



**SZÉCHENYI
EGYETEM**
UNIVERSITY OF GYŐR



Széchenyi István Egyetem, Távközlési Tanszék

IP csomagtovábbító rendszerek teljesítményének összehasonlító elemzése

Készítette: Kosák Melinda Magdolna

Konzulens: Dr. Lencse Gábor

2024

Tartalomjegyzék

Tartalomjegyzék.....	1
1. Bevezetés.....	2
2. A mérési módszer elméleti háttere	3
2.1. RFC 2544.....	3
2.2. RFC 4814.....	6
2.3. RFC 5180.....	6
2.4. DPDK	7
2.5. A Linux kernel csomagtovábbítása	9
2.6. FD.io VPP.....	12
2.7. Siitperf.....	13
3. Az elvégzett munka.....	15
3.1. Feladat ismertető.....	15
3.2. Mérési összeállítás.....	15
3.2.1. Fizikai szinten	15
3.2.2. IP szinten.....	16
3.2.3. Siitperf.....	18
3.3. FD.io telepítése különböző szervereken.....	19
3.3.1. FD.io beállítása	20
3.4. Mérési eredmények.....	22
3.4.1 A Linux kernel csomagtovábbítási teljesítménye	22
3.4.2 Az FD.io VPP csomagtovábbítási teljesítménye	25
3.4.3. Összehasonlítás	27
4. Összefoglalás.....	28
Irodalomjegyzék.....	29

1. Bevezetés

Az Internet életünk nélkülözhetetlen részévé vált. Működésében alapvető fontosságú az Internet protokoll (IP), amely az adatcsomagok továbbítását végzi. A TCP/IP protokollcsalád nagyon sokrétű és sok mindenre kiterjed. Dolgozatom szerves részét képezik, ugyanis ezek szabályozzák az alapvető működéstől egészen a bonyolult összeállításokig. A technológia fejlődésével egyre gyorsabb információáramlásra van szükségünk. Rohanó világunk az információ sebesség állandó növekedéséhez szokott az elmúlt években. A sebesség mellett természetesen a biztonság is nagyon fontos.

Az IP-nek jelenleg két verziója van használatban: a régi 4-es verzió (IPv4) és a jelenleg terjedőben levő új, 6-os verzió (IPv6). Az IPv4 és IPv6 csomagok továbbítására számos szabad szoftver megoldás létezik: képes rá a Linux kernel, egy nagy teljesítményű megoldás az FD.io VPP. Dolgozatomban az FD.io VPP szoftver vizsgálata kapta a legnagyobb figyelmet, mert azt állítják, hogy ők a „Világon a leggyorsabb és legbiztonságosabb hálózati csomagtovábbítók.”, ezt az állítást fogom megvizsgálni.

A legjobb tudásom szerint végeztem a vizsgálatot összehasonlítva a két szoftvert. Nem csupán elméleti ismeretek alapján végeztem, hanem nagy figyelmet szenteltem a fizikai mérésekre is. Méréseimnél figyelembe vettem a hardveres specifikációkat, így még részletesebb összehasonlítást tudtam végezni. A dolgozatom tetőpontja az előbb említett összehasonlítás, melyben konzulensem tapasztalata nagy segítséget nyújtott.

2. A mérési módszer elméleti háttere

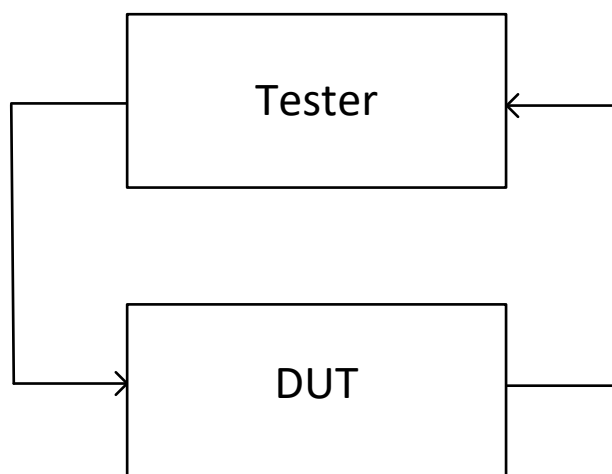
Ebben a fejezetben bemutatom azokat az RFC¹-ket, amiket a munkám során nagy szerepet játszottak. Továbbá részletezem azokat az IP csomagtovábbító szoftvereket, amiket használtam: Linux kernel és FD.io VPP. A Siitperf mérőprogramról is fog szólni az egyik alfejezet.

2.1. RFC 2544

Az RFC 2544-nek [1] az a célja, hogy meghatározza, hogyan kell egy hálózatot összekötő eszköz teljesítményét megmérni objektívan. A legfontosabb szempontok, amiket figyelembe kell vennünk egy sikeres méréshez:

- mérési összeállítás
- DUT (device under test) beállításai
- Keret (frame format, frame sizes)
- a tesztelési idő (min. 60 perc)

Az RFC-ben 3 különböző mérési összeállítást mutattak be, számomra az első volt a megfelelő, ami így néz ki:



1. ábra A választott mérési összeállítás (forrás: RFC 2544)

Ez a fajta összeállítás az ideális és általában használt a méréseknél. Eszerint mindkét eszköz rendelkezik adó és vevő porttal is. A kapcsolat a két eszköz között úgy jön létre, hogy a Tester a küldő portján keresztül csomagokat küld a DUT fogadó portjára, majd a DUT a csomagokat továbbítja és megékezik a Testerhez. Mivel a tesztelő egyszerre küldi és fogadja a

¹ Az RFC (angolul Request Ror Comments) dokumentumok írják le a TCP/IP protokollcsalád működését, a legfontosabb hálózati protokollokat, és a teljesítményük vizsgálatára használt mérési eljárásokat is.

tesztforgalmat, így könnyen megtudja állapítani, hogy az összes továbbított csomagot sikeresen megkapta.

A tesztek mérése előtt a DUT gép konfigurálása a megadott lépések szerint egy elhagyhatatlan lépés. A szükséges protokolloknak engedélyezve kell lenniük, minimális változtatások engedélyezettek csak, amik fontosak a specifikus mérésekhez. Nem elfogadható például a keretkezelő pufferek (frame handling buffers) méretének megváltoztatása a keretkezelési sebességek tesztjei között, vagy az összes szállítási protokoll letiltása egy kivételével az adott protokoll átviteli teljesítményének tesztelése során. A konfiguráláshoz tartozik még az útvonal frissítések (routing update), keep alive frequency.

TCP/IP over Ethernet keretformátumok vannak használatban, amik lehetnek: learning frame, routing update frame, management query frame, test frame. A 4 keret típusból a teszt keretet emelném ki, ami egy UDP Echo Request, amely annyi bitet tartalmaz, hogy kitöltse a szükséges keretméretet. A keretformátumon kívül, amit fontosnak tartok még megemlíteni, az a keretméret. Ethernetre vonatkoztatva a keretméreteket az alábbiak lehetnek: 64, 128, 256, 512, 1024, 1280, 1518 Ethernet használata során.

Ahhoz, hogy jobban be tudjam mutatni, hol alkalmazzák a test frame-eket, először néhány összehasonlító eljárást (benchmarking test) típust szeretnék kiemelni a szabványból:

- Throughput (áteresztőképesség)
- Latency (késeltetés)
- Frame loss rate (keretvesztesség arány)
- Back-to-back frames
- System recovery (rendszer felépülés)
- Reset (visszaállítás eredeti helyzetbe)

A *throughput* az a másodpercenkénti maximum keretmennyiség, ahol átvitel közben nem történt keretvesztés. Ennek mérése egy olyan mérési eljárás, ahol Tester egy adott keretráta (frame rate) mellett egy adott ideig (legalább 60s) kereteket küld a DUT-nak, és ellenőrzi, hogy a DUT-tól a keretek visszaérkeznek-e hiány nélkül. Ha az elküldött keretszám nem egyezik a kapottal, akkor a következő méréshez csökkenti a keretrátát. Sikeres mérés esetében pedig a következő méréshez növeli azt. A gyakorlatban bináris keresést használnak a lehető legmagasabb olyan keretráta megállapítására, ahol még minden keret hiánytalanul megérkezik. Fontosnak tartom megjegyezni, hogy bár az RFC 2544-ből származó 1. ábrán a nyilak egyirányúak, az RFC azt írja elő, hogy a mérést kétirányú forgalommal kell elvégezni.

A *latency* mérés célja, hogy meghatározza azt az időt, ami eltelik, amíg egy csomag a forrástól (source) a meghatározott célállomásig (destination) megérkezik. A latency meghatározása előtt a throughput mérést kell elvégezni. A latency mérés, azután annak az eredményét veszi alapul, hogy a megfelelő keretráta (másodpercenkénti keretek száma) mellett küldje a kereteket a DUT-on keresztül. Ennek a mérésnek 120 másodpercig kell tartania. A teszt során egy keret kap egy egyedi azonosítót (identifying tag), amit 60 másodpercnél küld a Tester. Közvetlenül a megjelölt keret küldése után a Tester eltárol egy időbélyeget, ezt Timestamp A-nak nevezik. A második időbélyeget (Timestamp B) akkor tárolja el a Tester, amikor a megjelölt keret teljesen megérkezett. A késletetés meghatározásához pedig a Timestamp B-ből kivonjuk a Timestamp A-t. Ennek a mérésnek legalább 20-szor le kell mennie és minden egyes mérés után rögzítve lesz a B-A értéke, majd a latency ezeknek az értékeknek az átlagából kerül meghatározásra, így egy pontosabb eredményt kapunk.

A *frame loss rate* meghatározza, hogy egy adott keretráta mellett egy meghatározott idő alatt elküldött adott mennyiségű keretből hány veszik el. Ennek az aránynak a kiszámolásához az alábbi képletet használja:

$$((\text{küldött_csomagok_száma} - \text{vett_csomagok_szám}) * 100) / \text{küldött_csomagok_száma}$$

A teszt legelőször az átviteli közeg által lehetővé tett 100%-os sebességgel indul, majd az egymás követő tesztek között 10%-kal csökkenti a sebességet mindaddig, amíg 2 egymást követő teszten nem voltak elveszett csomagok, vagy amíg el nem éri a 10%-os sebességet, mert akkor a mérés sikertelennek nyilvánítja.

A *back-to-back frames* mérés lényege, hogy megvizsgálja, meddig tud a DUT a maximum keretrátával dolgozni. Nagy mennyiségű csomagot küld a DUT-hoz úgy, hogy a keretek között csak a minimális interframe gap időtartam telik el, és megszámlolja, hány elküldött keret érkezett meg. Ha mindegyik csomag megérkezik, akkor növeli a keretek mennyiségét (burst of frames), ezzel arányosan természetesen a mérés időtartamát, ha kevesebb, akkor meg csökkenti a számukat. A tesztnek legalább 50-szer le kell futnia és legalább 2 másodpercig kell egy mérésnek tartania.

A *system recovery* az az idő, ami kell ahhoz a rendszernek, hogy helyreálljon egy túlterhelés után. Ezt az időt a throughput alapján lehet meghatározni.

A *reset* az az idő, hogy milyen gyorsan tud a rendszer felállni egy hardware, software visszaállítás (reset) után vagy áramkimaradás esetén. Ezt a tesztek csak azoknál az eszközöknél kell elvégezni, amik közvetlen kapcsolatban állnak a DUT-tal.

Az RFC 2544 többféle eszköz, például az ISO OSI modell második rétegében működő bridge vagy switch valamint a harmadik rétegében működő, IP csomagtovábbító (router) teljesítményének a mérését is definiálja. Routerok teljesítményének a mérésére lefoglalt egy IP-cím tartományt is, ez a 198.18.0.0/15. A tartomány alsó felét (198.18.0.0/16) az 1. ábra szerinti összeállítás bal oldalán, a felső felét (198.19.0.0/16) a jobb oldalán kell használni. A DUT interfészeihez az adott tartomány 1-re végződő IP címét kell rendelni (198.18.0.1 és 198.19.0.1).

2.2. RFC 4814

Az RFC 4814 [2] több különböző témát is tárgyal. A méréseim szempontjából ezek közül a 4.5. pontjában említett az álvéletlen portszámok (pseudorandom port numbers) az érdekesek. A ma használt hálózati kártyák képesek arra, hogy a beérkező csomagokban található forrás és cél IP-címek, illetve TCP vagy UDP forrás- és célportszámok felhasználásával, egy hash függvény segítségével a csomagok által okozott megszakításokat szétosszák a különböző CPU magokhoz, hogy azok feldolgozzák őket. Ezt a folyamatot hívják vételoldali skálázásnak (RSS: Receive-Side Scaling), aminek a célja a skálázhatóság elérése. Az RSS támogatása érdekében az RFC 4814 ajánlja a pseudorandom portszámok használatát. Az ajánlott tartományok:

- forrás portszámoknál: 1024 – 65535
- cél portszámoknál: 1 – 49151

2.3. RFC 5180

Az RFC 5180 [3] sokban hasonlít az RFC 2544-hez, de míg az az IPv4 címekre fókuszált, addig az RFC 5180 már az IPv6-os címekre készült. Az RFC-ből kiemelném a keretméretet és a DUT protokoll címeket.

IPv6 esetén a csomagok fejrészének mérete 40 bájt, míg IPv4-nél 20 bájt. Az 1518 bájtos keretméret az untagged Ethernet frame maximális méretét jelenti, azonban az IEEE 802.3-as szabvány megengedi, hogy 2048 bájtosra növeljék, hogy megfeleljen az IEEE 802.1AE MAC

biztonsági protokoll által előírt frame tags és egyéb kapszulázások (encapsulation) elhelyezésének.

Az IANA (Internet központi adminisztrációjáért felelős szervezet) 2011-ben bejelentette, hogy az IPv4-es központi készlete kimerült. Szükséges volt egy megnövelt címtartományra (32 bit helyett 128 bit), amit a jövőben könnyebben lehet fejleszteni, így került bemutatásra az IPv6. Az IANA az IPv6-os címtartományból kijelölt egy részt, annak mérésére (benchmarking). Ezek a címek nem elérhetőek publikusan, és nem használhatók internetes forráscímekként vagy célállomás címekként. Az RFC 5180-ban a DUT IPv6-os címeket használ. A lefoglalt tartomány: 2001:2::/48. Az RFC 2544-hez hasonlóan, itt is meg kell felelni a címtartományt. Az 1. ábra bal felén használható tartomány: 2001:2::/49, a jobb felén használható pedig: 2001:2:0:8000::/49. A DUT protokoll alapján az interfészek címzésénél az alábbi feltételeknek kell a címeknek megfelelniük:

- Az előtaghossz /126, ami egy ponttól-pontig linket (point-to-point link) szimulál egy fő router számára.
- Az előtaghossz kisebb vagy egyenlő /64.

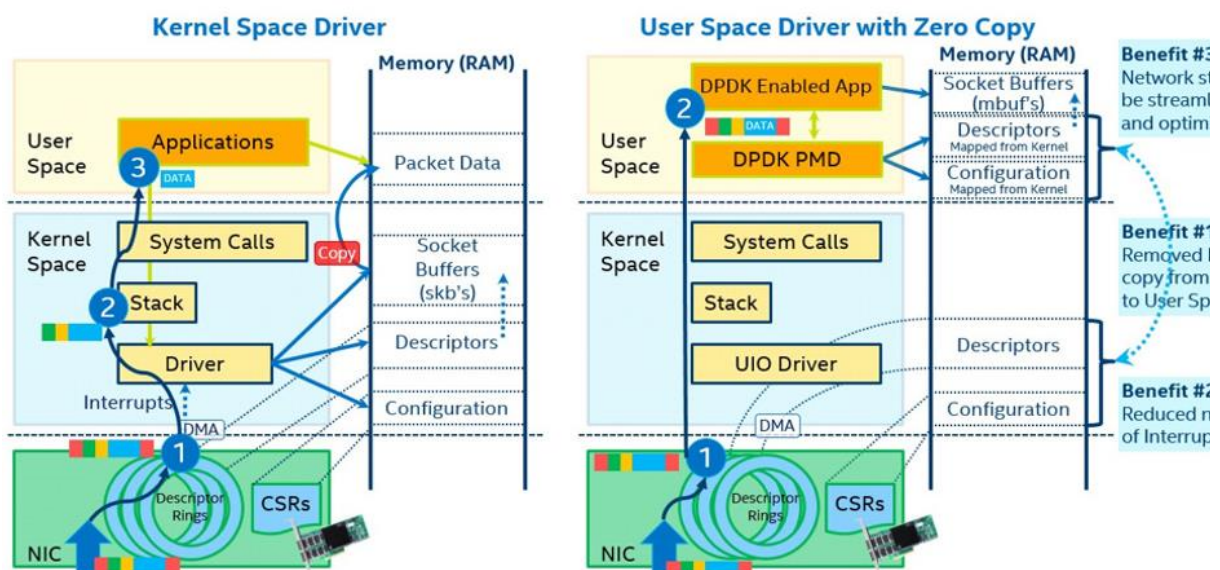
Az utolsó pont alapján nem ajánlott előtaghosszat választani a 64-128 tartományból, leggyakrabban a /64-es előtagot használják. Az IPv6 unicast interfész ID-jét a DUT-on ::1-re kell beállítani..

2.4. DPDK

A DPDK (Data Plane Development Kit) [4] egy nyílt forráskódú, Linux felhasználói felületen működő keretrendszer, amit azért hoztak létre, hogy javítsák a csomagfeldolgozási sebességeket különféle CPU-architektúrákon. Az Intel 2010-ben adta ki a szabadalmat, és 2013-ban jött létre az open source közösség DPDK.org webhelyen. [10]

Gyors népszerűsége tette szert, és ma már szinte az összes nagy CPU architektúrát, NIC-et támogatja és több platformon is elérhetővé vált. Hogy miért volt szükség a DPDK-ra? Egy olyan világban, ahol bárki számára már elérhető az Internet és ezekhez olyan alkalmazások és vezetékes/ vezeték nélküli eszközök készültek, mint videó streaming, okos készülékek, routerek, VoIP stb., amik számára nélkülözhetetlen a gyorsaság. Legfontosabb szempontokká váltak a teljesítmény, az áteresztőképesség (throughput) és az alacsony késleltetés, ezeket szeretné mindenki folyamatosan fejleszteni. A DPDK-nak egyik fő előnye ez is, hiszen lehetővé teszi a nagy sebességű hálózati adatcsomag alkalmazások (high-speed data packet networking applications) gyors fejlesztését.

Miért olyan gyors a DPDK? A DPDK úgy éri el a gyors csomagfeldolgozást, hogy olyan könyvtárakból és illesztőprogramokból (drivers) áll, amik képesek kikerülni az operációs rendszer network stack-jét, ezzel pedig csökkentik a késleltetést. Így a felhasználók lényegesen nagyobb teljesítményt kapnak a csomagfeldolgozási területen. A DPDK képes közvetlenül kezelni a hardware regisztereket, így nem a Linux kernelen át kell elérnie őket. Mivel közvetlenül képes kommunikálni a hardware-rel, így ez sebesség- és teljesítmény javulást eredményez. Vannak olyan programok, amik támogatják a DPDK-t ezzel pedig képesek még gyorsabban kezelni az érkező csomagokat. Ezeknél az alkalmazásoknál engedélyezve van, hogy a hálózati kártyák, közvetlenül DMA-zák a memóriát. A DMA (Direct Memory Access) segítségével az eszköz közvetlenül írhat vagy olvashat a memóriából, anélkül hogy a processzornak közbe kellene avatkoznia minden egyes adatátvitelnél. Ezért a DPDK-enabled alkalmazásoknál a hálózati kártya gyorsabban tudja feldolgozni a küldött vagy fogadott üzeneteket, mert a DMA lehetővé teszi, hogy elérje ezeknek az alkalmazásoknak a memóriáját, így kihagyva a CPU terhelését.



2. ábra Kernel és a DPDK csomagok útvonalának összehasonlítása [4]

A DPDK kulcsfontosságú elemei a következők [4]:

- **Environment Abstraction Layer (EAL)** — Egy olyan általános interfészt biztosít, ami elrejt a környezet specifikus jellemzőit az alkalmazások elől.

- **Ethernet Poll Mode Driver (PMD)** — Úgy tervezték, hogy tudjon működni olyan mechanizmusok nélkül, amik aszinkron és megszakítás alapú jelzések, helyettük a hálózati kártya folytonos lekérdezését végzi.
- **Memory management** — A többszintű virtuális memóriakezelés lassító hatását kiküszöbölő, ún. „huge page” memóriát használja, abban képes objektumokat kezelni (lefoglalni, felszabadítani) figyelembe véve a NUMA (Non-Uniform Memory Access) követelményeit is.
- **Buffer management** — Előre elhelyezett és fix méretű puffereket kezel, amiket a DPDK által kezelt memóriában tárolnak, ezzel lényegesen csökkentve azt az időt, amit az operációs rendszer azzal töltene, hogy elhelyezze és eltávolítsa őket.

A DPDK több könyvtárból áll [4]:

- *lib* - DPDK könyvtárak forráskódja
- *drivers* - DPDK poll módú illesztőprogramok forráskódja
- *app* - DPDK alkalmazások forráskódja (automatikus tesztek)
- *examples* - DPDK alkalmazáspéldák forráskódja
- *config, buildtools, mk* - Keretrendszerrel kapcsolatos makefile-ok, szkriptek és konfiguráció

2.5. A Linux kernel csomagtovábbítása

Ebben a fejezetben a Linux kernel csomagtovábbításáról fogok írni. [9] Először általánosságban ismertetem, hogy egy csomagtovábbítónak milyen feladatokat kell megoldani, azután rátérek arra, hogyan történik ez a Linux kernel esetén.

A csomagtovábbítást három részre oszthatjuk, és három kérdést is kell megválaszolnunk:

- Kinek küldjük tovább? (Útválasztás (routing))
- Hogyan küldjük tovább? (Tördelés (fragmentation))
- Milyen QoS biztosításával? (QoS (szolgáltatásminőség))

Az első kérdésre a választ, az adott csomagban levő célcím és az útválasztási tábla (routing table) alapján határozzuk meg. A küldés során tördelésre lehet szükség, amit a csomag és a következő hálózat MTU-ja (Maximum Transmission Unit) határoz meg, hogy mekkora méretben érkezik és továbbítható. Végül a harmadik kérdésre a válasz, hogy milyen minőségű lesz a csomagtovábbítás, azaz mennyi veszteség várható, az, hogy nem lehet előre megmondani „Best effort” nincs garancia. Esetleg lehet alkalmazni a ToS (Type of Service) mező

felhasználásával egyéb protokollokat és mechanizmusokat. Ezt azonban az Internet esetén általában figyelmen kívül hagyják.

A hálózat kiválasztásánál figyelembe veszi az IP cím Network ID és Host ID-ját, amit a hálózati maszk segítségével határoz meg. A cél IP-cím és az útválasztó tábla adott sorához tartozó hálózati maszk között bitenkénti ÉS műveletet végez. Ha ennek eredménye azonos az útválasztó táblázat adott sorában szereplő célhálózat címmel, akkor azt mondjuk, hogy illeszkedés van. Abban az esetben, ha a cél IP cím több hálózatra is illeszkedik, akkor legspecifikusabb illeszkedés alapján kell továbbítani. Ez azt jelenti, hogy ott, ahol a leghosszabb a hálózati maszk. Ezt az eljárást hívják Leghosszabb Prefixre illeszkedés algoritmusának (Longest Prefix Match Algorithm).

Mind a dedikált hardver routerek, mind a Linuxban implementált csomagtovábbító algoritmus ezt az elvet követi.

A Linux kernel a csomagtovábbítás során a beérkező hálózati csomagokat feldolgozza, és továbbítja a megfelelő helyre. Ennek során a kernel különböző mechanizmusokat használ, például a skaláris csomagfeldolgozást és az RSS-t (Receive Side Scaling). [8]

A skaláris csomagfeldolgozás során a kernel egyenként dolgozza fel a beérkező csomagokat. [5]

Egy megszakításkezelő függvény (alap esetben) egyetlen csomagot vesz át egy hálózati interfésztől, és egy sor függvényen keresztül dolgozza fel. A skalár csomag-feldolgozás egyszerű azonban nem a leghatékonyabb, mert:

- Amikor a kódút hossza meghaladja a mikroprocesszor utasítás-cache (I-cache) méretét, akkor a processzor folyamatosan új utasításokat tölt be a gyorsítótárba, ami csökkenti a teljesítményt.
- A másik probléma a mély hívási stack (call stack) használatából merül fel. Ahogy a függvények egymást hívják, a lokális változók kiesnek a processzor gyorsítótárjából, ami további teljesítményvesztést eredményez.

Ez a módszer egyszerű, de hatékonysága korlátozott lehet, mivel minden egyes csomag feldolgozásakor ugyanazt a hívási láncot kell végrehajtani, ami időigényes lehet, és megterheli a processzort és a gyorsítótárakat.

Receive Side Scaling (RSS) egy olyan mechanizmus a számítógép-hálózati kommunikációban, amely segít optimalizálni a hálózati csomagok kezelését és feldolgozását a számítógépes rendszerben. A RSS egyik feladata, hogy lehetővé tegye a hálózati interfészvezérlők (NIC) számára, hogy több vevőoldali várakozó sorral (multi-queue) rendelkezzenek, és az érkező

csomagokat különböző sorokba rendezzék, így a CPU magok között eloszlik a csomagfeldolgozásnak a terhe. A RSS technológiát használva a NIC az érkező csomagokat egy hash függvény segítségével osztja be a különböző sorokba, majd ezeket a sorokat különböző CPU-k feldolgozzák. Ez a hash függvény általában a hálózati és/vagy a szállítási réteg fejlécében helyezkedő adatmezők tartalmát használja. Linux esetén beállítástól függően vagy csak a forrás és cél IP-címeket vagy ezeken kívül forrás és cél TCP vagy UDP portszámokat is képes figyelembe venni. Az RSS segítségével a hálózati terhelés hatékonyabban oszlik meg a rendszerben, így növelve a teljesítményt és csökkentve a csomagok késleltetését. Amikor egy csomag érkezik, a NIC értesíti a megfelelő CPU-t az IRQ segítségével. Az IRQ-kat felhasználva az RSS lehetővé teszi a hálózati adatforgalom hatékony terheléelosztását a rendelkezésre álló CPU-k között. Minden fogadási sorhoz külön IRQ van társítva. Ennek eredményeként a fogadási megszakításokat több CPU is kezelheti, ami növeli a rendszer teljesítményét és skálázhatóságát. A PCIe eszközök számára az értesítési útvonalat üzenettel jelzett megszakítások (MSI-X) használják, amelyek minden megszakítást egy adott CPU-hoz irányíthatnak. A jól konfigurált IRQ és RSS beállítások segítenek maximalizálni a hálózati kommunikáció hatékonyságát és optimalizálni a rendszer teljesítményét.

A Linux kernelben a csomagtovábbítás során a skaláris csomagfeldolgozás és az RSS egyaránt fontos szerepet játszik. A skaláris feldolgozás alapvetően egyszerű, de lehet, hogy nem elég hatékony a nagy terhelésű hálózatok esetén. Az RSS segítségével a kernel képes hatékonyabban kezelni a nagy terhelést, és elosztani a csomagok feldolgozását a rendszerben lévő több CPU között, ezáltal növelve a teljesítményt és csökkentve a késleltetést.

A NUMA (Non-Uniform Memory Access, elosztott memóriahozzáférés) egy olyan többprocesszoros rendszer, ahol a memória-hozzáférés ideje függ a memóriának a processzorhoz viszonyított helyzetétől: az egyes processzorok a saját helyi memóriájukat gyorsabban érik el, mint egy másik processzor helyi memóriáját. A NUMA lényege, hogy javítsa a rendszer teljesítményét. [14]

Többféleképpen is megközelíthetjük ezt a rendszert. Ahhoz, hogy jobban megérthessük, két szempontot érintek: a hardver és a Linux szoftver nézőpontjából. [13]

Hardver szempontból a NUMA egy több egységből álló rendszer és mindegyik tartalmazhat 0 vagy több CPU-t, helyi memóriát és/vagy IO buszt. Ezeket az egységeket más néven "celláknak" hívják. A cellák össze vannak kapcsolva, például pont-pont kapcsolattal.

A Linux szempontjából a NUMA rendszerek elsősorban a Cache Coherent NUMA (ccNUMA) rendszerek. Ezekben a rendszerekben minden memória látható és hozzáférhető minden CPU számára, és a gyorsítótárat az összekapcsoló kezeli.

A Linux szoftver nézőpontjából a rendszert hardveres erőforrásokat tartalmazó szoftveres entitásokra, vagy "csomópontokra" (node) osztja fel. Ezeket a csomópontokat a fizikai cellákra képezi le, és a hozzáférések a helyi memóriához általában gyorsabbak, mint a távoli memóriához való hozzáférések.

A Linux minden csomóponthoz külön memóriakezelő rendszert hoz létre, és minden memóriazónához egy zonelistát. A zonelistában szereplő zónák/node-ok azok, amelyeket a rendszer az adott memóriaallokáció igényeinek kielégítése érdekében keres.

Az alapértelmezett beállítások szerint a Linux azon a csomóponton próbálja meg kielégíteni a memóriaallokációs kéréseket, amelyhez az adott kérés kiszolgálásáért felelős CPU van rendelve. Ha a helyi csomópont nem képes kielégíteni a kérést, akkor a rendszer más csomópontok zónáit is megvizsgálja, hogy megtalálja az első olyan zónát, amely képes kielégíteni a kérést.

A NUMA rendszernek számos előnye van, például a gyors adatmozgás és az alacsonyabb késleltetés a multiprocessing rendszerben. Azonban hátrányok is vannak, például a magas költség és a nagyobb konfigurációkhoz való nehéz implementáció.

2.6. FD.io VPP

Az FD.io [5] a Fast Data Project-nek a rövidítése, ami egy nyílt forráskódú ingyenes szoftver. FD.io a VPP (Vector Packet Processing) gyors, és képes nagy teljesítményű forgalmat kezelni. Több platformon is használható. Mit jelent az, hogy vektor csomagfeldolgozás? Két fajta csomagfeldolgozás létezik az egyik a skalár a másik pedig a vektor. Azok a programok használják a vektor csomagfeldolgozást, amiknek nagy a teljesítmény igénye. Azért képes nagyobb teljesítményre, mert szemben a skalár feldolgozással, egyszerre több csomagot is képes fogadni, majd ezt a csomag vektort átadja a feldolgozó funkciónak, ami azt feldolgozza, így spórolva időt.

A VPP egy gráf alapú továbbítási modellt használ, ami miatt könnyen lehet bővíteni és a pluginokat saját igényeink szerint alakíthatjuk. Ez és a vektorcsomag feldolgozás hozzájárul a magas áteresztőképességéhez. Három dolgot emelnék ki a VPP architektúrájából, amik lényegesen hozzájárulnak a gyorsaságához:

- Egy session layer, amely megkönnyíti a szállítási protokollok és alkalmazások közötti interakciót
- Cserélhető (pluggable) szállítási protokollok, beleértve a TCP-t, a QUIC-ot, a TLS-t, az UDP-t
- VCL (VPPComs könyvtár), egy olyan könyvtárcsomag, amelynek célja a host stack használatának megkönnyítése az alkalmazás szempontjából

Az FD.io VPP tervezésének központjában a csomagfeldolgozási grafikon (Packet Processing Graph) áll.

A csomagfeldolgozási grafikon azt jelenti, hogy bárki új grafikus csomópontokat csatlakoztathat. Emiatt könnyű a VPP fejlesztése, és azt is jelenti, hogy a plugineket specifikus célokra lehet testre szabni. A VPP konfigurálható a Low-Level API segítségével.

A FD.io VPP összegyűjti a csomagok vektorát az RX gyűrűkből, akár 256 csomagot is egyetlen vektorban. Ezután a csomagfeldolgozási grafikon alkalmazása történik, csomópontonként (beleértve a plugineket) az egész csomagvektorra. A fogadott csomagok bejárják a vektorban lévő csomagfeldolgozási grafikon csomópontjait, ettől a hálózati feldolgozás által képviselt minden grafikus csomópont alkalmazódik az egyes csomagokra. A grafikus csomópontok kicsik, modulárisak, és laza kapcsolatban vannak. Emiatt egyszerűbb újabb grafikus csomópontok bevezetése és a meglévők módosítása.

A pluginek osztozott könyvtárak, és a VPP futásideje alatt töltődnek be. A VPP a plugineket a könyvtárak keresésével találja meg, majd a futtatás során dinamikusan tölti be őket. Egy plugin teljesen függetlenül építhető a FD.io VPP forrásfájltól, ami azt jelenti, hogy független komponensként kezelhető.

2.7. Siitperf

Ebben a részben bemutatom a Siitperf mérőprogramot, ami a Tester gépeken futott. A siitperf neve onnan származik, hogy eredetileg SIIT (Stateless IP/ICMP Translation) átjárók teljesítményének a mérésére készült [6]. Rugalmassága miatt alkalmas IPv4 és IPv6 csomagtovábbítók (routerek) teljesítményének a mérésére is. A Siitperf célja, hogy egy olyan mérést lehessen vele végrehajtani, ami nagy teljesítményképességű, és könnyen lehessen alkalmazni kutatásoknál. A Siitperf felépítését illetően a legjobban, úgy lehet jellemezni, mint binárisok és shell scriptek halmazát. A binárisok feladata IPv4 vagy IPv6 adatcsomagokat tartalmazó Ethernet tesztkeretek küldése és fogadása, figyelembe véve a keretrátát (küldött

vagy fogadott keretek száma másodpercenként). A binárisokat C++ nyelven valósították meg, DPDK segítségével mennek a mérések, hogy elérjék a kellően magas teljesítményt. A siitperf négy mérési eljárást valósított meg: áteresztőképesség (throughput) csomagvesztés (frame loss rate), késleltetés (latency), PDV (packet delay variation). Az első háromnak a szempontjai ugyanazok, mint az RFC 2544-ben bemutatottam. A negyedik az RFC 8219-ben van leírva. A négy mérést a siitperf három darab bináris állomány segítségével valósítja meg, ugyanis a throughput és a frames loss rate mérés elemi lépése azonos, és a siitperf-tp binárisban lett megvalósítva, míg a két különböző benchmark eljárásához két különböző bash shell script lett írva és az is végzi el őket. A throughput mérés bash shell scriptje bináris keresést valósít meg annak érdekében, hogy meghatározza azt legmagasabb keretrátát (frame rate) értéket, amely mellett még veszteségmentes a csomagtovábbítás. A frame loss rate mérés bash shell scriptje pedig különböző keretráták végez méréseket, és meghatározza a csomagvesztési arányt. A késleltetés (latency) és a PDV (Packet Variation Delay) benchmark eljárásoknál a siitperf-lat és siitperf-pdv binárisok valósítják meg az elemi mérést, és a megfelelő bash shell scriptek hívják meg őket. A késleltetés és a PDV méréseket az áteresztőképesség (throughput) mérés által meghatározott keretrátát (frame rate) mellett kell elvégezni. A C++ programok az egyes mérés sorozatok alatt állandó paramétereiket a konfigurációs fájlból veszik. Amelyek viszont változhatnak, azokat parancssorból kapják az őket meghívó bash shell scripttől. Ezután a program meghívja a szükséges mérés inicializáló funkcióját, amely a DPDK Absztrakciós Rétegét (EAL) használva, visszaállítja és elindítja a hálózati interfészeket, és néhány ellenőrzést is lefuttat. Végül a fő program végrehajtja a megfelelő mérési eljárást. A mérési eljárás elindít egy-egy küldőt és fogadót az aktív irányonként (külön szálakban). Ezeket az adott CPU magokkal garantált teljesítmény érdekében végzik el. Miután a küldő és fogadó szálak befejeződtek, a fő szál összegyűjti és kiértékeli az eredményeket. A bináris program a standard outputra kiírt eredményeket a bash shell scriptek dolgozzák fel, és a bináris program többszöri futtatásával előállítják a felhasználó számára a végső eredményeket. [7]

3. Az elvégzett munka

Ebben a fejezetben arról olvashatnak, milyen eszközöket használtam, hogyan konfiguráltam őket. A második felében az FD.io telepítéséről és annak konfigurálásáról lesznek fejezetek. Végül pedig a mért eredményeket fogom elemezni.

3.1. Feladat ismertető

Az IP-nek jelenleg két verziója van használatban: a régi 4-es verzió (IPv4) és a jelenleg terjedőben levő új, 6-os verzió (IPv6). Munkámban megvizsgálom az IPv4 és IPv6 csomagtovábbítást Linux kernelen és FD.io VPP-én. Dolgozatomban az FD.io VPP szoftver vizsgálata kapta a legnagyobb figyelmet, mert azt állítják, hogy ők a „Világon a leggyorsabb és legbiztonságosabb hálózati csomagtovábbítók”, ezt az állítást fogom megvizsgálni. Ehhez pedig a kiválasztott megoldások teljesítményét és skálázhatóságát is összehasonlítom: megvizsgálom, hogyan változnak az eredmények a felhasznált CPU magok számának függvényében. Munkámban nem csak a méréseimet mutatom be, hanem a szoftver telepítését és használatát is.

3.2. Mérési összeállítás

3.2.1. Fizikai szinten

Az egyetemen konzulensemtől két darab szervert kaptam a méréseimhez. Dolgozatom során a szerver párosra DUT2 és TESTER2 néven fogok hivatkozni, mert így hívták a szervereket.

A mérések során a DUT2 és a TESTER2 is egy-egy Dell PowerEdge R620 szerver volt, Intel Xeon E5-2620 processzorokkal, amikbe Intel X540 10G/1G hálózati kártyákat szereltem be. A CPU 2000 MHz-es, de a VPP mérések során az órajelet átállítottam 1200 MHz-re (/etc/tlp.conf), annak érdekében, hogy a VPP nagy teljesítmény ellenére is meg tudjam mérni legalább 2 worker thread használata esetén is csomagtovábbítás teljesítményét. A szervereket közvetlenül Cat6 UTP patch kábelekkel kötöttem össze.

A mérések szempontjából fontos információ, hogy a használt szerverekben a hálózati interfészek és a páros számú CPU magok (0, 2, 4, 6, 8, 10) a 0-s NUMA node-hoz, a páratlan számú CPU magok pedig az 1-es NUMA node-hoz tartoznak.

A szerverek főbb paramétereit az 1. táblázatban foglaltam össze.

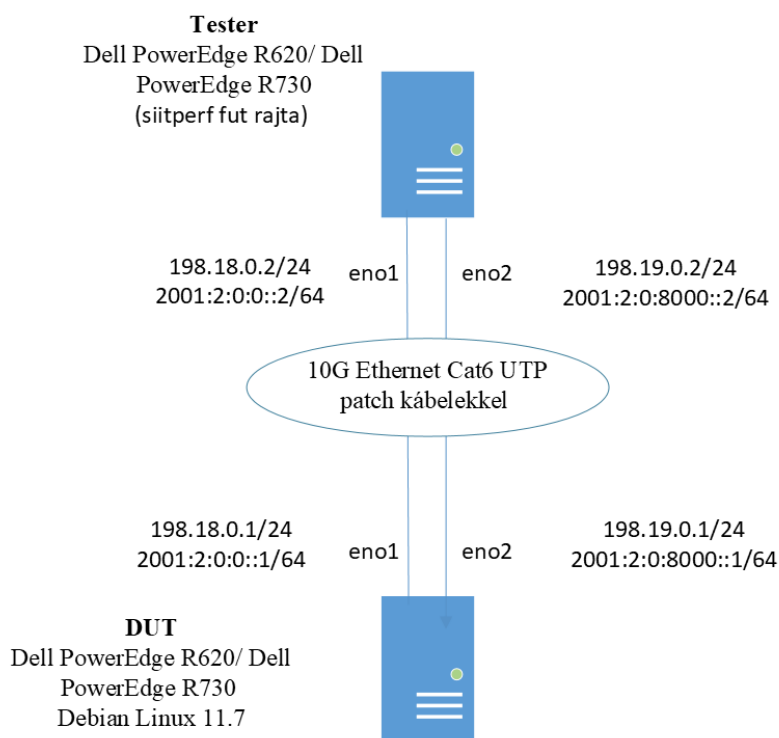
1. táblázat Hardware adatai

Szerverek nevei	DUT2/TESTER2
Szerverek típusai	Dell PowerEdge R620
Processzor típusa	Intel Xeon E5-2620
Processzor órajele	2.00 GHz
Processzorok száma	2 db
RAM mennyisége	32 GB
RAM modulok száma	4 db
Egy RAM modul mérete	8 GB
RAM órajel	1333 MHz, DDR3
Hálózati kártya típusa	Intel X540 10G/1G
Hálózati kártya sebessége	10G/1G

3.2.2. IP szinten

Ebben az alfejezetben bemutatom milyen operációs rendszert használtam, annak verzióját és hogyan lettek az interfészek összekötve az általam használt szervereken.

A TESTER2 szerveren az eredetileg rajta levő Debian 9.13 operációs rendszert használtam, mert a siitperf használt verziója azt igényelte. A DUT2-re Debian 11.7-et telepítettem és a kernelt 5.10.0-25-ös verzióra frissítettem. A mérési összeállítás a 3. ábrán látható.



3. ábra Mérési összeállítás

Egy méréshez használt szervernél miután feltelepítettük a számunkra alkalmas operációs rendszert, mindig az IP címek beállításával kezdtem. A DUT2 az eno1 és eno2-re interfészekre állítottam be statikus IP címeket. Ezek rendre a következők IPv4-nél: 198.18.0.1/24 és 198.19.0.1/24. IPv6 esetén pedig: 2001:2:0:0::1/64 és 2001:2:0:8000::1/64.

Létrehoztam egy DUT-Settings nevű könyvtárat, amiben különböző scripteket hoztam létre. Ezek a scriptek segítettek, abban, hogy a különböző mérésekhez gyorsan és hatékonyan be tudjam állítani a megfelelő beállításokat. Mivel a felhasznált mérőprogram nem képes az ARP illetve NDP kérésekre válaszolni, az IP verzióknak megfelelően nekem kellett beállítanom a DUT2- n a TESTER2-höz tartozó IP-cím MAC-cím összerendelési információkat.

A következő parancsokat használtam:

```
ip neighbor add 198.18.0.2 lladdr 24:6e:96:3b:fb:00 dev eno1 nud permanent
ip neighbor add 198.19.0.2 lladdr 24:6e:96:3b:fb:02 dev eno2 nud permanent
```

```
ip neighbor add 2001:2:0:0::2 lladdr 24:6e:96:3b:fb:00 dev eno1 nud permanent
ip neighbor add 2001:2:0:8000::2 lladdr 24:6e:96:3b:fb:02 dev eno2 nud permanent
```

A „permanent” feladata pedig az, hogy a bejegyzés az ARP/NDP cache-ben maradjon, és ne legyen automatikusan törölve.

Az RFC-k követelményeinek megfelelően a mérések során a siitperf az UDP szállítási szintű protokollt használta a mérések során. Ennek megfelelően beállítottam, hogy a hálózati interfészek hash függvényébe ne csak a forrás és cél IP-címek, hanem a forrás és cél UDP portszámok is bekerüljenek.

IPv4 illetve IPv6 esetén az alábbi parancsokat használtam:

```
ethtool -N eno1 rx-flow-hash udp4 sdfn
ethtool -N eno2 rx-flow-hash udp4 sdfn
ethtool -n eno1 rx-flow-hash udp4
ethtool -n eno2 rx-flow-hash udp4
```

```
ethtool -N eno1 rx-flow-hash udp6 sdfn
ethtool -N eno2 rx-flow-hash udp6 sdfn
ethtool -n eno1 rx-flow-hash udp6
ethtool -n eno2 rx-flow-hash udp6
```

Mindegyik csoportban az első két parancs végezte a beállítást, az utána következő két parancs pedig a beállított érték lekérdezését, amire beállítás sikerességének ellenőrzéséhez volt szükség.

3.2.3. Siitperf

Ebben a fejezetben a TESTER2-re telepített mérőprogram fájljait bemutatom, amiket használtam a mérések során. Két fájlt módosítottam az egyik a binary-rate-alg.sh, a másik a siitperf.conf.

A binary-rate-alg.sh fájlnak a paramétereit módosítottam és így néz ki:

```
#Parameters
impl="tester2-10G-dut2-1c-D11-k5-10-IPv6" # vizsgált implementáció neve
dir="b" # érvényes értékek: b,f,r; b: bidirectional (kétirányú), f: forward
(balról jobbra, 6 --> 4), r: reverse (jobbról balra, 6 <-- 4)
max=4000000 # maximum csomagszám
fs=84 # IPv6 keretméret; IPv4 keretméret mindig 20 bájt nál kevesebb
xpts=60 # mérési időtartalom (perben)
to=2000 # időtúllépés másodpercben
e=1 # mérési hiba: a különbség a bináris keresés felső és alsó korlátjának
értékei között, amikor bejezi a keresést
no_exp=20 # kísérletek száma
res_dir="results" # eredmények könyvtára
```

A binary-rate-alg.sh fájlal állítottam be a mérések paramétereit, és azt is, hogy az eredményeket milyen néven mentse el. Itt adtam meg a maximum keretrátát, amit a mérés során használhat. A bináris keresés „hiba” értéket állítottam még itt. Ez adja meg a bináris keresés leállási feltételét, amit az alábbi képlettel lehet meghatározni:

$$felső_limit - alsó_limit \leq hiba$$

1. egyenlet A bináris keresés leállási feltétele

A pontos eredmények elérése érdekében a hiba érték mindig 1 volt.

És beállítottam az iterációk számát is, hogy hányszor fusson le a mérés, ez pedig a kellően megbízható eredmények elérése érdekében 20 volt.

A siitperf.conf fájljának pedig az alábbi részeiben végeztem módosításokat:

```
IP-L-Vers 6 # Baloldali küldő IP verziója
IP-R-Vers 6 # Jobboldali küldő IP verziója
```

```

# A szerverek IP címei:
IPv4-L-Real 198.18.0.2
IPv4-R-Real 198.19.0.2
IPv6-L-Real 2001:2:0:0::2
IPv6-R-Real 2001:2:0:8000::2

# A szerverek MAC címei:
MAC-L-Tester 24:6e:96:3b:fb:00 # tester2 eno1
MAC-R-Tester 24:6e:96:3b:fb:02 # tester2 eno2
MAC-L-DUT ec:f4:bb:da:8f:70 # dut2 eno1
MAC-R-DUT ec:f4:bb:da:8f:72 # dut2 eno2

# RFC 4814 random port paraméterei

Fwd-var-sport 3 # A forrásport: 0: fix, 1: nő, 2: csökken, 3: álvéletlen
Fwd-var-dport 3 # A célport: 0: fix, 1: nő, 2: csökken, 3: álvéletlen
Fwd-sport-min 1024
Fwd-sport-max 65535
Fwd-dport-min 1
Fwd-dport-max 49151
Rev-var-sport 3 # A forrásport: 0: fix, 1: nő, 2: csökken, 3: álvéletlen
Rev-var-dport 3 # A célport: 0: fix, 1: nő, 2: csökken, 3: álvéletlen
Rev-sport-min 1024
Rev-sport-max 65535
Rev-dport-min 1
Rev-dport-max 49151

```

A siitperf.conf fájlban beállítottam, hogy a mérések során milyen IP verziós mérést fogok végrehajtani, a két szervernek mindkét IP címeit az eno1 és eno2 interfészen, és ezeknek a MAC címeit. A mérések random port mérések voltak az RFC 4814 alapján.

3.3. FD.io telepítése

A dolgozatomban az FD.io szoftvert fogom részletesebben bemutatni, annak telepítését és használatát. Az FD.io hivatalos oldala alapján igazodtam el a program telepítésekor. Négy csomagot kellett letölteni, én a 22.06-os verziót választottam, mert 2023-ban ez volt a stabil verzió szám. Az alábbi négy csomagra volt szükségem: libvppinfra, vpp, vpp-plugin-core, vpp-plugin-dpdk. Ezzel a négy csomag telepítésével már készen is áll az FD.io környezetünk.

Több fajta utat is választhatunk a csomagok beszerzése során. A DUT2-re való telepítéskor a hivatalos oldal packagecloud.io-ról választottam ki és töltöttem le a számomra megfelelő verziót és az operációs rendszeremmel kompatibilis csomagot.

Majd így telepítettem őket:

```
apt install ./libvppinfra_22.06-release_amd64.deb ./vpp_22.06-  
release_amd64.deb ./vpp-plugin-core_22.06-release_amd64.deb ./vpp-plugin-  
dpdk_22.06-release_amd64.deb.
```

Végül még egy újraindításra volt szükségem.

3.3.1. FD.io beállítása

Azonban használatához, még szükségem volt a DPDK csomagtovábbító csomag letöltésére is. Ezt egyszerűen az **apt install dpdk** és **apt install dpdk-dev** parancsokkal megtettem.

Ahhoz, hogy a hálózati interfészeket ne a Linux kernel kezelje, hanem a DPDK, és azon keresztül az FD.io VPP, a **bind-interface** nevű srciptemmel a hálózati interfészeket a DPDK driverrel kellett összekötnöm. Ehhez a kernel szempontjából „le kellett kapcsolnom” az interfészeket, amiket át akartam adni a DPDK-nak:

```
ifconfig eno1 down  
ifconfig eno2 down
```

Majd leválasztottam az eddig használt driver-éről:

```
dpdk-devbind.py -u 0000:01:00.0  
dpdk-devbind.py -u 0000:01:00.1
```

Ezután betöltöttem az **uio_pci_generic** drivert:

```
modprobe uio_pci_generic
```

Végül a DPDK-át támogató driverrel összekötöttem az interfészeket:

```
dpdk-devbind.py --bind=uio_pci_generic 0000:01:00.0  
dpdk-devbind.py --bind=uio_pci_generic 0000:01:00.1
```

Az alábbi parancs megmutatja a jelenlegi interfészeket, a PCI bus címüket, és azt, hogy milyen driverrel vannak összekötve:

```
dpdk-devbind.py --status
```

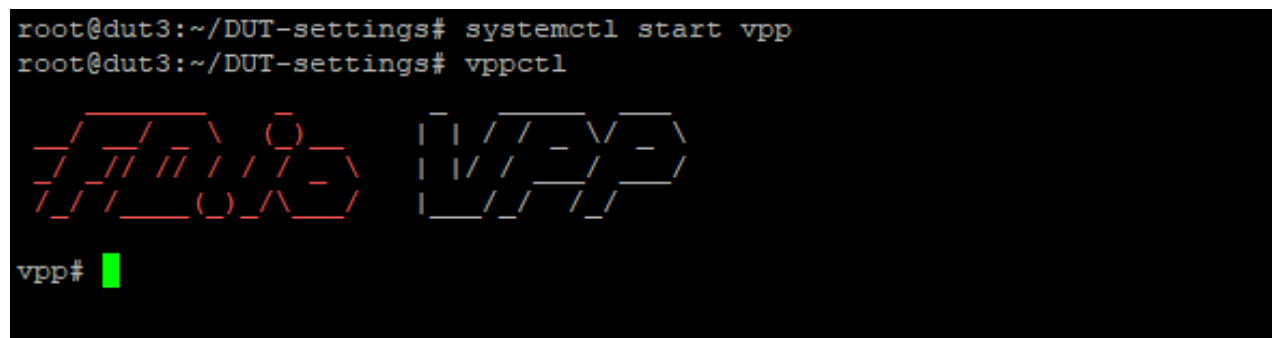
```
Network devices using DPDK-compatible driver  
=====
```

```
0000:01:00.0 'Ethernet Controller 10-Gigabit X540-AT2 1528'
drv=uiopci_generic unused=ixgbe,vfio-pci
0000:01:00.1 'Ethernet Controller 10-Gigabit X540-AT2 1528'
drv=uiopci_generic unused=ixgbe,vfio-pci
```

Network devices using kernel driver

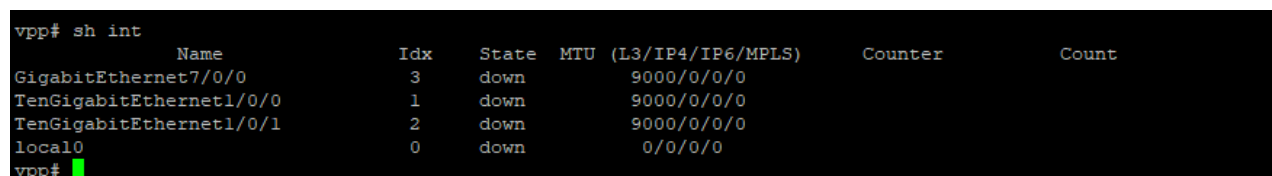
```
=====
0000:08:00.0 'I350 Gigabit Network Connection 1521' if=eno3 drv=igb
unused=vfio-pci,uiopci_generic
0000:08:00.1 'I350 Gigabit Network Connection 1521' if=eno4 drv=igb
unused=vfio-pci,uiopci_generic *Active*
```

Miután készen voltam a telepítéssel, és az interfészeket átadtam a DPDK-nak, neki is kezddhettem a VPP konfigurálásához. A VPP elindításához a **systemctl start vpp** parancsra volt szükségem, leállításához pedig a **systemctl stop vpp**-re. A FD.io VPP beállításainak a szerkesztéséhez pedig egyszerűen csak **vppctl** parancsot adtam ki. Ekkor az FD.io felülete fogadott, ami így nézett ki:



4. ábra FD.io VPP grafikus felülete

Az FD.io felülete hasonlít a Cisco programozási felületéhez, és a parancsai beszédes angol szavakból állnak. Például az interfészeket a **show interface** (rövidítve **sh int**) paranccsal tudtam megnézni. A képen láthatóak a DPDK-val összekötött interfészek nevei, amik nagyon hasonlítanak a Cisco által használtakéhoz.



Name	Idx	State	MTU	(L3/IP4/IP6/MPLS)	Counter	Count
GigabitEthernet7/0/0	3	down	9000	0/0/0/0		
TenGigabitEthernet1/0/0	1	down	9000	0/0/0/0		
TenGigabitEthernet1/0/1	2	down	9000	0/0/0/0		
local0	0	down	0	0/0/0/0		

5. ábra FD.io VPP interfészei

Ezen a felületen be kellett állítanom az IP címeket és az ip neighbor-okat, ezt az alábbi módokon tettem IPv4 esetén:

```
set interface ip address TenGigabitEthernet1/0/0 198.18.0.1/24
set interface ip address TenGigabitEthernet1/0/1 198.19.0.1/24
set interface state TenGigabitEthernet1/0/0 up
set interface state TenGigabitEthernet1/0/1 up

set ip neighbor TenGigabitEthernet1/0/0 198.18.0.2 24:6e:96:3b:fb:00
set ip neighbor TenGigabitEthernet1/0/1 198.19.0.2 24:6e:96:3b:fb:02
```

A VPP-nél is alkalmaztam a CPU magok számának állítását, ezt az /etc/vpp/startup.conf fájlban tettem meg. Ott kikerestem a CPU szegmenst, és az alábbi két részt írtam át:

```
## Set logical CPU core where main thread runs, if main core is not set
## VPP will use core 1 if available
main-core 2

## Set logical CPU core(s) where worker threads are running
corelist-workers 5
```

A „main-core”-on fut az FD.io VPP fő szála. Ha nem adunk meg ún. „worker”-eket, akkor a fő szál végzi a csomagfeldolgozást is. Ha viszont megadunk workereket, akkor a csomagfeldolgozást teljes egészében a megadott worker(ek) végzi(k), a fő szál csak a VPP keretrendszer működéséért felel.

3.4. Mérési eredmények

Ebben a fejezetben bemutatom, méréseim eredményeit, amit táblázatokba rendeztem.

3.4.1 A Linux kernel csomagtovábbítási teljesítménye

Először a Linux kernel IPv4 és IPv6 csomagtovábbítási teljesítményét mértem meg. A csomagtovábbítási teljesítmény leméréséhez megadtam a megfelelő beállításokat a mért eszközön. Ez látható a 3.2.2 fejezetben.

A Linux kernel IPv4 és IPv6 csomagtovábbításának áteresztőképességét 1, 2, 4, 6, 8 és 12 aktív CPU mag használatával mértem annak érdekében, hogy megvizsgáljam a tesztrendszer teljesítményét, valamint betekintést nyújtson a skálázhatóságába. A használt CPU magok fenti száma egy kompromisszumot tükröz két szempont között:

- A futási idő megtakarítás érdekében minél kevesebb mérés végrehajtása.
- Kellően alapos vizsgálat.

Ha a rendszerben található CPU magok száma 2 hatvány lett volna (például 16), akkor ideális lett volna mindig dupláznom az aktív CPU magok számát (1, 2, 4, 8, 16). Így azonban a 12 miatt még 6 CPU maggal is elvégeztem a mérést.

A DUT aktív CPU magjainak a számát a **maxcpus=n** kernel command line paraméterrel állítottam be.

Az IPv4 méréseknél a keretméret 64 byte volt, az IPv6 méréseknél pedig 84 byte. Az utóbbinak az volt az oka, hogy a siitperf IPv6 esetén 84 byte-nál kisebb keretméretet nem enged meg.

A teszteket minden paraméterkombináció mellett hússzor hajtottam végre.

A többször elvégzett mérések összesített eredményét gyakran az eredmények átlagával, az eredmények konzisztens (vagy éppen inkonzisztens) jellegét pedig az eredmények szórásával szokták jellemezni. Azonban az átlag érzékeny az esetleges „kilógó” mérési eredményekre, ezért én a vizsgálataim során főleg az arra nem érzékeny mediánt használom, de az előzőekben említett értékeket is megadom. Az eredmények konzisztenciájának a jellemzésére pedig diszperziót használom, amit úgy kapunk meg, hogy a maximum és minimum különbségét elosztjuk a mediánnal és megszorozzuk százszal, hogy százalékként kapjuk meg. A vizsgálatok megkönnyítése érdekében kiszámoltam még azt is, hogy egy adott CPU magszám melletti teljesítmény hogyan viszonyul a feleannyi aktív CPU magszám melletti teljesítményhez.

Végül megadtam még az 1 maghoz képesti *relatív skálázódást* is:

$$S(n) = P(n)/n$$

2. egyenlet Relatív skálázódás egy maghoz viszonyítva

Ahol n az aktív CPU magok számát, $P(n)$ az n aktív CPU mag melletti teljesítményt jelenti. Ennek értéke ideális esetben 1, ami azt jelenti, hogy a rendszer teljesítménye az aktív CPU magok számának függvényében lineárisan nő.

A Linux kernel IPv4 csomagtovábbításának teljesítményét jellemző throughput méréseim eredményei az 2. táblázatban láthatók. Fontosnak tartom megjegyezni, hogy az RFC 2544 követelményeinek megfelelően a mérést kétirányú forgalommal végeztem, de a siitperf az irányonkénti keretrátát várja input paraméterként és a mérés során elküldött/vett keretek számát is irányonként adja meg. Ennek megfelelően a táblázatban szereplő fps (frames per second, azaz: keret pár másodperc) értékek is irányonként értendők. (A kétirányú forgalom miatt a DUT kétszer ennyi keretet továbbított másodpercenként.)

Ami az eredmények minőségét illeti, azok minden esetben kellően stabilak, így alkalmasak a

2. táblázat Linux kernel Pv4 csomagtovábitás teljesítménye (átbocsátóképesség) az aktív CPU magok számának függvényében.

aktív CPU magok száma (db)	1	2	4	6	8	12
medián (fps)	442782	765223	1524287	2265108	2788933	3923150
minimum (fps)	442352	764369	1521483	2218749	2781247	3874999
maximum (fps)	443848	768128	1525409	2268067	2790527	3929688
átlag (fps)	442889	765329	1524244	2262774	2787973	3918181
szórás	363,64	814,01	981,57	10549,55	2234,20	15163,08
diszperzió (%)	0,34	0,49	0,26	2,18	0,33	1,39
fele annyi maghoz képest	--	1,7282	1,9920	--	1,8297	1,7320
relatív skálázódás 1 maghoz	n/a	0,8641	0,8606	0,8526	0,7873	0,7384

tendenciák vizsgálatára. A diszperzió (és a szórás is) 6 CPU mag esetén érte el a legmagasabb értékét (2,18%), ami még mindig kellően alacsony.

Az eredmények már első ránézésre is azt mutatják, hogy a DUT teljesítménye alapvetően jól skálázódik az aktív CPU magok számának növelésével. Az alaposabb vizsgálat a következőket mutatja:

Amikor 1 CPU mag helyett 2 CPU magot használtam, a rendszer teljesítménye nem a duplájára, hanem csak 1,7282 szerezére nőtt. Ezt a jelenséget két fő oka van:

1. Egy rendszer többmagos működésének teljesítmény költsége van az egyetlenmagos működéshez képest,
2. A 0-s CPU mag és a méréshez használt 10GbE Ethernet interfészek a 0-s NUMA node-hoz tartoznak, de az 1-es CPU mag az 1-es NUMA node-hoz tartozik, ami azt jelenti, hogy az 1-es CPU mag csak a 0-s magon keresztül tud a hálózati interfésszel kommunikálni: ez természetesen csökkenti a teljesítményt.

Utána viszont azt figyelhetjük meg, hogy amikor 2 helyett 4 aktív CPU magot használtam, akkor a teljesítmény igen jó közelítéssel duplázódott (1,992-szeresére nőtt). Ennek a magyarázata nagyon egyszerű. A fenti tényezők már a kétmagos rendszerben is jelen voltak, így a négymagos rendszerben nem okoztak további teljesítményromlást.

A továbbiakban a 4 magos rendszerhez képes a 8 magos rendszer teljesítménye már csak 1,8297-szeresére, a 6 magos rendszerhez képes a 12 magos rendszer teljesítménye már csak 1,732-szeresére nő. Ennek a csökkenésnek már más okai lehetnek, például az, hogy az összes CPU mag ugyanazokat a hálózati interfészeket használja, és a számuk növekedésével a konkurens hozzáférésükkel már bizonyos mértékben akadályozzák egymást.

Összességében megállapítható, hogy rendszer teljesítménye jól skálázódott: 1 CPU mag helyett 12 CPU mag használata esetén a throughput 442 782 fps-ról 3 923 150 fps-ra nőtt, ami 8,86-

szoros érték.

3. táblázat Linux kernel Pv6 csomagtovábbítás teljesítménye (átbocsátóképesség) az aktív CPU magok számának függvényében.

aktív CPU magok száma (db)	1	2	4	6	8	12
medián (fps)	425782	732299	1466585	2176369	2658915	3766926
minimum (fps)	424951	726561	1374999	1999999	2617186	3761709
maximum (fps)	426026	733467	1466974	2178284	2660414	3772095
átlag (fps)	425628	731888	1461745	2156176	2656771	3766972
szórás	291,61	1481,93	20459,50	54645,21	9443,81	2679,58
diszperzió (%)	0,25	0,94	6,27	8,19	1,63	0,28
fele annyi maghoz képest	--	1,7199	2,0027	--	1,8130	1,7308
relatív skálázódás 1 maghoz	n/a	0,8599	0,8611	0,8519	0,7806	0,7373

A Linux kernel IPv6 csomagtovábbításának teljesítményét jellemző throughput méréseim eredményei a 3. táblázatban láthatók. Ezek az eredmények alapvetően nagyon hasonlóak az 2. táblázatban szereplő eredményekhez. Két feltűnő különbség van:

- A diszperzió értékei 4 és 6 CPU mag mellett feltűnően nagyobbak. Ennek oka, hogy 4 CPU mag mellett az egyik 1 375 000 fps, 6 CPU mag mellett pedig az egyik 2 millió fps keretráta melletti teszt sikertelen volt. Ezeknek az oka valamilyen a rendszerben történő viszonylag ritka esemény lehet. Az átlagot már láthatóan befolyásolják, éppen ezért használtam a vizsgálataimhoz medián értékeket.
- Az IPv6 throughput kismértékben alacsonyabb, mint az IPv4 throughput. Ennek két oka lehetséges: egyrészt az IPv6 mérésnél a keretméret nagyobb volt, másrészt pedig magyarázata az, hogy az IPv6 címek négyszer olyan hosszúak, mint az IPv4 címek.

3.4.2 Az FD.io VPP csomagtovábbítási teljesítménye

Továbbiakban azokat a méréseket fogom bemutatni, ahol már használtam az FD.io VPP-t. IPv4 és IPv6 csomagtovábbítási teljesítmény leméréséhez megadtam a megfelelő beállításokat a mért eszközön. Ez látható a 3.3.1 fejezetben.

A FD.io VPP IPv4 és IPv6 csomagtovábbításának áteresztőképességének mérésekor 2-es CPU mag volt a main core, és 1 illetve 2 workert használtam, amit a páros (4, 6, 4 és 6), vagy a páratlan (3, 5, 3 és 5) számú CPU magokon futtattam annak érdekében, hogy megvizsgáljam a tesztrendszer teljesítményét, valamint betekintést nyújtsok a skálázhatóságába.

A tesztek minden beállítás mellett hússzor hajtottam végre. A CPU órajel frekvenciát 1200 MHz-re állítottam be, így jobban tudtam demonstrálni a FD.io VPP igazi erejét. (Erre azért volt szükség, mert az előzetes méréseim idején kiderült, hogy a névleges 2GHz CPU órajel mellett már 2 worker esetén is a Tester volt a szűk keresztmetszet, és a DUT valódi teljesítményét nem

tudtam megmérni.)

4. táblázat IPv4 csomagtovábbítás elemzése CPU magok számának változtatásával és FD.io VPP használatával.

használt CPU magok	4-es	6-os	4-es és 6-os	3-as	5-ös	3-as és 5-ös
workerek száma (db)	1	1	2	1	1	2
medián (fps)	2091964	2096376	4187904	1700940	1697002	3432319
minimum (fps)	2089688	2093309	4183576	1699461	1695153	3390624
maximum (fps)	2093880	2099060	4238282	1702287	1698181	3448609
átlag (fps)	2091944	2096289	4191721	1700981	1696889	3431086
szórás	1501,18	1721,36	12034,38	849,40	726,75	13274,60
diszperzió (%)	0,20	0,27	1,31	0,17	0,18	1,69

5. táblázat IPv6 csomagtovábbítás elemzése CPU magok számának változtatásával és FD.io használatával.

aktív CPU magok	4-es mag	6-os mag	4-es és 6-os	3-as	5-ös	3-as és 5-ös
workerek száma (db)	1	1	2	1	1	2
medián (fps)	1919142	1926082	3876009	1526365	1587369	3188186
minimum (fps)	1916951	1923811	3866209	1517536	1585936	3124999
maximum (fps)	1921052	1928529	3908204	1528194	1588904	3195313
átlag (fps)	1918898	1926321	3878199	1525300	1587528	3185315
szórás	1120,34	1119,19	10495,01	3163,36	746,77	14657,38
diszperzió (%)	0,21	0,24	1,08	0,70	0,19	2,21

A hibaváltozót itt is 1-re állítottam be.

A worker thread-ek futtatására használt CPU magok darabszáma, illetve sorszáma alapján rendeztem táblázatba a mért eredményeket. Ezeken a sorokon most is egyenként végig megyek az elemzés során.

Az FD.io VPP IPv4 csomagtovábbításának teljesítményét jellemző throughput méréseim eredményei a 4. táblázatban láthatók. Megvizsgálva a 4-es és a 6-os magot, azt láthatjuk, hogy a teljesítményben nincsen érdemi különbség (2 091 964 fps és 2 096 376 fps), és ha két workert használtam a 4+6 mag páron, akkor pedig a medián értéke jó közelítéssel megduplázódott (4 187 904 fps). Ugyanez elmondható a páratlan magokról is. Azonban a páratlan magoknál a medián értéke több mint 18%-kal alacsonyabb lett, ami egyértelműen az eltérő NUMA node-nak tulajdonítható.

Ami az eredmények minőségét illeti, egy worker thread mellett a diszperzió minden esetben 0,3% alatt van, tehát az eredmények nagyon stabilak. Két worker thread esetén a diszperzió jelentősen nő, de még így is 2% alatt marad.

Az FD.io VPP IPv6 csomagtovábbításának teljesítményét jellemző throughput méréseim eredményei a 5. táblázatban láthatók. Ezek az eredmények alapvetően nagyon hasonlóak az 3. táblázatban szereplő eredményekhez. Egy feltűnő különbség van: az IPv6 throughput kismértékben alacsonyabb, mint az IPv4 throughput. Ennek lehetséges okait a 3.4.1

alfejezetben már megadtam. Egyébként ugyanazok a tendenciák mutathatók ki, mint a 3. táblázat esetén.

3.4.3. Összehasonlítás

A Linux kernel és az FD.io VPP csomagtovábbítási teljesítményének összehasonlítása során fontos szempont, hogy míg a Linux kernel teljesítményét 2GHz, az FD.io VPP teljesítményét 1.2GHz órajel mellett mértem. Az FD.io VPP még ilyen hátrány mellett is meggyőzően teljesített. Míg a Linux kernel 1 CPU magon IPv4 csomagtovábbításban 442 782 fps, IPv6 csomagtovábbításban pedig 425 782 fps teljesítményt nyújtott, addig az FD.io VPP 1 worker thread esetén is több mint 2 millió IPv4 csomagot, illetve több mint 1,9 millió IPv6 csomagot továbbított. (Még ha a hálózati kártyától eltérő NUMA node-hoz tartozó magokat használtam, akkor is kb. 1,7 millió fps illetve 1,5 millió fps volt a teljesítménye.)

Amikor a Linux rendszer mind a 12 CPU magot használta a névleges 2GHz órajel frekvencia mellett, a teljesítménye akkor is csak 3,92Mfps, illetve 3,77Mfps volt IPv4, illetve IPv6 csomagtovábbítás esetén. Ezzel szemben az FD.io csupán 2 CPU mag (a 4-es és a 6-os) használatával, ráadásul 1,2GHz órajel frekvencia mellett több, mint 4,18Mfps illetve 3,87Mfps teljesítményt ért el IPv4, illetve IPv6 csomagtovábbítás esetén. Az eredményeim igazolják, hogy az FD.io VPP valóban egy nagy teljesítményű megoldás IP csomagok továbbítására.

4. Összefoglalás

Dolgozatomban összefoglaltam az IP csomagtovábbító rendszerek közül a Linux kernel és az FD.io VPP működését, valamint a teljesítményük méréséhez szükséges háttérismereteket.

Részletesen ismertettem a Linux kernel és az FD.io VPP IPv4 és IPv6 csomagtovábbítási teljesítményének és skálázhatóságának mérését, amelyet két Dell PowerEdge R620 szerver és a siitperf mérőprogram használatával végeztem el.

IPv4 és IPv6 csomagtovábbítás teljesítményét megmértem mindkét rendszeren, különböző CPU magok számával és beállításokkal. A Linux kernel tesztjei azt mutatták, hogy általában jól skálázódik a CPU magok számának növekedésével, bár voltak kisebb teljesítmény-ingadozások az eltérő NUMA node-ok miatt. Az FD.io VPP tesztjei során megállapítottam, hogy még alacsonyabb órajel mellett is jelentős teljesítményt nyújt, és stabil méréseket produkált.

A FD.io VPP számos esetben túlszárnyalta a Linux kernel teljesítményét, annak ellenére is, hogy alacsonyabb órajellel mértem. Ez azt jelzi, hogy az FD.io VPP valóban hatékony megoldás IP csomagok továbbítására, és versenyképes alternatívát jelent a Linux kernelhez képest.

Irodalomjegyzék

- [1] S. Bradner, J. McQuaid, "Benchmarking methodology for network interconnect devices", IETF RFC 2544 (Mar. 1999)
<https://doi.org/10.17487/RFC2544>
- [2] D. Newman, T. Player, "Hash and stuffing: Overlooked factors in network device benchmarking", IETF RFC 4814, 2008. DOI: 10.17487/RFC4814
<https://www.rfc-editor.org/rfc/rfc4814#section-4.5>
- [3] C. Popoviciu, A. Hamza, D. Dugatkin, "IPv6 benchmarking methodology for network interconnect devices", IETF RFC 5180, 2008. DOI: 10.17487/rfc5180
<https://www.rfc-editor.org/rfc/rfc5180.html>
- [4] Intel® Ethernet Controller E810 Data Plane Development Kit (DPDK)
<https://edc.intel.com/content/www/us/en/design/products/ethernet/config-guide-e810-dpdk/dpdk-overview/>
- [5] FD.io VPP hivatalos oldala
<https://fd.io/docs/vpp/v2101/whatisvpp/extensible>
- [6] G. Lencse, "Design and Implementation of a Software Tester for Benchmarking Stateless NAT64 Gateways", *IEICE Transactions on Communications*, vol. E104-B, no.2, pp. 128-140. February 1, 2021. DOI: 10.1587/transcom.2019EBN0010
- [7] G. Lencse, Making Stateless and Stateful Network Performance Measurements Unbiased
<https://www.hit.bme.hu/~lencse/publications/ECC-2024-multiple-ip-for-review.pdf>
- [8] Scaling in the Linux Networking
Stack<https://www.kernel.org/doc/Documentation/networking/scaling.txt>
- [9] G. Lencse, IPv4, Számítógép-hálózatok előadás fóliák, 2023.
https://www.tilb.sze.hu/tilb/targyak/GKNB_TATM004/05-IPv4-2023.pdf
- [10] DPDK hivatalos oldala
<https://www.dpdk.org/about/>
- [11] G. Lencse, Performance Analysis of ICT Systems
https://www.tilb.sze.hu/tilb/targyak/NGD_MDA64_1/NGD_MDAA64_1-lecture-4-2020.pdf
- [12] Marek Majkowski, "How to receive a million packets per second", The Cloudflare Blog.
<https://blog.cloudflare.com/how-to-receive-a-million-packets>
- [13] The Linux Kernel, What is NUMA
<https://www.kernel.org/doc/html/v4.18/vm/numa.html>

[14] Rahul Awati, non-uniform memory access (NUMA)

<https://www.techtarget.com/whatis/definition/NUMA-non-uniform-memory-access>

Ábrajegyzék

1. ábra A választott mérési összeállítás (forrás: RFC 2544).....	3
2. ábra Kernel és a DPDK csomagok útvonalának összehasonlítása [4]	8
3. ábra Mérési összeállítás.....	16
4. ábra FD.io VPP grafikus felülete.....	21
5. ábra FD.io VPP interfészei	21

Táblázatjegyzék

1. táblázat Hardware adatai	16
2. táblázat Linux kernel Pv4 csomagtovábbítás teljesítménye (átbocsátóképesség) az aktív CPU magok számának függvényében.	24
3. táblázat Linux kernel Pv6 csomagtovábbítás teljesítménye (átbocsátóképesség) az aktív CPU magok számának függvényében.	25
4. táblázat IPv4 csomagtovábbítás elemzése CPU magok számának változtatásával és FD.io VPP használatával.	26
5. táblázat IPv6 csomagtovábbítás elemzése CPU magok számának változtatásával és FD.io használatával.	26

Egyenletjegyzék

1. egyenlet A bináris keresés leállási feltétele.....	18
2. egyenlet Relatív skálázódás egy maghoz viszonyítva	23