eredményeket mutatja.

	Docker	***LXC***	***SINGULARITY***	***KVM***	***NATIVE***
ÁTLAG (S)	38,34	38,33	38,33	38,41	38,33
SZÓRÁS (S)	0,0093	0,0105	0,0086	0,0189	0,0076

1. táblázat. CPU teszt mérési eredmények



29. ábra. CPU teszt mérési eredmények

Látható, hogy az összes konténer technológia a natív futtatási sebességgel közel azonos eredményeket ért el. Egyedül a virtualizált megoldás (KVM) mutat lassulást, ugyanakkor jól látható, hogy az is elenyésző sebességvesztés tekintettel a virtualizáció nyújtotta előnyökhöz képest. A szórás minimális volt, vagyis az eredmények stabil rendszerre utalnak.

A hálózati teljesítménymméréshez az iperf tool-t használtam, mely méri a lehető legnagyobb elérhető adatátviteli sebességet a szerver és kliens között. Ehhez 2 eszközt kell használni, mely során az egyik a szerver a másik a kliens szerepét tölti be. Lehetőségünk van akár a szerepek, akár az adatátvitel irányát is megfordítani. Én a méréseket 60 másodpercig futtattam a kliens és a szerver között, mindezt úgy, hogy a szerver konténerizálva volt. Ezt a legegyszerűbb módon úgy tudjuk megoldani, hogy valamilyen már előre kész konténert töltünk le, és indítjuk el.

```
      docker run --name=iperf3 -d --restart=unless-stopped -p 5201:5201/tcp
-p 5201:5201/udp mlabbe/iperf3
```

A Linux ilyenkor a Docker telepítésekor létrehozott hálózatból választ egy címet a konténernek, majd a megfelelő portokat (TCP/UDP 5201) rámappeli a konténerből erre az IP címre. Ahhoz, hogy ezt elérjük, a kliens oldalon meg kell adnunk routingban, hogy az adott IP cím a konténert futtató hoszt-on keresztül érhető el. Ezeket az IP-ket aztán a hoszton futtatott Linux Bridge-en keresztül érhetjük el.

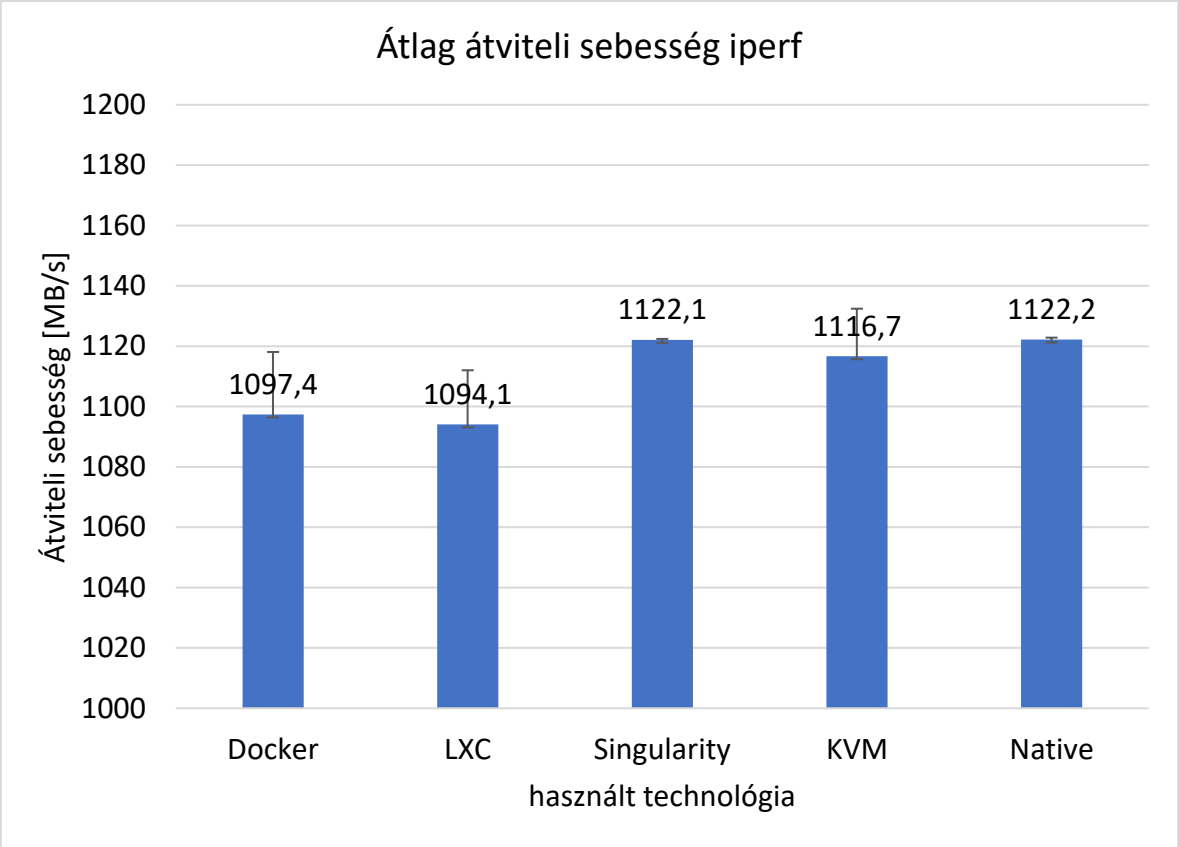
Ha ez megvan, akkor a kliensen a következő parancsot adjuk ki.

```
      iperf3 -i 2 -f M -c <IPerf Server IP> -t 60
```

A mérési eredményeket az alábbi táblázat tartalmazza.

	DOCKER	***LXC***	***SINGULARITY***	***KVM***	***NATIVE***
ÁTLAG (MB/s)	1097,4	1094,1	1122,1	1116,7	1122,2
SZÓRÁS (MB/s)	20,689	17,916	0,316	15,72	0,632

2. táblázat. Átviteli sebességek mérési eredménye

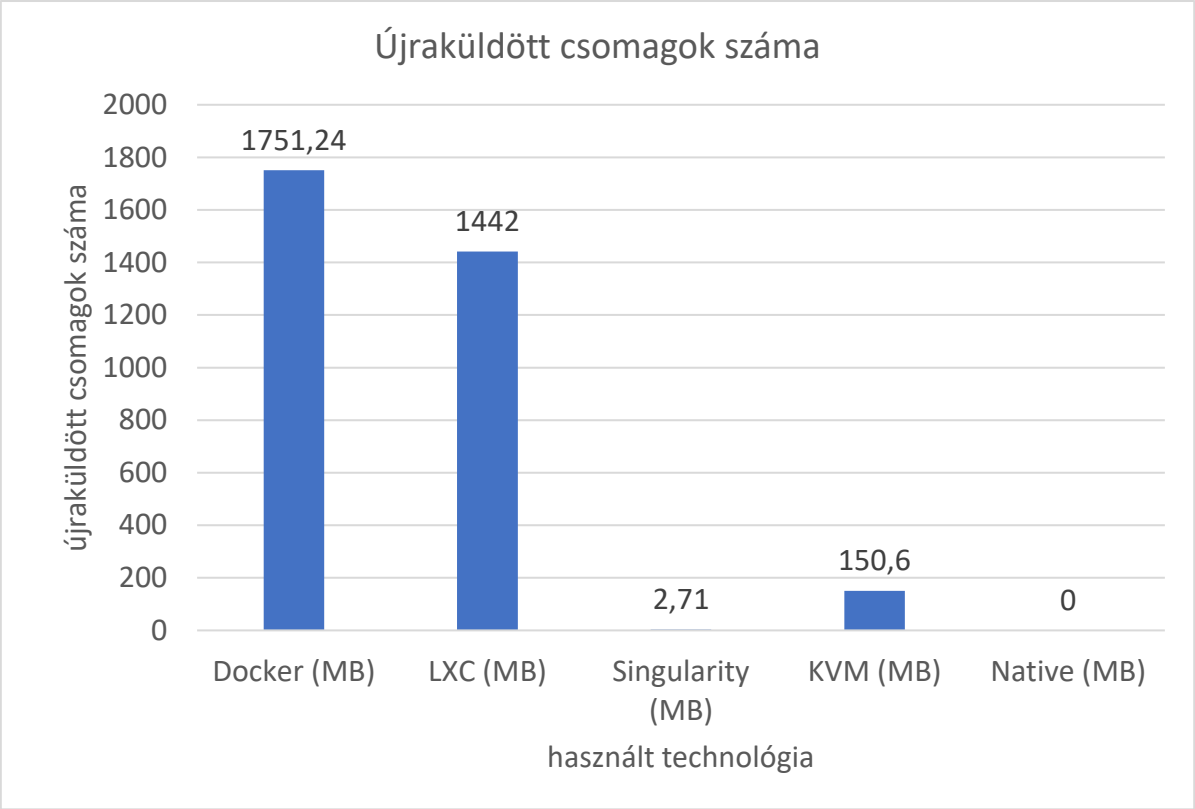


30. ábra. Átviteli sebességek mérési eredménye

A mérési eredményekből, jól látszik, hogy a Docker és LXC némiképp elmarad a többi megoldástól. A mérések eredményei alapján jól látható, hogy ez a két megoldás szórásban is kiemelkedik a többi megoldástól, vagyis az eredményeik kevésbé stabilak. A mérések során felfigyeltem az iperf által kijelzett magas retransmission rate-re is, mely arról árulkodik, hogy az iperf-nek sokszor újra kellett küldenie az egyes csomagokat, így alakult ki az alacsonyabb átviteli sebesség.

	Docker	***LXC***	***Singularity***	***KVM***	***Native***
Átlag	1751,24	1442	2,71	150,6	0

3. táblázat. Újraküldött csomagok száma iperf mérés alatt



31. ábra. Újraküldött csomagok száma iperf mérés alatt

Jól látható, hogy a Docker és az LXC az újraküldött csomagok számában is óriási hátrányban van. Érdekesség, hogy a KVM virtualizáció is Linux Bridge rendszert használ a hálózati forgalmának lebonyolítására, de így is tizedannyi újraküldött csomagot generált. Megfigyelhető továbbá, hogy mivel a natív futtatás 0 újra küldést generált, így kijelenthetjük, hogy a fizikai hálózati eszköz hibátlanul működik.

Ahhoz, hogy ezt a jelenséget megértsük, meg kell vizsgálnunk a konténerek hálózati stack felépítését.

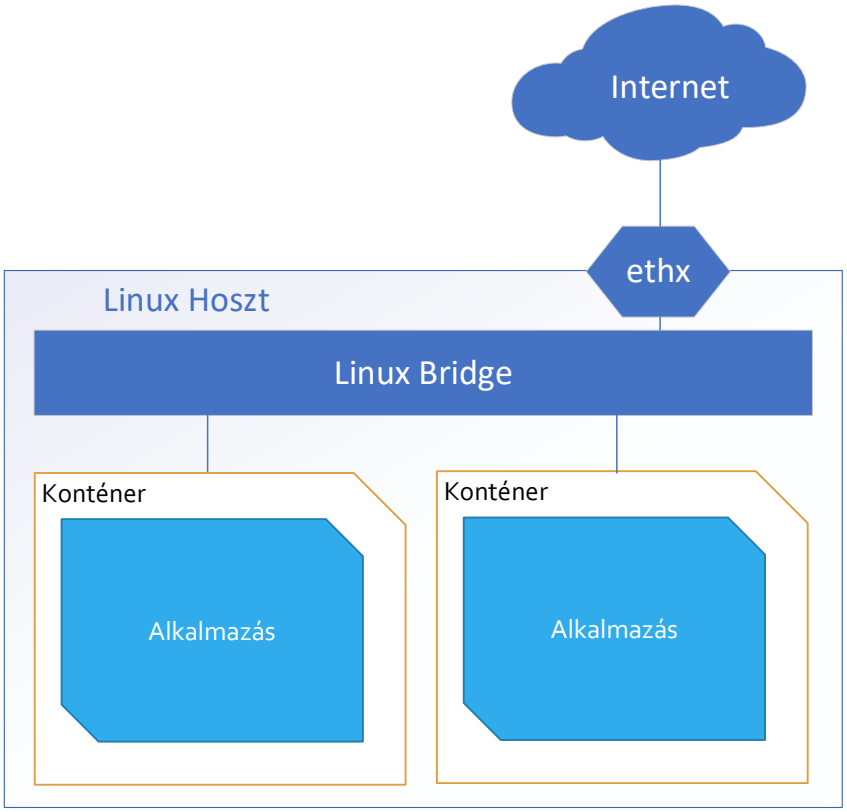
4.3.1 Leggyakoribb konténer hálózati üzemmódok

4.3.1.1 Hálózat nélküli üzemmód (None)

Neve alapvetően magába foglalja működését is. Olyan konténer, melynek nincs szüksége hálózati hozzáférésre. Ugyanakkor loopback device-szal rendelkezik. [48] [49]

4.3.1.2 *Bridge mód (Alapértelmezett)*

A Docker és az LXC alapértelmetetten a Linux bridge utilst használja a hálózat felépítésére. Ez gyakorlatilag annyit tesz, hogy a Linux egy szoftver switchet hoz létre, mely segítségével a konténerek hálózati interfészei elérhetik az ugyanazon hoszton futó más konténereket. Ehhez a hálózati címfordítást a Linux beépített IPtables toolja biztosítja, mely segít a hálózati izolációban is. [48], [49]



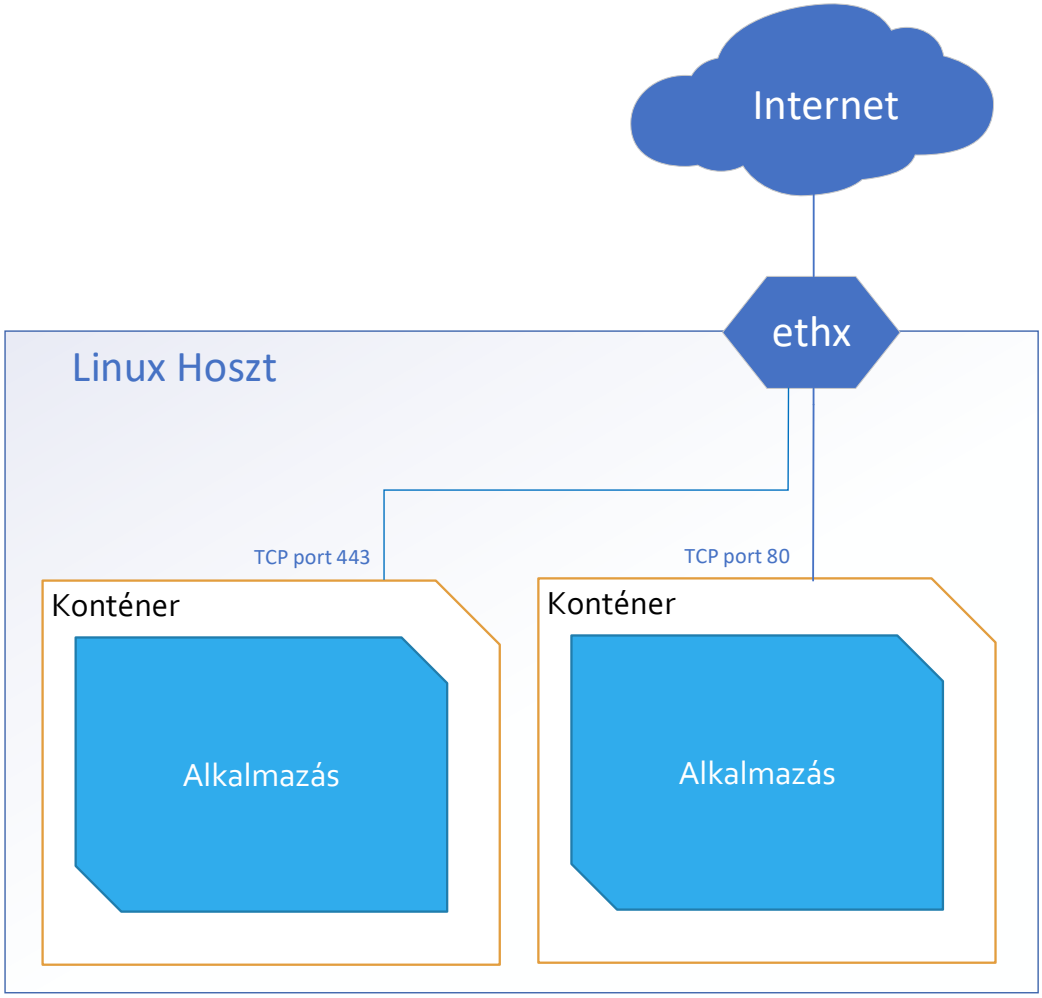
32. ábra. Alapértelmezett konténer hálózati blokkvázlat

Mivel a Linux Bridge egy CPU-n emulált hálózati eszköz, így annak teljesítménye nem determinisztikus, sok esetben függ a processzor állapotától és terheltségétől. Ezzel szemben minden konténer megoldásnál van lehetőségünk átállítani az alapértelmezett üzemmódot.

4.3.1.3 *Host mód*

Hoszt módnál a Konténer és a hoszt gyakorlatilag osztozik a fizikai interfészen. Ilyen módon a konténer képes a hoszt interfészébe beinjektálni saját csomagjait,

valamint hozzáfér az összes csomaghoz mely a hosztra érkezik. Ugyanakkor nem képes – hacsak nem privilégizál üzemmódban fut – konfigurálni azt. Emellett nagy hátránya, hogy a port számokon is osztozik a konténer és a hoszt operációs rendszer network stack-je, így pl.: ha konténerben futó apache2 webszerver figyel a hoszt 443-as portján, úgy a hoszt már nem képes használni a 443-as portot, illetve más konténer sem használhatja ezt. Ebben az esetben a konténereknek speciális hozzáférést kell biztosítani a fizikai eszközhöz. [48], [49]



33. ábra. Konténer hoszt mód

4.3.1.4 Overlay

Az overlay gyakorlatilag tunnel-eket alakít ki a kommunikációra az egyes konténerek között, akár több hoszt között is. Ennek eredményeként a konténerek egy

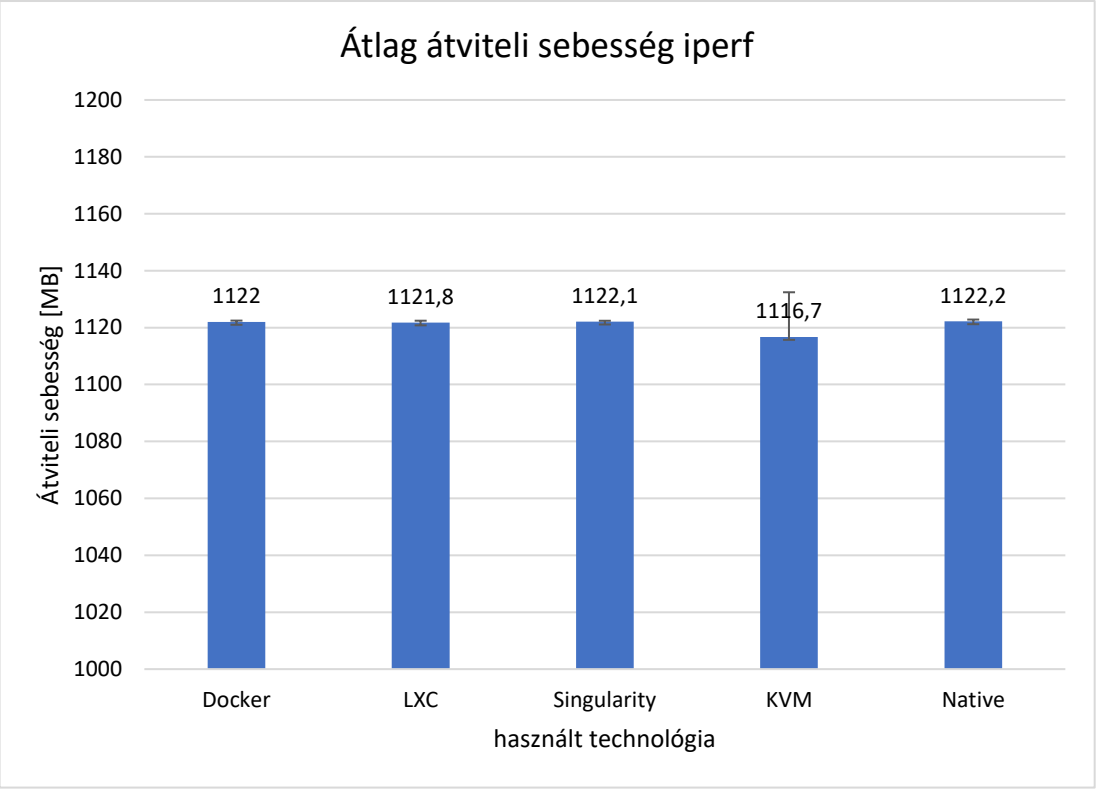
alhálózatban lehetnek úgy, hogy tunnelek segítségével szállítjuk az egyes alhálózatokat a konténerekhez. A Docker alapértelmezetten ehhez a rendszerhez a VXLAN technológiát alkalmazza. [48]

4.4 Hoszt mód mérési eredmények

Ezek alapján a méréseket újra elvégeztem, de már oly módon, hogy a hálózati módot host networkre állítottam. Ez úgy tudom elérni, hogy Docker esetében a konténert `--net=host` opcióval indítom el, így a hálózati namespace nem az alapértelmezett Linux Bridge lesz hanem megegyezik a host network namespace-ével. Az LXC esetében az LXC konfigurációs állományában adtam meg a `lxc.network.type = phys` beállítást, mely megadja, hogy milyen valós hálózati eszközt használjon a konténer. Ezen beállítások után a méréseim a következőképpen alakultak.

	Docker	***LXC***	***Singularity***	***KVM***	***Native***
Átlag (MB)	1122	1121,8	1122,1	1116,7	1122,2
Szórás (MB)	0,471	0,632	0,316	15,72	0,632

4. táblázat. Mérési eredmények hoszt network módban



34. ábra. Mérési eredmények hoszt network módban

Jól látható, hogy ebben a módban az eredmények kisimultak, és minden technológia hasonló értéket hoz. Az összehasonlításból az is látszik, hogy a szórás is minimális értékre csökkent, így az eredmények stabilnak módhatóak. Emellett a mérések során folyamatosan figyeltem az újraküldött csomagok számát, mely drasztikusan lecsökkent az előző mérésekhez képest.

	Docker	***LXC***	***Singularity***	***KVM***	***Native***
Átlag	1,5	3,7	2,71	150,6	0

5. táblázat. Újraküldési ráta hoszt network módban

Ebben a módban a leggyengébb teljesítményt a virtualizáció hozza, mely még mindig Linux Bridge-et használ a hálózati elérés módjának, ugyanakkor sokkal jobban teljesít, mint amikor konténereket használtam.

A mérésekből látszik, hogy a Singularity konténer technológia, mely a HPC rendszerekhez lett fejlesztve, alapértelmezetten a hoszt networking opciót használja, így teljesítménye alapértelmezett beállítások mellett is képes hozni a natív futtatásokkal

közel azonos eredményeket. Publikációm írásakor még nem álltak rendelkezésre teljesítménytesztek erről a megoldásról, de láthatóan a rugalmasságát valamelyest csorbítva maximalizálták a teljesítménymutatók minden aspektusát.

4.5 2. Téziscsoport

1. Összeállítottam egy tesztrendszert melynek segítségével képes voltam az egyes konténer technológiák teljesítménymutatóit összehasonlítani.
   a. Több tesztet is elvégeztem a hálózati teljesítményt illetően. Ezek alapján megállapítottam, hogy a konténer technológiák alapvető hálózati beállítása ugyan rugalmasságot eredményez, viszont teljesítményben alulmarad a speciális hoszt mód beállítástól, mely során a konténer hálózati forgalmát közvetlen a hoszt hálózati forgalmába injektáljuk be ez által kihagyva az alapértelemeztt Linux Bridge-et a forgalomból.
2. A Singularity konténer megoldás alapbeállításban is minden teljesítménymutatóban közel a natívan futtatott referenciatesztek szintjét hozza, így különösen ajánlott rugalmasságot megkövetelő rendszerekben alkalmazni.

Téziseimet alátámasztó publikáció:

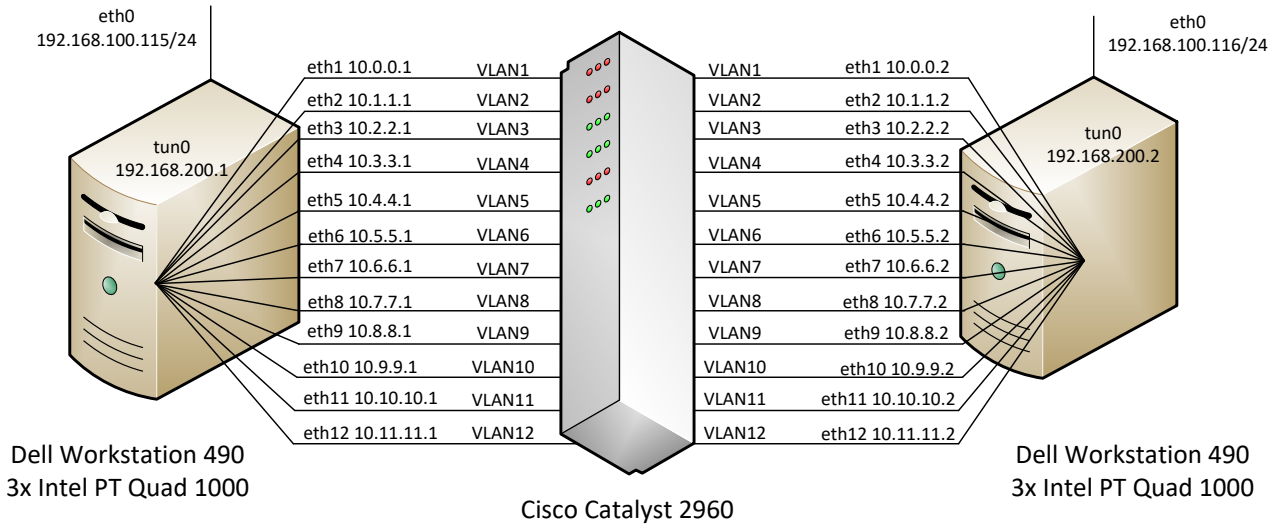
Á. Kovács, Comparison of different Linux containers, *In Proc of 2017 40th International Conference on Telecommunications and Signal Processing (TSP)*, 2017, pp: 47-51

5 MultiPath használata konténer technológiákkal

Ebben a fejezetben az eddig tárgyalt technológiák ötvözésének lehetőségeit mutatom be. A Multipath technológiák két különböző szinten vezethetők be egy rendszerbe: user szinten és kernel szinten. Mivel a konténer technológiák lényege, hogy a hoszt kernelen osztoznak, így nemcsak a user szintű (MPT) multipath technológa, hanem a kernel szintű (MPTCP) mutlipath technológia tesztelésére is lehetőség nyílik.

5.1 mérési összeállítás és telepítés

Az eddigi méréseimet alapul véve hasonló összeállításban teszteltem az egyes technológiák működését, valamint teljesítményét.



35. ábra. Konténer és MultiPath technológiák fizikai mérési elrendezése

A mérési elrendezés megegyezik a korábban mért rendszerekkel, de ebben az esetben a végpontok nem natívan futottak a gépeken, hanem konténerizálva. Ehhez először a már ismert módon konténereket hoztam létre az iperf mérőrendszerrel telepítve az alábbi definíciós fájlokat alkalmazva.

Dockerfile:

```
FROM debian:9

WORKDIR /root/

RUN apt update

RUN apt install -y libssl-dev iperf net-tools
```

Singularity Definíciós fájl:

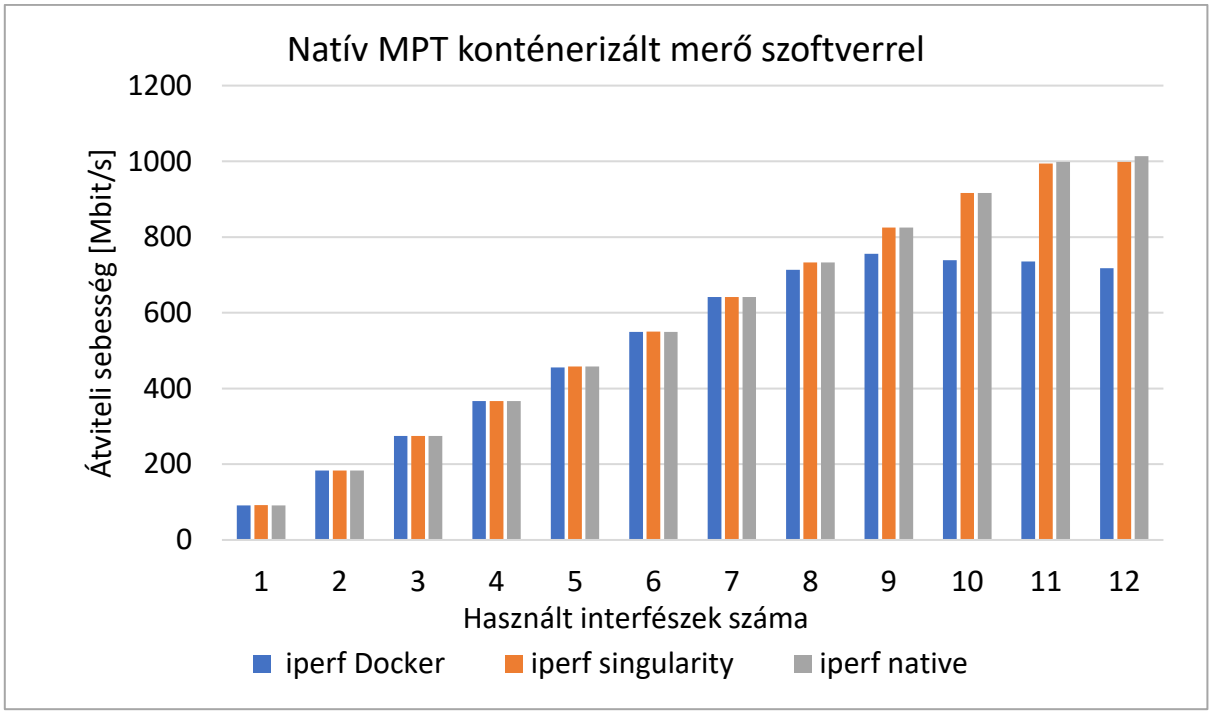
```
Bootstrap: docker

From: debian:9

%post

apt update

apt install -y libssl-dev iperf net-tools
```

5.2 Mérések bemutatása

Az első mérésnél a megvizsgáltam, hogy a hosztra natívan telepített MPT Multipath technológiát képes-e a konténerizált mérőszoftver kihasználni. Tehát a mérési eljárás hasonló, mint a 2.3.2 fejezetben, de annyi különbséggel, hogy itt az iperf mérőszoftver már konténerizálva volt és összehasonítottam a natív mérőszoftver eredményeivel is. Ahhoz, hogy kiküszöböljen a 4.3.1 fejezetben taglalt szűk keresztmetszetet, melyet a Linux Bridge okoz, itt is `--host` opcióval indítottam a Docker konténert.

5.3 Mérési eredmények

5.3.1 Tesztmérések



36. ábra. Natív MPT konténerizált mérő szoftverrel

A mérésekből jól látható, hogy a Singularity konténerben futtatott iperf mérőszoftver a natív futtatáshoz képest elenyésző különbséget mutat, ugyanakkor a Docker konténer megoldásnál a 9. hálózati interfész után csökkenő eredményeket produkál, és nem képes a folyamatos dinamikus átviteli kapacitás aggregálására, mint a Singularity-ben vagy a natívan futtatott módokban.

5.3.2 Konténerizált MPT

Ezek után a MultiPath technológia konténerizálása következett. Először az MPT szoftvert konténerizáltam. Ehhez szükség volt fordító környezet telepítésére a konténerbe, majd ezek után az MPT forráskódját bemásolva a konténerben fordítottam le azt. Ahhoz, hogy a konfigurációs állományok könnyen módosíthatók legyenek, a hoszt fájlrendszer egy általam kijelölt mappáját `-v` opcióval (Docker Volume) vagy

pedig `-B` (Bind, Singularity) ezt a könyvárat felcsatoltam a konténerben futtatott MPT megfelelő mappájába, így a mérések során ezt módosíthattam. Ezt mindkét konténer technológiával megtettem.

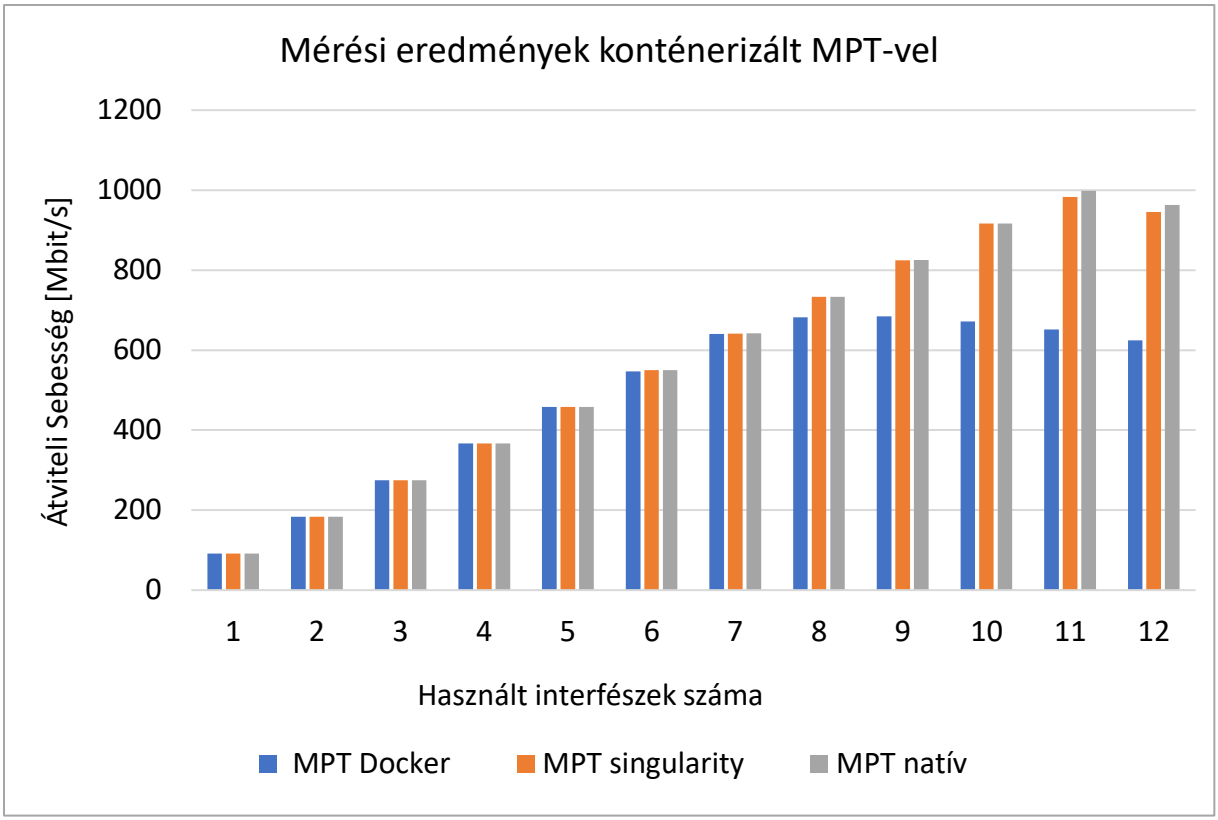
Működéséből fakadóan az MPT egy új virtuális interfészt hoz létre indulása során, ezért a Docker esetében speciális jogosultságok szükségesek. Alapértelmezetten a Docker konténerek nem privilegizált üzemmódban futnak, és nem férnek hozzá a különböző eszközökhöz. Ahhoz, hogy ezt megadjuk, esetünkben szükség volt a `--cap-add=NET_ADMIN` opcióra, mellyel hálózati adminisztrátori hozzáférést adunk valamint a `--device /dev/net/tun:/dev/net/tun` opcióra, mellyel megadhatjuk azt az eszközt, amelyre ez érvényes. Így a konténerben futtatott MPT már képes volt új `tun` eszközt létrehozni indulásakor. A Singularity esetében nem szükséges ilyen beállítás megtétele, hiszen itt képesek vagyunk `root` felhasználóként is teljesen privilegizált üzemmódban indítani a konténert. Méréshez kiadott `iperf` parancsok:

szerver:

```
iperf3 -s
```

kliens

```
iperf3 -c 10.100.0.2 -t 120 -i 2 -y C
```



37. ábra. Mérési eredmények konténerizált MPT-vel

Jól látható, hogy a konténerizált MPT MultiPath megoldás szinte teljes mértékben megegyezik a natívan implementált MPT mérési eredményeivel. Ugyanakkor az is megfigyelhető, hogy a Dockerben futtatott MPT már a 9. hálózati interfész hozzáadásánál elkezd csökkenő tendenciát mutatni, míg natív futtatásnál ez jelenség csak a 12. hálózati interfész hozzáadásánál jelentkezett. Ugyanakkor a Singularity konténerben használt megoldás teljesen megegyező mérési eredményeket produkált.

5.3.3 Konténerizált MPTCP

Az MPTCP egy kernel szintű MultiPath technológia. A konténerizációs technológiákból adódik, hogy hoszt Operációs rendszer kernelén „osztoznak”, így az MPTCP telepítéséhez elég volt a hoszt kernelét kicserélni, hogy az összes hoszton futó konténerizációs megoldás a képes legyen a MultiPath technológia használatára. Azért, hogy tesztelhessem ezen technológiát is, az előző mérésekben használt iperf Singularity konténert használtam méréseimhez, a hoszton telepített MPTCP kernel-lel egyetemben.

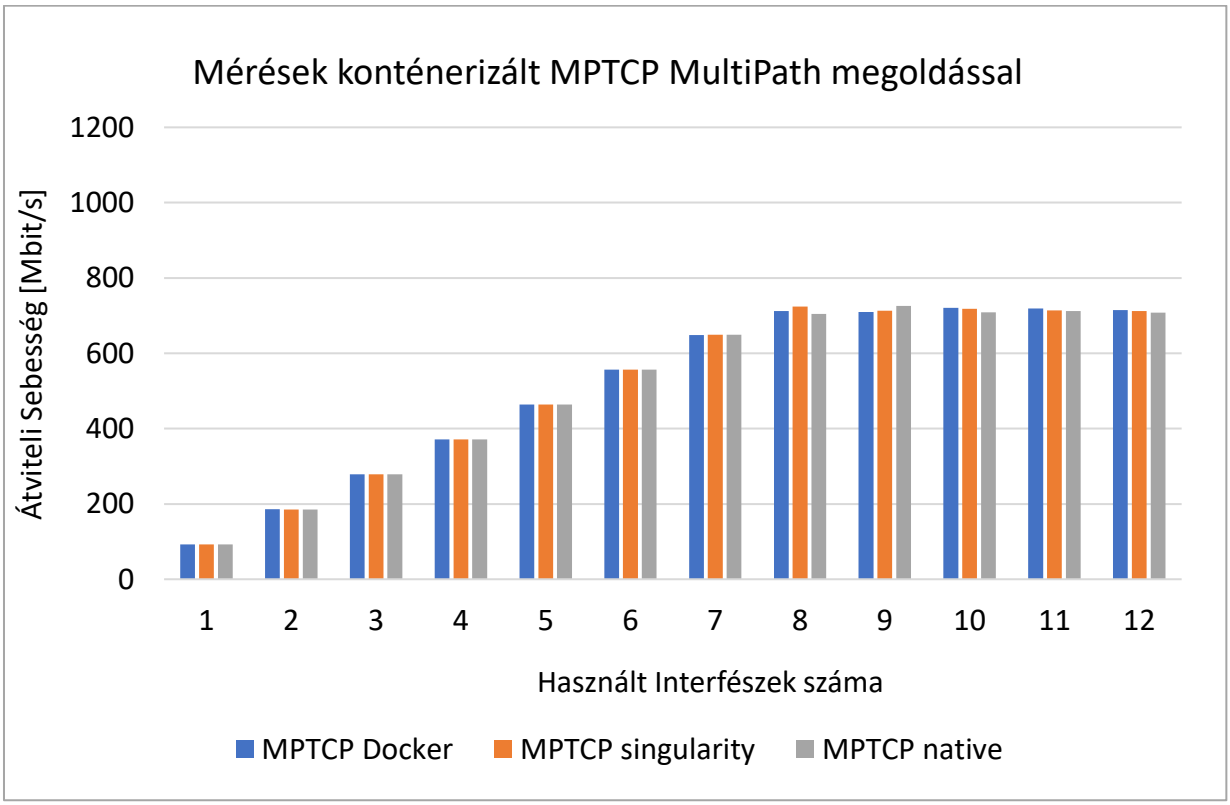
Megvizsgáltam, hogy van-e különbség, ha kliensként, szerverként esetleg mindkét futtatott iperf komponenst Singularity konténerben futtatom, de a méréseim azt mutatták nincs ilyen fajta különbség az egyes szcenáriók között. Mérésekhez használt parancsok:

szerver:

```
iperf3 -s
```

kliens

```
singularity exec /root/iperf.simg iperf -c 10.0.0.2 -t 120 -i 2 -y C
```



38. ábra. Mérések konténerizált MPTCP MultiPath megoldással

Jól látható, hogy a mérési eredmények teljesen megegyeznek a 2.3.3 fejezetben mért értékellek, vagyis ebben az esetben is találkozhatunk a MPTCP maximálisan használható interfészek számának korlátjával [25], viszont addig a pontig míg a 9. interfészt nem adjuk hozzá az átvitelhez, a natív teljesítményt nyújtja.

5.4 3. Téziscsoport

1. Az egyes MultiPath technológiák implementálása konténerizált környezetben. Mind az MPT-t mind az MPTCP-t lehetséges konténer technológiákkal ötvözni, az MPT-t képesek vagyunk konténerbe telepíteni, amennyiben megfelelő jogosultságokkal látjuk el az adott konténert, míg az MPTCP-t a konténerizációból adódóan csak a hoszt kernelének cseréjével implementálhatjuk, cserébe viszont bármely már elkészült konténerizált alkalmazás bizonyos esetekben módosítás nélkül használhatja azt.
   a. A teljesítmény tesztekkel megmutattam, hogy míg a natívan futtatott MPT egészen 12 hálózati interfészig képes a hálózati interfészek sebességének aggregálására, addig Dockerben futtatva ez a csak 8 interfészig érvényes, ott folyamatosan csökkeni kezd ez az aggregációs képesség. Ebben az esetben a MPT legnagyobb előnyét veszti le az MPTCP-vel szemben.
   b. Az MPT-Singularity kettős megoldásával gyorsan létrehozható egy rugalmas MultiPath végpont bármely felhő alapú rendszeren. Ezekkel a tulajdonságaival ez a kettős ideális HPC rendszerekbe, vagy intenzív hálózati forgalmat bonyolító rendszerekbe.

Téziseimet alátámasztó publikáció:

Á. Kovács, G. Lencse, Evaluation of layer 3 multipath solutions using container technologies, *In Proc of 2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, 2017, pp: 207-211

6 Összefoglalás, jövőbeni kutatási tervek

Kutatásaim során mind a Multi-Path technológiákban, mind a konténerizált megoldásokban elmélyedtem. Egyes ilyen megoldások a jövő meghatározó és széles körben elterjedt kommunikációs rendszerek alapjai lehetnek. Feltett szándékom, hogy tovább folytatom kutatási tevékenységemet a témában, és tervezem az egyes megoldásokat életből vett valós helyzeteket szimulálva tesztelni, kiemelni erősségeiket, gyengeségeiket. Valamint ezek publikálásával segíteni a fejlesztőket a még jobb megoldások kialakításában.

Publikációim, előadásaim során mindig sok kérdést, érdeklődést kaptam a lehető legjobb technológia kiválasztása témakörében. Sok esetben ezen kérdések megválaszolása a legnehezebb, hiszen ezek a technológiák napról napra fejlődnek, újdonságokkal rukkolnak elő. Független hivatkozásaim száma jelzi, hogy eredményeimet számos más kutató használta fel, így hozzájárultak ezen még korántsem kiforrott kutatási terület fejlődéséhez. Rengeteg potenciális kutatási feladat és kérdés vár még megválaszolásra amellett, hogy szinte minden nap jön valami újdonság, melyet érdemes lehet megvizsgálni. A következő bekezdések ezek közül ragadnak ki néhányat.

6.1 Mérési módszertant érintő kutatások

A méréseim során általában az iparban elterjedten használt iperf mérőprogramot használtam, illetve az eredményeimet esetenként HTTP letöltéssel is ellenőriztem. Létezik egy mérési eljárás, amit az IETF (Internet Engineering Task Force) elsősorban hálózati eszközök (pl. switchek, routerek) teljesítőképességének vizsgálatára dolgozott ki. Az eljárás eredeti verzióját az RFC 2544 írja le. Bár ez elvileg független az IP protokoll verziójától, szemléletében és példáiban az IP protokoll 4-es verziója jelenik meg. A mérési eljárás újabb változatát az RFC 5180 írja le, ami az IPv6-ra koncentrál, és az ún. IPv6 áttérési technológiákat kizárja a tárgyköréből. Ez utóbbiakra az RFC 8219 definiált mérési módszertant. Mindhárom RFC közös jellemzője, hogy a throughput méréseknél az ún. zero loss kritériumot használja. Definíció szerint a throughput az a

legmagasabb konstans frame rate, ami mellett a vizsgált eszköz (DUT: Device Under Test) veszteségmentesen képes a test frame-ek továbbítására a vizsgálat teljes időtartama alatt (legalább 60s). Hálózati eszközök esetén ez a definíció nagyon hasznos, mert szigorú garanciát ad az eszközök átviteli jellemzőire. Ugyanakkor bizonyos esetekben, különösen szoftveres megoldások esetén, ez a kritérium túl szigorú lehet, ezért az RFC 2544 / RFC 5180 szerint működő, kereskedelmi forgalomban kapható mérőeszközök általában lehetőséget adnak 0-tól különböző "Loss Tolarance" érték beállítására. Témavezetőmmel meg is mutattuk, hogy a gyakorlat szempontjából a nem nulla veszteség sok esetben nem okoz problémát. [50]

Hipotézisem szerint az általam vizsgált virtualizációs megoldások esetén jelentősen elérhet a mért throughput abban az esetben, ha a megengedett veszteség mértéke nulla, illetve ha nullától különböző, de a felhasználók szempontjából érdemi minőségromlást nem okozó érték (pl. 0,01%, azaz 10000 csomagból 1 veszhet el). Érdekes kihívást jelent annak vizsgálata, hogy ez az eltérés milyen mértékű az egyes virtualizációs megoldások esetén.

6.2 Eltérő sebességű linkek aggregációja

Az MPT és MPTCP megoldások aggregációs képességeinek vizsgálatánál mindig azonos sebességű utak átviteli kapacitását összegeztem. Ez a mérések szempontjából kézenfekvő megoldás volt: több azonos link használata nem jelentett nehézséget. Azonban a gyakorlatban előfordulnak olyan helyzetek, amikor az átviteli utak sebessége jelentősen eltérő. (Például egy eszköz rendelkezik Ethernet, WiFi, 4G/5G interfésszel.) Ilyen mérések kivitelezése számos nehézséggel jár, például WiFi esetén az egyéb eszközök forgalma jelentősen befolyásolhatja a mérést, a 4G/5G eszközök esetén a mérésekhez szükséges nagy adatforgalom jelentős költségekkel járhat. Ha csak azt szeretném megvizsgálni, hogy az eltérő sebességeknek milyen hatása van az aggregáció hatékonyságára, akkor viszont ezeket a problémákat ki tudom küszöbölni több azonos vezetékes interfész használatával úgy, hogy ezek sebességét mesterségesen korlátozom (például NetEm használatával, amint [50] cikkünkben is

tettük). Ezek mellett akár megvizsgálható az egyes különböző sebességű linkek nemcsak sebességparamétereinek változtatása, hanem akár csomagvesztéssel illetve késleltetéssel való terhelése is. Egy ilyen kísérletsorozat a jelenlegi Multi-Path technológia egyik jellegzetessége lehet, mely során a hibával terhelt linket kevésbé terheljük, mint a hibákat generáló linket.

6.3 Aggregációs vizsgálatok nagyobb CPU magszám mellett

Amikor az MPT és MPTCP megoldások aggregációs képességeit vizsgáltam, csak 4 CPU magot tartalmazó eszközökbe tudtam három darab 4-portos PCI Express hálózati kártyát beletenni annak érdekében, hogy 12 azonos sebességű linkig el tudjam végezni a méréseket. Az eredmények alapján az a hipotézisem, hogy lényegesen több CPU mag használata az MPTCP számára kedvezőbb feltételeket teremtene. Ennek vizsgálata is egy érdekes lehetséges kutatási feladat.

7 Köszönetnyilvánítás

Köszönetemet szeretném kifejezni témavezetőmnek, Dr. Lencse Gábornak, a dolgozat elkészítéséhez nyújtott segítségért. Külön köszönöm, hogy alapszakos hallgatóként felfigyelt rám és azóta is segíti a tudományos pályafutásomat.

Köszönöm a Távközlési Tanszék munkatársainak a segítséget, mellyel támogatták tudományos munkámat, kiváltképp Dr. Borbély Gábornak és Budai Tamásnak.

Köszönöm szüleimnek, hogy ösztönöztek a továbbtanulásra, mindig támogattak és kiálltak mellettem. Az ő támogatásuk nélkül nem készült volna el a dolgozat.

Hálával tartozom a feleségemnek, Majának, aki támogatott a disszertáció elkészítésében.

Ábrajegyzék

1. ábra. Felhő alapú infrastruktúrák eloszlása [1] 4
2. ábra. Vállalati felhő stratégák alakulása 2022-ben [2] 5
3. ábra Felhő szolgáltatók használata [2] 5
4. ábra. TCP és MPTCP felépítése 12
5. ábra. MPT architektúra (x, y ∈ {4, 6}) 14
6. ábra. Az MPT kommunikáció beágyazása 14
7. ábra. MPT működésének folyamata [15] 15
8. ábra. Multi-Path mérési összeállítás 18
9. ábra. MPT iperf mérési eredmények (100Mbit/s) 22
10. ábra. HTTP letöltés mérési eredmények (100Mbit/s) 22
11. ábra. MPT átviteli sebességek GRE-over-UDP előtt 23
12. ábra. Az MPTCP iperf tesztek 27
13. ábra. MPTCP HTTP letöltési tesztek 27
14. ábra. Az MPT és MPTCP átviteli aggregációs képességinek összehasonlítása 28
15. ábra. CPU foglaltság MPT processzek esetén 30
16. ábra. CPU foglaltság MPTCP esetén 31
17. ábra. MPT iperf tesztek 1 Gbit/s sebességen 32
18. ábra. MPTCP iperf tesztek 1 Gbit/s sebességen 32
19. ábra. MPT CPU kihasználtság 1 Gbit/s sebességen 33
20. ábra. MPTCP CPU kihasználtság 1Gbit/s sebességen 34
21. ábra. MPT tesztek 1Gbit/s sebességen 4 iperf szállal 35
22. ábra. MPTCP tesztek 1Gbit/s sebességen 4 iperf szállal 35
23. ábra. MPT CPU kihasználtság 1Gbit/s sebességen 4 iperf szállal 36
24. ábra. MPTCP CPU kihasználtság 1Gbit/s sebességen 4 iperf szállal 36
25. ábra. Hpyervisor kontra Konténer alapú virtualizáció 41
26. ábra. Docker Arhitektúra [33] 44
27. ábra. Adatrétegek a konténer image-ben. 45

28. ábra. Mérési összeállítás 50
29. ábra. CPU teszt mérési eredmények 53
30. ábra. Átviteli sebességek mérési eredménye 55
31. ábra. Újraküldött csomagok száma iperf mérés alatt 56
32. ábra. Alapértelmezett konténer hálózati blokkvázlat 57
33. ábra. Konténer hoszt mód 58
34. ábra. Mérési eredmények hoszt network módban 60
35. ábra. Konténer és MultiPath technológiák fizikai mérési elrendezése 63
36. ábra. Natív MPT konténerizált mérő szoftverrel 65
37. ábra. Mérési eredmények konténerizált MPT-vel 67
38. ábra. Mérések konténerizált MPTCP MultiPath megoldással 68

Irodalomjegyzék

[1] „IDG Cloud Computing Survey," 06 08 2020. [Online]. Available: https://www.idg.com/tools-for-marketers/2020-cloud-computing-study/. [Hozzáférés dátuma: 2022].

[2] Flexera, „Cloud Computing Trends: Flexera 2022 State of the Cloud Report," 2022. [Online]. Available: https://www.flexera.com/blog/cloud/cloud-computing-trends-2022-state-of-the-cloud-report/. [Hozzáférés dátuma: 2022].

[3] P. Mell, T. Grace, „The NIST Definition of Cloud," in *NIST Special Publication 800-145,* https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf, 2011.

[4] Internet Engineering Task Force, „IETF," RFC793, [Online]. Available: https://tools.ietf.org/html/rfc793. [Hozzáférés dátuma: 2021].

[5] G. Lencse, B. Almási, S. Szilagyi, „Investigating the Multipath Extension of the GRE in UDP Technology," *Computer Communications (Elsevier),* kötet 103, szám 12, pp. 29-38, 2017. https://doi.org/10.1016/j.comcom.2017.02.002

[6] K. Ákos, „Comparing the aggregation capability of the MPT communications library and multipath TCP," in *Proceedings of 7th IEEE Conference on Cognitive Infocommunications (COGINFOCOM),* IEEE Hungary Section, (2016) 496 p. pp. 157-161. , 6 p. , https://doi.org/10.1109/CogInfoCom.2016.7804542

[7] B. Almási, A. Harmann, „An Overview of the multipath communication technologies.,” *Proc. Advenced in Wireless Aensor Networks,* pp. 7-11, 2013. ISBN 978-963-318-356-4, Debrecen University Press,

[8] C. Raiciu S. Barre C. Pluntke A. Greenhalgh M. Wischik and D. and Handley, „Improving Datacenter Performance and Robustness with Multipath TCP,” *Proc. of SIGCOMM 2011.,* pp. 266-278, 2011. https://doi.org/10.1109/CloudNet.2016.53

[9] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, „On the Impact of Packet Spraying in Data Center Networks,” *Proc. of IEEE INFOCOM 2013,* pp. 78-91, 2013. https://doi.org/10.1109/INFCOM.2013.6567015

[10] L. Valiant, “A Scheme for Fast Parallel Communication,” SIAM journal, „A Scheme for Fast Parallel Communication,” *SIAM journal on computing,* kötet 11, szám 2, pp. 351-361, 1982. https://doi.org/10.1145/1462153.1462156

[11] M. Al-Fares A. Loukissas and A. Vahdat, „A Scalable Commodity Data Center Network Architecture,” *Proc. of SIGCOMM 2008.,* pp. 170-187, 2008. https://doi.org/10.1145/1402958.1402967

[12] A. Greenberg J. R. Hamilton N. Jain S. Kandula C. Kim P. Lahiri et al. , „VL2: A Scalable and Flexible Data Center Network,” *Proc. of SIGCOMM 2011.,* pp. 67-77, 2011. https://doi.org/10.1145/1594977.1592576

[13] M. Tekaya, N. Tabbane, S. Tabbane, „Multipath routing mechanism with load balancing in ad hoc,” *Proc Int. Conf. Computer Engineering and Systems (ICCES),* pp. 67-72, 2010. https://doi.org/10.1109/ICCES.2010.5674892

[14] Internet Engineering Task Force, „TCP Extensions for Multipath Operation with Multiple Addresses,” IETF, 2020. [Online]. Available: https://tools.ietf.org/html/rfc8684. [Hozzáférés dátuma: 2021].

[15] S. Szilágyi, „MPT Multipath Comminuccations Library,” 2021. [Online]. Available: https://irh.inf.unideb.hu/~szilagyi/index.php/kutatas/. [Hozzáférés dátuma: 2022].

[16] L. Yong, E. Crabbe, X. Xu és T. Herbert, „GRE-in-UDP Encapsulation,” IETF, https://tools.ietf.org/html/rfc8086, 2017.

[17] G. Lencse, S. Szilágyi, F. Fejes és M. Georgescu, „MPT Network Layer Multipath Library,” IETF, https://tools.ietf.org/html/draft-lencse-tsvwg-mpt-06, 2020.

[18] G. Lencse, Sz. Szilagyi, F. Fejes, M. Georgescu, „MPT Network Layer Multipath Library,” IETF, 2021.

[19] B. Almási, Sz. Szilágyi, „Multipath ftp and stream transmission analysis using the MPT software environment.,” *International Journal of Advanced Research in Computer and Communication Engineering.,* kötet 2, szám 11, pp. 4267-4272, 2013.

[20] F. Fejes, S. Rácz and G. Szabó, „Application agnostic QoE triggered multipath switching for Android devices,” *2017 IEEE International Conference on Communications (ICC),* pp. 1-7, 2017. https://doi.org/10.1109/ICC.2017.7997450

[21] S. Szilágyi and I. Bordán, „Investigating the Throughput Performance of the MPT-GRE Network Layer Multipath Library in Emulated WAN Environment,” *Acta Tech. Jaurinensis,* . kötet 15, . szám 1, pp. 7-14, 2022. https://doi.org/10.14513/actatechjaur.00639

[22] Z. Gal, B. Almási, T. Daboczi, R. Vida, S. Oniga, S. Baran, and I. Farkas, „Internet of things: application areas and research results of the FIRST project,” *Infocommunications Journal,* kötet 6, szám 3, pp. 37-44, 2014.

[23] K. Ákos, „Evaluation of the Aggregation Capability of the MPT Network Layer Multipath Communication Library and Multipath TCP,” *ACTA*

POLYTECHNICA HUNGARICA, kötet 6, pp. 129-147, 2019. https://doi.org/10.12700/APH.16.6.2019.6.9

[24] G. Lencse és Á. Kovács, „Advanced measurements of the aggregation capability of the MPT multipath communication library," *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems,* kötet 4, szám 2, pp. 41-48, 2015. https://doi.org/10.11601/ijates.v4i2.112

[25] „MPTCP Developer mailing list," [Online]. Available: https://listes-2.sipr.ucl.ac.be/sympa/arc/mptcp-dev/2016-02/msg00018.html. [Hozzáférés dátuma: 2021].

[26] S. Szilágyi, I. Bordán, L. Harangi, B. Kiss , „Throughput Performance Comparison of MPT-GRE and MPTCP in the Gigabit Ethernet IPv4/IPv6 Environment," *Journal of Electrical and Electronics Engineering,* kötet 12, szám 1, pp. 57-60, 2019. https://doi.org/10.14513/actatechjaur.00639

[27] Intel, „3rd Generation Intel® Xeon® Scalable Processors," Intel, https://www.intel.com/content/www/us/en/products/details/processors/xeon/scalable/platinum.html, 2022.

[28] S. Dillenburg, „medium.com," 5 2020. [Online]. Available: https://medium.com/an-idea/a-brief-history-of-container-virtualization-57fc96c02924. [Hozzáférés dátuma: 2022].

[29] R. Osnat, „https://blog.aquasec.com," 2021. [Online]. Available: https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016. [Hozzáférés dátuma: 2022].

[30] C. Burnikse, „The next generation of virtualization?," Ark Invest, 2014. [Online]. Available: https://ark-invest.com/articles/analyst-research/containers-virtualization/.

[31] T. Bui, „Analysis of docker security," *arXiv preprint,* 2015.

[32] „docker.com," [Online]. Available: https://docs.docker.com/get-started/. [Hozzáférés dátuma: 2022].

[33] „Docker Manual," Docker, [Online]. Available: www.docker.io. [Hozzáférés dátuma: 2022].

[34] Merkel, D., „Docker: lightweight linux containers for consistent development and deployment," *Linux Journal,* kötet 239, 2014.

[35] Bukhary Ikhwan Ismail, Ehsan Mostajeran Goortani, Mohd Bazli Ab Karim, Wong Ming Tat, Sharipah Setapa, Jing, „Evaluation of Docker as Edge Computing Platform," *Conference on Open Systems (ICOS),* pp. 130-135, 2015. https://doi.org/10.1109/ICOS.2015.7377291

[36] Baker, M., „Over half of psychology studies fail reproducibility test," *Nature News,* kötet 27, 2015. https://doi.org/10.1038/nature.2015.18248 https://doi.org/ 10.1186/s13742-015-0087-0

[37] Fiona F., Ascelin G., „Science is in a reproducibility crisis: How do we resolve it?," phys.org, 2013. https://phys.org/news/2013-09-science-crisis.html, [Hozzáférés dátuma: 2022].

[38] Belmann, P., Dröge, J., Bremges, A., McHardy, A. C., Sczyrba, A., & Barton, M. D., „Bioboxes: standardised containers for interchangeable bioinformatics software," *Gigascience,* kötet 4, szám 1, 2015. 2015. https://doi.org/ 10.1186/s13742-015-0087-0

[39] Gorgolewski KJ, Alfaro-Almagro F, Auer T, Bellec P, Capotă M, Chakravarty MM, „BIDS apps: Improving ease of use, accessibility, and reproducibility of neuroimaging data analysis methods.," *PLoS Comput Biol,* kötet 13, szám 3, 2017.

[40] Hosny, A., Vera-Licona, P., Laubenbacher, R., & Favre, T. , „AlgoRun: a Docker-based packaging system for platform-agnostic implemented

algorithms.," *Bioinformatics,* kötet 32, szám 15, pp. 2396-2398, 2016. https://doi.org/10.1093/bioinformatics/btw120

[41] Priedhorsky, R., & Randles, T., „Charliecloud: Unprivileged containers for user-defined software stacks in hpc," *Proceedings of the international conference for high performance computing,* pp. 1-10, 2017. https://doi.org/10.1145/3126908.3126925

[42] „Shifter: User Defined Images," NERSC, [Online]. Available: https://www.nersc.gov/research-and-development/user-defined-images/. [Hozzáférés dátuma: 2022].

[43] Kurtzer GM, Sochat V, Bauer MW, „Plos ONE - Singularity: Scientific containers for mobility of compute," 2017. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0177459. [Hozzáférés dátuma: 2021].

[44] Rizki Rizki; Andrian Rakhmatsyah; M. Arief Nugroho , „Performance analysis of container-based hadoop cluster: OpenVZ and LXC," *4th International Conference on Information and Communication Technology,* pp. 1-4, 2016 . https://doi.org/10.1109/ICoICT.2016.7571957

[45] Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio,, „An Updated Performance Comparison of Virtual Machines and Linux Containers," *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS),* pp. 171-172, 2015. https://doi.org/10.1109/ISPASS.2015.7095802

[46] Miguel G. Xavier; Marcelo V. Neves; Fabio D. Rossi; Tiago C. Ferreto; Timoteo Lange; Cesar A. F. De Rose, „Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments," *21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing* , pp. 233-240, 2013. https://doi.org/10.1109/PDP.2013.41

[47] „linuxcontainers.org,” [Online]. Available: https://linuxcontainers.org/lxc/getting-started/. [Hozzáférés dátuma: 2022].

[48] V. Thakur, „www.metricfire.com,” metricfire, 02 2020. [Online]. Available: https://www.metricfire.com/blog/understanding-dockers-net-host-option/. [Hozzáférés dátuma: 2022].

[49] K. Suo, Y. Zhao, W. Chen and J. Rao, „An Analysis and Empirical Study of Container Networks,” *IEEE INFOCOM 2018 - Conference on Computer Communications,* pp. 189-197, 2018. https://doi.org/10.1109/INFOCOM.2018.8485865

[50] G. Lencse, Á. Kovács, K. Shima, „Gaming with the Throughput and the Latency Benchmarking Measurement Procedures of RFC 2544,” *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems,* kötet 9, szám 2, pp. 10-17, 2020. http://dx.doi.org/10.11601/ijates.v9i2.288