

Széchenyi István Egyetem, Távközlési Tanszék

# **IPv4aaS célokat szolgáló IPv6 áttérési technológiák összehasonlító elemzése**

Készítette: Bazsó Ádám

Konzulens: Lencse Gábor

**2022.**

# Tartalomjegyzék

|       |                                                              |    |
|-------|--------------------------------------------------------------|----|
| 1     | Bevezetés .....                                              | 1  |
| 2     | Technológiák bemutatása.....                                 | 2  |
| 2.1   | 464XLAT .....                                                | 2  |
| 2.2   | MAP-E (Mapping of Address and Port with Encapsulation) ..... | 4  |
| 2.2.1 | Alapvető leképezési szabályok (BMR).....                     | 5  |
| 2.2.2 | Továbbító leképezési szabályok (FMR).....                    | 6  |
| 2.3   | MAP-T (Mapping of Address and Port using Translation).....   | 7  |
| 2.4   | Dual-Stack Lite (DS Lite).....                               | 8  |
| 2.4.1 | B4 alkotóelem.....                                           | 8  |
| 2.4.2 | AFTR alkotóelem .....                                        | 8  |
| 2.5   | Lightweight 4over6 .....                                     | 9  |
| 2.6   | Összehasonlítás .....                                        | 10 |
| 2.6.1 | Hálózati címfordítás (NAT/NAPT).....                         | 10 |
| 2.6.2 | IPv4 címmegosztás .....                                      | 11 |
| 2.7   | Implementációk.....                                          | 12 |
| 2.7.1 | TAYGA .....                                                  | 12 |
| 2.7.2 | Jool.....                                                    | 12 |
| 2.7.3 | ClatD.....                                                   | 12 |
| 2.7.4 | Snabb lwAFTR.....                                            | 12 |
| 2.7.5 | Pyswmap .....                                                | 13 |
| 2.7.6 | OpenWRT.....                                                 | 13 |
| 3     | Teljesítmőképesség-vizsgálat.....                            | 14 |
| 3.1   | Mérési módszertan .....                                      | 14 |
| 3.1.1 | Mérési összeállítás és terminológia .....                    | 15 |
| 3.1.2 | Előzetes Tesztfázis .....                                    | 16 |
| 3.1.3 | A kapcsolatkövető táblázat bejegyzéseinek vezérlése .....    | 17 |
| 3.1.4 | Maximális kapcsolatfelépítési sebesség mérése .....          | 17 |
| 3.1.5 | Valódi tesztfázis .....                                      | 18 |
| 3.2   | Siitperf .....                                               | 18 |
| 3.3   | Mérési környezet .....                                       | 20 |
| 3.4   | Siitperf telepítése a mérőgépre.....                         | 22 |

|        |                                            |    |
|--------|--------------------------------------------|----|
| 3.5    | Jool telepítése és működési módjai .....   | 23 |
| 3.6    | Csomagtovábbítás mérések.....              | 24 |
| 3.7    | 464XLAT konfigurálása .....                | 28 |
| 3.8    | 464XLAT mérése .....                       | 30 |
| 3.9    | MAP-T Tervezése.....                       | 35 |
| 3.10   | MAP-T konfigurálása három szerverrel ..... | 35 |
| 3.11   | MAP-T mérése három szerverrel .....        | 39 |
| 3.12   | MAP-T konfigurálása négy szerverrel .....  | 41 |
| 3.13   | MAP-T mérése négy szerverrel .....         | 43 |
| 3.14   | Eredmények összehasonlítása .....          | 45 |
| 3.14.1 | CPU skálázódás .....                       | 45 |
| 3.14.2 | Kapcsolatszám skálázódás .....             | 47 |
| 3.15   | Összegzés.....                             | 48 |

## Ábrajegyzék

|           |                                                                     |    |
|-----------|---------------------------------------------------------------------|----|
| 1. ábra:  | 464XLAT megvalósítása vezetékes hálózatoknál [3] .....              | 2  |
| 2. ábra:  | 464XLAT megvalósítása mobilhálózatoknál [3] .....                   | 3  |
| 3. ábra:  | MAP-E felépítése [2] .....                                          | 4  |
| 4. ábra:  | MAPIIPv6 címformátum [4] .....                                      | 5  |
| 5. ábra:  | IPv6 Interfész azonosító [4].....                                   | 6  |
| 6. ábra:  | MAP IPv6 cím származtatása [4] .....                                | 6  |
| 7. ábra:  | MAP-T felépítése [2] .....                                          | 7  |
| 8. ábra:  | Dual-Stack Lite felépítése [5] .....                                | 8  |
| 9. ábra:  | Lightweight 4over6 felépítése [2].....                              | 9  |
| 10. ábra: | Dual DUT mérési összeállítás[13] .....                              | 14 |
| 11. ábra: | Mérési összeállítás állapottartó NATxy átjárók méréséhez [15] ..... | 15 |
| 12. ábra: | Dual DUT mérési összeállítás [13] .....                             | 21 |
| 13. ábra: | StarBED sbeasy kezelőfelület .....                                  | 21 |
| 14. ábra: | Mérési összeállítás csomagtovábbítás méréshez 3 szerverrel .....    | 25 |
| 15. ábra: | Mérési összeállítás csomagtovábbítás méréshez 4 szerverrel .....    | 26 |
| 16. ábra: | 464XLAT mérési konfiguráció .....                                   | 28 |
| 17. ábra: | MAP-T konfiguráció .....                                            | 35 |
| 18. ábra: | MAP-T CE konfiguráció (készült [7] alapján) .....                   | 36 |
| 19. ábra: | MAP-T mérési összeállítás 4 szerverrel.....                         | 41 |
| 20. ábra: | CPU skálázódás diagram .....                                        | 45 |
| 21. ábra: | Kapcsolatszám skálázódás diagram .....                              | 47 |

# 1 Bevezetés

A technológia gyorsuló ütemű fejlődése miatt egyre több hálózatba kapcsolható eszköz kerül a piacra, így az áttérés az IPv6-ra kulcsfontosságú kérdéssé válik, mivel az IPv4 IP-címekkel már nehéz gazdálkodni. Ez azt jelenti, hogy az Internet központi adminisztrációjáért felelős szervezet (IANA) központi készlete 2011-ben kimerült [1]. Ennek főbb oka, hogy eredetileg a protokoll által megcímezhető eszközök száma kevésnek bizonyult, ez közelítőleg 4,3 milliárd címet jelentett, de ennek egy része is kihasználatlan maradt (E osztály).

Súlyosbítja a körülményeket az egyre növekvő számban piacra kerülő mobil eszközök (okos telefonok), az egyre növekvő népesség, valamint az eredetileg gyenge hatékonysággal megoldott címtartományok kiosztása.

Az IPv4 címtér kimerülése már régóta probléma, ennek elkerülésére új technológiákat fejlesztettek ki. Ezek közé tartozott a címosztályokat használó rendszer leváltása, azaz az osztály nélküli forgalomirányítás (CIDR – Classless Inter-Domain Routing), valamint a hálózati címfordítás (NAT – Network Address Translation) használata is. Ezekkel mindig csak arrébb tolták az elkerülhetetlent, az IPv6-ra való áttérést.

Azonban mielőtt ezt megtehetnénk, gondolnunk kell az eddig felépített rendszerre is. Ugyanis ezt nem tehetjük meg egyik pillanatról a másikra. Jelen helyzetben már a használatban lévő eszközök száma miatt is egy fokozatos átállás szükséges ennek megvalósításához. Többféle módszer létezik, hogy az IPv4-et, mint szolgáltatást át tudjuk ültetni IPv6-ra.

Az öt legfontosabb IPv4aaS (IPv4-as-a-Service, azaz: IPv4, mint szolgáltatás) megoldás:

1. 464XLAT
2. MAP-E
3. MAP-T
4. Dual Stack Lite
5. Lightweight 4over6 (lw4o6)

Ezek a módszerek beágyazással, címfordítással igyekeznek az IPv4-et mint szolgáltatást fenntartani, ahol szükséges, és segíteni az IPv6-ra való áttérést. Dolgozatomban ezeket a metódusokat fogom összehasonlítani és elemezni.

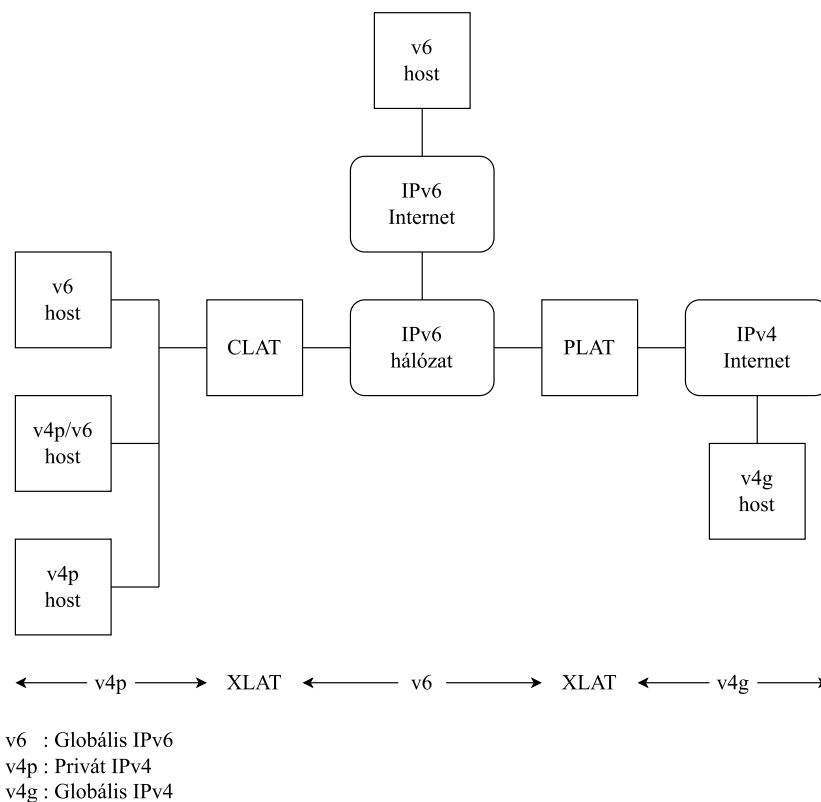
## 2 Technológiák bemutatása

### 2.1 464XLAT

A 464XLAT módszer egy olyan megoldás, ami a csak IPv4-re képes alkalmazásoknak, illetve eszközöknek privát IPv4 címeket ad. Kliens oldalon az IPv4-es csomagokat egy ún. CLAT (Customer-side Translator), ami mobil esetben a felhasználó eszközén található, IPv6-ra fordítja, ezt egy állapotmentes NAT46 fordítással teszi meg. Tehát minden IPv4 címhez eltérő IPv6 cím tartozik. Ehhez *IPv4 címet beágyazó IPv6 címeket* (IPv4-embededd IPv6 address) használ mind a forrás- mind a célcímeknél. [3]

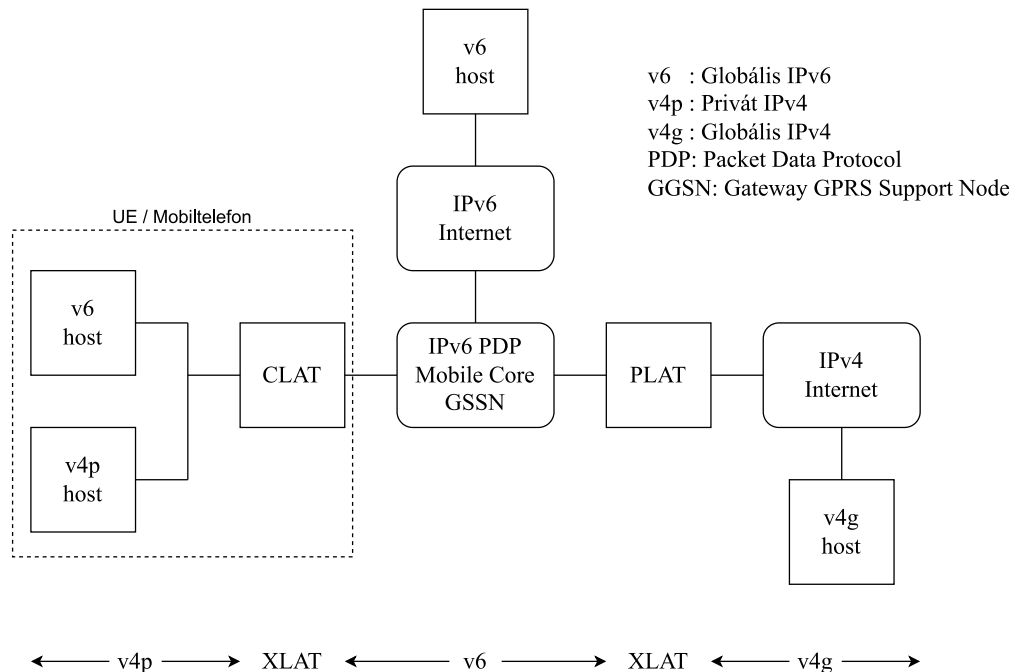
A szolgáltatói hálózatokban van egy ún. PLAT (Provider-side Translator), ami egy állapottartó NAT64. A cél IPv4 címet kicsomagolja az IPv4 címeket beágyazó IPv6 címből, a forráscím pedig a PLAT külső interfészéhez rendelt publikus IPv4 címtartományból származik. [3]

Vezetékes és mobilhálózatoknál eltérhet a CLAT elhelyezése. Előbbinél ezt általában külön eszközként valósítják meg, utóbbinál a CLAT funkciókat szoftveresen oldják meg az adott mobil eszközön. A vezetékes rendszer felépítését az 1. ábra mutatja. [3]



1. ábra: 464XLAT megvalósítása vezetékes hálózatoknál [3]

Látható, hogy kliens oldalon csak IPv4, csak IPv6 és mindkettőre képes eszköz is lehet a hálózaton. Mobilhálózat esetén a felépítés a 2. ábrán látható módon alakul.



2. ábra: 464XLAT megvalósítása mobilhálózatoknál [3]

A 464XLAT együtt használható a DNS64+NAT64 megoldással is, ekkor háromféle forgalmat különböztetünk meg:

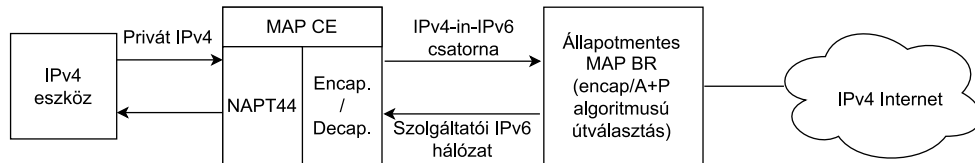
1. Kizárólag IPv4-re képes (vagy azt preferáló) alkalmazás vagy kliens forgalma: 464XLAT használatával halad át a szolgáltatói hálózaton.
  2. IPv6-ra képes kliens és IPv6-ra képes szerver között: natív IPv6 forgalom.
  3. Kizárólag IPv6-ra képes (vagy azt preferáló) kliens vagy alkalmazás forgalma, kizárólag IPv4-re alkalmas szerverrel: NAT64+DNS64 használatával megy át a szolgáltatói hálózaton.
- [3]

Amennyiben a DNS64+NAT64 kimarad, akkor kétféle forgalom van a rendszerben:

1. Kizárólag IPv6-ra képes kliens és IPv6-ra képes szerver között: natív IPv6 forgalom
2. A többi esetben a kliens és a szerver is IPv4-et használ, így kétszeres fordításon halad keresztül a forgalmuk. [3]

## 2.2 MAP-E (Mapping of Address and Port with Encapsulation)

A MAP-E egy állapotmentes algoritmust használ. A felhasználóknak kiosztott IPv4 címeket beágyazza a klienseknek kiadott IPv6 prefixbe, így egyetlen MAP-szabály segítségével temérdek felhasználót lehet kiszolgálni (ezt nevezzük MAP-tartománynak). Ezzel az algoritmussal lehetséges peer-to-peer kapcsolatot biztosítani a közös MAP-szabályon osztozó ügyfeleknek. [2]



3. ábra: MAP-E felépítése [2]

A felhasználói végen (Customer Edge – CE) található útválasztó egy állapottartó NAPT44-et hajt végre, amivel a privát IPv4-es forráscímeket és portokat fordítja a MAP-szabály által megadott tartományba. Ezután történik az IPv4-es csomagok beágyazása IPv6-osokba. A csomagokat elküldi egy hostnak vagy a határ útválasztónak (Border Router – BR), ha a cél IPv4 címet nem definiálták a MAP-szabályban. [2]

A határ útválasztót ellátják egy MAP-szabálykészlettel, ami meghatározza az ügyfél által küldött csomagok kibontásának módját, érvényesíti az IPv4-es címeket és a hozzájuk tartozó 4. rétegbeli portokat, kiszámolja a bejövő IPv4-es forgalom cél IPv6-os címeit. [2]

Kétféle MAP-szabályt különböztetünk meg:

1. Alapvető leképezési szabályok (Basic Mapping Rule – BMR) – kötelező:

Egy CE (Customer Edge) több IPv6 prefixszel is rendelkezhet, viszont egy BMR csak egy prefixhez tartozhat. A végfelhasználói IPv6 prefixszel együtt arra használják, hogy az IPv4 prefixeket, címeket vagy megosztott címeket, a CE-hez rendelt port készlet azonosítót származtassák (PSID). [4]

2. Továbbító leképezési szabályok (Forwarding Mapping Rule – FMR) – opcionális:

Továbbításra (szállításra) használják. A BMR is lehet FMR. Minden FMR egy bejegyzést eredményez egy táblázatban a szabály IPv4 prefixhez. Ez megad egy cél IPv4 címet és egy portot a MAP-tartományban. Egy eszköz az illeszkedő FMR-t arra használja, hogy ebből származtassa a végfelhasználói IPv4 címet annak az interfésznek, amin keresztül az adott cél IPv4 cím és port kombináció elérhető. [4]

Mindkét típusú MAP-szabály ugyanazokkal a paraméterekkel rendelkezik:

- Szabály IPv6 prefix
- Szabály IPv4 prefix
- Szabály beágyazó bitek hossza

A MAP-eszköz általában a leghosszabban illeszkedő alapvető leképezési szabályt keresi a végfelhasználói és a szabály IPv6 prefixek között. [4]

### 2.2.1 Alapvető leképezési szabályok (BMR)

Kötelező, tehát minden IPv6 prefixnek rendelkeznie kell vele. A CE használja, hogy IPv4 prefixszel, osztott címmel, vagy akár teljes címmel tudja ellátni magát. Az alábbi ábra mutatja egy MAP IPv6 cím felépítését. [4]

| n bit               | o bit    | s bit                | 128-n-o-s bit       |
|---------------------|----------|----------------------|---------------------|
| Szabály IPv6 prefix | EA bitek | Alhálózati azonosító | Interfész azonosító |

← Végfelhasználói IPv6 prefix →

4. ábra: MAPIPv6 címformátum [4]

A *szabály IPv6 előtag* (prefix) közös azok között a CE-k között, amelyek ugyanazt a BMR-t használják a MAP-tartományon belül. Az *EA bitek* mező kódolja el a CE specifikus IPv4 cím és port információkat. Ezek azok a bitek, amely egyediek minden egyes adott IPv6 szabály prefixnél. Tartalmazhatnak teljes vagy részleges IPv4 címet, és osztott címek esetén egy PSID-t. A PSID (port set ID, azaz port készlet azonosító) egy algoritmus alapján határozza meg a CE-hez rendelt portok halmazát. [4]

A MAP IPv6 címet úgy hozzák létre, hogy összekapcsolják a végfelhasználói IPv6 előtagot az alhálózat azonosítóval (ha a végfelhasználói IPv6 prefix rövidebb, mint 64 bit) és az interfész azonosítóval. [4]

Az IPv6 *Interfész azonosító* a következőképpen néz ki.

|               |          |        |
|---------------|----------|--------|
| 128-n-o-s bit |          |        |
| 16 bit        | 32 bit   | 16 bit |
| 0             | IPv4 cím | PSID   |

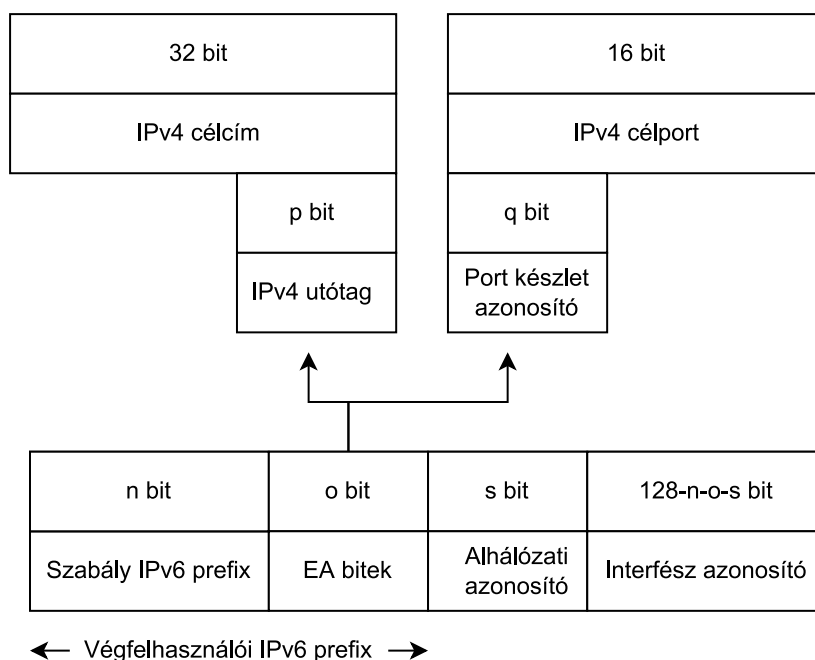
5. ábra: IPv6 Interfész azonosító [4]

IPv4 előtag esetén az IPv4 cím mezőt jobbról töltik fel nullákkal, maximum 32 bit hosszan. A PSID mezőt pedig balról töltik fel nullákkal, így egy 16 bites mező keletkezik. IPv4 prefix, vagy egy teljes IPv4 cím esetén a PSID mező értéke 0. [4]

Ha a végfelhasználói IPv6 előtag hossza nagyobb, mint 64, az interfész azonosító jelentős része felülíródik a prefix tartalmával. [4]

## 2.2.2 Továbbító leképezési szabályok (FMR)

A továbbító leképezési szabály opcionális. Mesh módban alkalmazzák, hogy közvetlen CE-k közötti kapcsolatot tudjanak létesíteni. FMR-szabály hozzáadásakor egy IPv4 utat hoznak létre a szabály IPv4 prefix táblájában. [4]

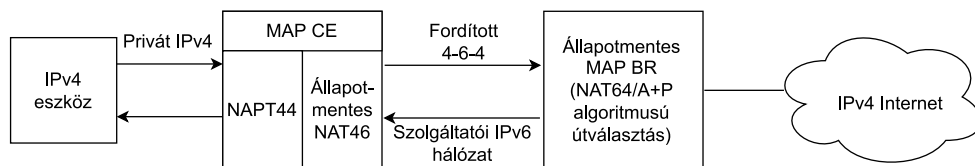


6. ábra: MAP IPv6 cím származtatása [4]

## 2.3 MAP-T (Mapping of Address and Port using Translation)

A MAP-T ugyanazt az algoritmust használja, mint a MAP-E. A lényeges különbség a kétszeres fordításban van az IPv6-os hálózatában (NAT46 a felhasználói oldalon, NAT64 a határ útválasztónál). Emiatt hasonlít a 464XLAT-ra is. [2]

A MAP-T a felhasználói végpontokon (CE) állapotmentes NAT44-et használ a forgalom nyilvános IPv4 címre való fordítására, a port tartományt az IPv6 prefixhez tartozó BMR alapján számolja ki. A határ útválasztót is ugyanaz a szabálykészlet látja el, mint a klienst, így lehetővé téve a beérkező IPv6 forgalom számára, az állapotmentes visszafordítást arra a nyilvános IPv4 címre, amelyet az ügyfél használ. [2]

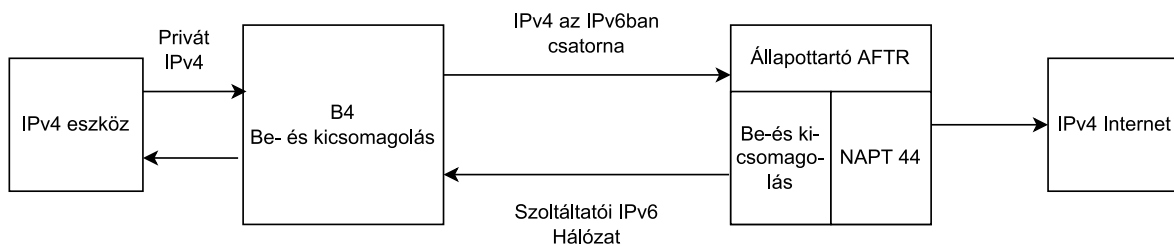


7. ábra: MAP-T felépítése [2]

A beágyazás helyetti fordítás használata lehetővé teszi, hogy a csak IPv4 címmel rendelkező eszközök összhangban legyenek azokkal az IPv6 készülékekkel a MAP-T tartományon belül, amelyek IPv4 címeket beágyazó IPv6 címekkel rendelkeznek. [2]

## 2.4 Dual-Stack Lite (DS Lite)

A Dual-Stack Lite a dolgozatomban tárgyalt technológiák közül kronológiai sorrendben az első áttérési módszer, amit kifejlesztettek. A DS-Lite egy „Basic Broadband Bridging” (B4) funkciót alkalmaz a felhasználói (CE) routernél, amely beágyazza az IPv4-es forgalmat IPv6-osba, és átküldi azt az IPv6-os natív szolgáltatói hálózaton keresztül egy központi *címcsaládfordító routerre* (Address Family Transition Router – AFTR). Az AFTR elvégzi a forgalom kibontását, majd átfordítja az IPv4 csomagot a publikus IPv4 forráscímre állapotartó NAPT44 segítségével. [2]



8. ábra: Dual-Stack Lite felépítése [5]

### 2.4.1 B4 alkotóelem

A B4 egység egy funkció, amit a dual-stack képes eszközökbe implementáltak, vagy egy közvetlenül csatlakoztatott készülék, amely nyit egy csatornát egy AFTR-hez. Ez egy többpont-pont IPv4-in-IPv6 csatorna, amely a szolgáltató AFTR eszközénél végződik.

Mivel beágyazást használ az IPv4 forgalom IPv6 feletti szállításához, így csökken a datagram hasznos MTU-ja. Sajnos az útvonal MTU felderítés nem egy megbízható módszer a probléma orvoslására, ezért ezt úgy oldják meg a szolgáltatói oldalon, hogy megnövelik az MTU méretet az összes B4 és AFTR közötti kapcsolatnál minimum 40 bájtal, hogy hozzá tudják illeszteni a beágyazó IPv6 fejrészt beágyazott az IPv4 datagramhoz anélkül, hogy tördelni kelljen az IPv6 csomagot. Azonban nem mindegyik szolgáltató teszi ezt meg, így a B4-nek tördelés és összeillesztés végrehajtására is képesnek kell lennie. [5]

### 2.4.2 AFTR alkotóelem

Az AFTR-egység egy IPv4-in-IPv6 csatorna végpont és egy NAT44 kombinációja egy eszközbe implementálva. Másképpen kifejezve, ez egy pont-multipont IPv4-in-IPv6 csatorna, amely a B4 egységekben végződik. [5]

Ahogy már azt korábban említettem, a csomag szétdarabolására és újra összeillesztésére ügyelni kell a csatorna végpontjain. Így az AFTR-nek is végre kell hajtania ezt a műveletet, ha az összeköttetés

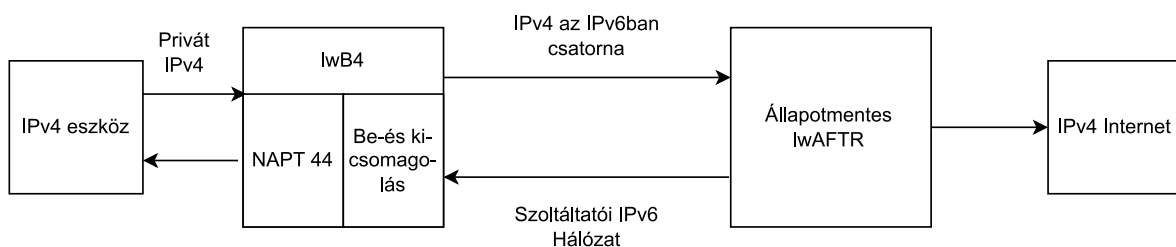
MTU-ja nem éri el a beágyazás miatt megnőtt csomagok méretét. A csomag tördelésének a beágyazás után kell történnie, az újra összeállításának pedig a kibontás előtt. [5]

## 2.5 Lightweight 4over6

A Lightweight 4over6 a Dual-Stack Lite változata. A fő különbség abban rejlik, hogy az állapottartó NAT44 funkciót áthelyezték a központi AFTR-ből a felhasználó B4 részéhez (lwB4), így az AFTR (lwAFTR) itt csak a cím és port szám alapú forgalomirányítást, és a forgalom kibontását és beágyazását végzi. [2]

Ahhoz, hogy a megfelelő klienshez juttassák a csomagokat és megoldják az IPv4 címmegosztást, cím és portszám alapú útválasztási modellt használnak, így minden lwB4 részt egyedi IPv4 címmel és port-tartománnyal tudnak ellátni. Így az lw4o6 egyben a MAP-E speciális esete is. [2]

Az lwAFTR-be implementáltak egy összekötési táblát, amely kliensenként bejegyzést tartalmaz a felhasználó forrás IPv4 címéről és a kiosztott 4. rétegbeli port tartományról az IPv6 csatorna végponti címéhez. Ez lehetőséget ad a kimenő forgalom ellenőrzésére, valamint a bejövő csomagok megfelelő beágyazására és továbbítására. [2]



9. ábra: Lightweight 4over6 felépítése [2]

## 2.6 Összehasonlítás

Adatsíkon kétféle megközelítést különböztetünk meg a szolgáltató IPv6-os hálózatán való áthaladáskor:

- 4-6-4 címfordítás
- 4-a-6-ban beágyazás [2]

|                   | 4-6-4 XLAT | MAP-T | MAP-E | DS-Lite | Lw4o6 |
|-------------------|------------|-------|-------|---------|-------|
| 4-6-4 címfordítás | X          | X     |       |         |       |
| 4-6-4 beágyazás   |            |       | X     | X       | X     |

1. táblázat: Elérhető áthaladási módszerek a szolgáltató hálózatán [2]

Dolgozatomban taglalt megoldásokban, minden beágyazás alapú módszer egy IP az IP-ben csatornát használ. Ez egy állapotmentes eljárás, amelyhez nincs szükség további csatorna fejrész hozzáadására. [2]

Mind a két megközelítés a szolgáltató hálózatán áthaladó csomagok méretbeli növekedését eredményezi. A 4-6-4 kétszeres fordítás (464XLAT és MAP-T) 20 bájtnyi többletet ad (a 20 bájtos IPv4 fejrész helyére egy 40 bájtos IPv6 fejrész kerül). A beágyazás 40 bájtnyi többletet eredményez (egy IPv6 fejrész kerül az IPv4 fejrész elé). [2]

Ez a méretbeli növekedés jelentős probléma lehet, ha a hálózatban található olyan összeköttetés, amely kisebb MTU-val rendelkezik. Ennek az az eredménye, hogy a csomagot szét kell törölni az IPv6 tartomány bemenetén. Továbbá eredményezheti azt is, hogy implementálni kell a puffrelést és a töredék újraegyesítését a felhasználó oldali és a szolgáltató oldali eszközökben. [2]

### 2.6.1 Hálózati címfordítás (NAT/NAPT)

A szolgáltató az IPv6-ra való áttérés megoldása érdekében a MAP-T kettős állapotmentes címfordítást használ. Először a CE-nél IPv4-ről IPv6-ra (NAT46), majd IPv6-ról IPv4-re a szolgáltatást ellátó hálózatánál. [2]

A 464XLAT használhat kettős címfordítást (állapotmentes NAT46-ot és egy állapottartó NAT64-et), vagy egyszeres fordítást (állapottartó NAT64-et), attól függően, milyen forgalom halad át a hálózaton. [2]

A kettős címfordítás egyik lépése egy állapotmentes NAT IPv4-ről IPv6-ra, amely nem fordítja át az IPv4 fejléc „option” mezőjét és/vagy a multicast IP/ICMP csomagokat. Ez a módszer csak port számokat használó IP protokollok szállítására alkalmas (pl. TCP, UDP), illetve ICMP esetén is működnek valamilyen egyedi azonosító jellegű mező használatával. A beágyazás alapú eljárások ezzel szemben az IPv4 fejlécszt érintetlenül szállítják, így multicast csomagok is átvihetők, és minden IP feletti protokollt támogatnak. [2]

## 2.6.2 IPv4 címmegosztás

A publikus IPv4 címek elfogyása miatt az együttélési technológiáknak meg kell oldaniuk a megosztott IPv4 címtartomány támogatását is. Ahhoz, hogy ennek a követelménynek eleget tudjanak tenni, elengedhetetlen eszköz az állapottartó NAPT funkció. A legfőbb különbség az, hogy ez hol található az architektúrában. [2]

A dolgozatomban tárgyalt megoldások két kategóriába sorolhatóak:

- Központosított állapottartó NAPT (DS-Lite, 464XLAT)
- Szétszított állapottartó NAPT (lwg4o6, MAP-E, MAP-T) [2]

A központosított modellben egy eszköz olyan, mint egy CGN/AFTR vagy PLAT (állapottartó NAT64) látja el a NAPT funkciókat, és kapcsolatonként kezel állapotokat minden aktív ügyfél forgalmánál. [2]

Az elosztott modellben általában az ügyfél eszköze felel az állapottartó NAPT funkciókért és kezeli az összetartozó eszközök forgalmának állapotát kapcsolatonként. Itt a központi berendezések csak állapotmentes be- és kicsomagolást hajtanak végre. [2]

Az állapottartó technológiák, mint a 464XLAT és a DS-Lite hatékonyabbak lehetnek port kiosztás, így publikus IP cím megtakarítás szempontjából is. Ennek az ára az állapottartó működés a szolgáltatói hálózaton, ami nem skálázódik jól (későbbi méréseimben ez látható). Ez függhet attól, hogyan valósítják meg az adott áttérési módszert. [2]

## 2.7 Implementációk

Az alábbi táblázatban látható implementációkat gyűjtöttem össze a munkámhoz.

| Név          | URL                                                                                                 |
|--------------|-----------------------------------------------------------------------------------------------------|
| TAYGA        | <a href="http://www.litech.org/tayga/">http://www.litech.org/tayga/</a>                             |
| Jool         | <a href="https://www.jool.mx">https://www.jool.mx</a>                                               |
| ClatD        | <a href="https://github.com/toreanderson/clatd">https://github.com/toreanderson/clatd</a>           |
| Snabb lwAFTR | <a href="https://github.com/Igalia/snabb">https://github.com/Igalia/snabb</a>                       |
| Pyswmap      | <a href="https://github.com/ejordangottlieb/pyswmap">https://github.com/ejordangottlieb/pyswmap</a> |
| OpenWRT      | <a href="https://openwrt.org/">https://openwrt.org/</a>                                             |

2. táblázat: Elérhető implementációk

### 2.7.1 TAYGA

A TAYGA egy kernelen kívüli állapotmentes NAT64 implementáció Linuxhoz. A TUN illesztőprogramot használja az IPv4 és IPv6 csomagok cseréjére a kernellel. Célja, hogy egy minőségi NAT64 szolgáltatást nyújtson azoknak a hálózatoknak, ahol a dedikált hardver túlzás lenne. [6]

### 2.7.2 Jool

A Jool egy nyílt forráskódú implementáció IPv4/IPv6 címfordításhoz Linuxra. A 3.2.x verzióig csak állapottartó NAT64-ként működött, a 3.3.0 verziótól már támogatja állapotmentes a NAT64-et is. A 4.2.0 verziótól kezdve pedig implementálja a MAP-T protokollt is. [7]

### 2.7.3 ClatD

A ClatD a 464XLAT áttérési technológiának a CLAT komponensét implementálja, az RFC 6877-ben leírtak szerint. A csomagok IPv4 és IPv6 közötti fordításához Nathan Lutchansky TAYGA szoftvercsomagjára támaszkodik. [8]

### 2.7.4 Snabb lwAFTR

A Snabb lwAFTR a Lightweight 4over6 áttérési technológia internet felőli részét implementálja. [9]

### 2.7.5 Pyswmap

A Pyswmap egy MAP-E, MAP-T Pythonban írt modul. [10]

### 2.7.6 OpenWRT

Az OpenWRT projekt egy Linux operációs rendszer, amely beágyazott eszközöket céloz meg. Ahelyett, hogy egyetlen statikus firmware-t készítenének, egy teljesen írható fájlrendszert biztosít csomagkezeléssel. Ezek között található 464XLAT, MAP-E, MAP-T, DS-Lite és Lw4o6 kliens oldali támogatás. [11]

### 3 Teljesítőkéesség-vizsgálat

Dolgozatomban két áttérési technológia teljesítményét mérem le, ezek a 464XLAT és a MAP-T. Azért erre a kettőre esett a választásom, mert mindegyik címfordítós módszerrel dolgozik, és megvalósítható a NIC Mexico, ITESM által fejlesztett Jool [7] nyílt forráskódú IPv4/IPv6 fordítóval. Így a választásom jó lehetőséget ad a teljesítményük összehasonlítására. További összehasonlítási alapként szolgáló referencia létrehozása érdekében lemérem a Linux kernel által végzett IPv4 és IPv6 csomagtovábbítás teljesítményét is.

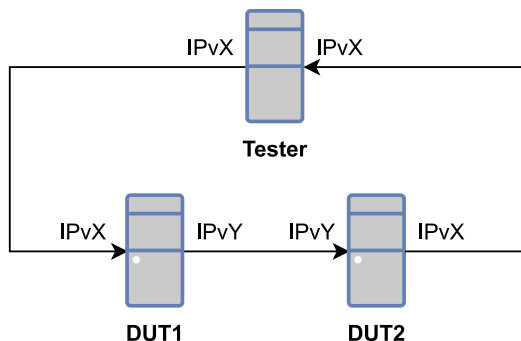
#### 3.1 Mérési módszertan

Az RFC 8219 [13] elvileg mindkét technológiához definiálja a mérési módszertant, ami a kétszeres fordítást használó technológiák teljesítményének a mérésére kétféle lehetőséget ad:

1. Egy vizsgált eszközt tartalmazó összeállítás.
2. Két vizsgált eszközt tartalmazó összeállítás.

Az első esetben a 464XLAT technológia CLAT és a PLAT eszközeinek, valamint a MAP-T technológia CE és BR eszközeinek teljesítményvizsgálata külön-külön történik, és a vizsgálat elvégzéséhez speciális mérőszoftver szükséges, amellyel a MAP-T esetén nem rendelkezünk.

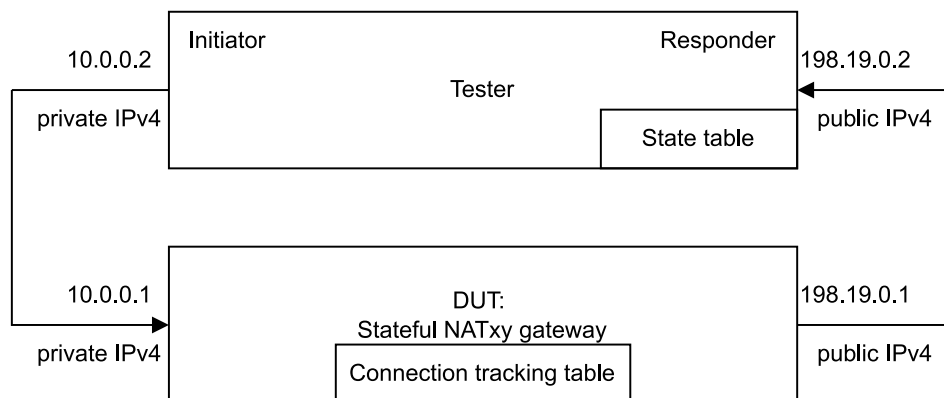
A második esetben a 10. ábraán látható módon a CLAT és PLAT, illetve a CE és BR eszközök teljesítményvizsgálata együttesen történik, és így elegendő, ha a mérőeszköz mindkét portján IPv4-et használ. Bár ez nagy könnyebbséget jelent, azonban még így is van módszertani probléma, ugyanis a két eszközt egy rendszernek tekintve is egy állapotartó NAT44 fordítást végrehajtó rendszerrel állunk szemben. Erre a problémára nyújt megoldást a jelenleg fejlesztés alatt álló Internet Draft [15]. Az alábbiakban ennek alapján mutatom be a mérési módszertan legfontosabb részleteit.



10. ábra: Dual DUT mérési összeállítás [13]

### 3.1.1 Mérési összeállítás és terminológia

A mérési módszer kellően általános, bármelyik oldalon bármelyik IP verzióval működőképes, ezért szerepel a nevében NATxy. Jelen esetben természetesen elegendő a NAT44-re fókuszálni. A 11. ábra a mérési összeállítást mutatja, amelyen IPv4-es címek láthatók, a jól ismert NAT44 alapján. Ami a szállítási rétegbeli protokollt illeti, az RFC2544 az UDP-val való mérést ajánlja, ezt az RFC8219-ben is megtartották. Általános ajánlásként itt is az UDP-t alkalmazzuk. [15]



11. ábra: Mérési összeállítás állapotartó NATxy átjárók méréséhez [15]

A következő elemek találhatók a mérőrendszerben:

*Kapcsolatkövető tábla* (connection tracking table): Az állapotartó NATxy átjáró egy kapcsolatkövető táblát használ a publikusból a privát irányba való fordításhoz. A mérete, a szervezése és a tartalma ismeretlen a Tester számára. [15]

*Számnégyes* (four tuple): A négy szám, amely azonosítja a kapcsolatot a forrás IP cím, forrás port szám, cél IP cím, cél port szám. [15]

*Állapot tábla* (state table): A Tester Válaszadó portja eltárolja a vett teszt keretekben található számnégyeseket az állapotáblájába.

*Kezdeményező* (Initiator): A Tester azon portja, amely kapcsolatot kezdeményezhet az állapotartó DUT-on keresztül a privátból a publikus irányba. Elméletileg minden forrás és cél port számot használhat az RFC4814 által ajánlott tartományból. Ha a használt számnégyes nem tartozik egy létező kapcsolathoz, azt a DUT új kapcsolatként regisztrálja a kapcsolatkövető táblájába. [15]

*Válaszadó* (Responder): A Tester azon portja, amely nem kezdeményezhet kapcsolatot a DUT-on keresztül a privátból a publikus irányba. Csak meglévő kapcsolatokon keresztül küldhet kereteket. Ezen célból csak olyan szám négyeseket használ, amelyeket előzőleg kinyert a megérkezett teszt keretből, és eltárolt az állapot táblájában. [15]

*Előzetes tesztfázis:* Teszt kereteket küld a kezdeményező a válaszadónak a DUT-on keresztül, hogy a kapcsolatkövető és az állapot táblát feltöltse számnégyesekkel. [15]

*Valódi tesztfázis:* A tényleges teszt (például áteresztőképesség, válaszidő stb.) végrehajtása az előzetes fázis befejezése után. [15]

### 3.1.2 Előzetes Tesztfázis

Az előzetes tesztfázis két célt szolgál:

1. A DUT kapcsolatkövető táblájának a feltöltése. Ez azért fontos, mivel a maximális kapcsolat felépítési sebesség kisebb lehet, mint a legnagyobb keret továbbítási sebesség (ez az áteresztőképesség, throughput). [15]
2. A Tester válaszadó portja állapottáblájának érvényes számnégyesekkel való feltöltése. Ez előfeltétel ahhoz, hogy a válaszadó olyan kapcsolatokhoz tartozó kereteket tudjon továbbítani, amely már létezik a DUT kapcsolatkövető táblájában. [15]

Az előbb említett két cél elérése mindig szükséges a valódi tesztfázis előtt, de az előzetes tesztfázis alkalmazható valódi tesztfázis nélkül. Ekkor a maximális kapcsolatépítési sebesség (maximum connection establishment rate) mérése történik. [15]

Az előzetes tesztfázist minden, a valódi tesztfázisban történő mérés előtt el kell végezni. Ilyenkor a következő zajlik le:

1. A kezdeményező teszt kereteket küld a válaszadónak a DUT-on keresztül, meghatározott sebességgel.
2. A DUT elvégzi a teszt keretek állapottartó fordítását, és az új számnégyeseket eltárolja a kapcsolatkövető táblájában.
3. A válaszadó fogadja a fordított kereteket, és frissíti az állapottábláját az érkezett számnégyesekkel. Az előzetes tesztfázisban a válaszadó nem továbbít teszt kereteket. [15]

Amikor az előzetes tesztfázist egy valódi mérési fázisban történő mérés előkészítéseként hajtjuk végre, a mérés időtartamát és a keretsebességet úgy kell megválasztani, hogy:

- A használt keretsebesség kissé alacsonyabb legyen a maximális kapcsolat felépítési sebességénél.
- Minden, a valódi tesztfázisban használható portszám kombináció használatra kerüljön annak érdekében, hogy a valódi tesztfázisban már ne történjen kapcsolatfelépítés. [15]

### 3.1.3 A kapcsolatkövető táblázat bejegyzéseinek vezérlése

Ahhoz, hogy tiszta feltételeink legyenek a méréseinkhez, szükséges a kapcsolatkövető tábla bejegyzéseit ellenőrzés alatt tartani. Egyszerűen elérhetjük a következő két szélső esetet:

1. Minden keret egy új bejegyzést hoz létre a DUT kapcsolatkövető táblájában a régiek törlése nélkül. Ez szükséges a maximum kapcsolat felépítési sebesség méréséhez.
2. Nincsenek új bejegyzések a DUT kapcsolatkövető táblájában, és a már meglévőket sem töröljük a teszt alatt. Ez az ideális a valódi tesztfázis méréseinek, mint például az áteresztőképesség és késleltetés (latency) tesztek. [15]

Innentől kezdve a következő három feltételezést használjuk:

1. Egyetlen forrás és cél címpárt használtunk a tesztek során.
2. A NATxy átjáró kapcsolatkövető táblája elég nagy ahhoz, hogy az összes kapcsolatot eltárolja a különböző forrás- és célport szám kombinációkkal együtt.
3. Minden kísérlet üres kapcsolatkövető táblával indul.

Az első szélső helyzet elérhető úgy, hogy az előzetes fázisban minden teszt keretnél különböző forrás- és célport szám kombinációkat használunk, és az UDP időtúllépést (timeout) a NATxy átjáróban olyan hosszúra állítjuk, hogy nagyobb legyen, mint az előzetes tesztfázis végrehajtásának ideje. [15]

A második szélső helyzet elérhető úgy, hogy az előzetes fázisban minden lehetséges forrás- és célport szám kombinációt felhasználunk, és az UDP időtúllépést a NATxy átjáróban olyan hosszúra állítjuk, hogy nagyobb legyen, mint az előzetes és valós tesztfázis futásideje plusz a két fázis között eltelt idő. [15]

Az RFC4814 megköveteli az álvéletlen port számok használatát, ami egy eléggé jó közelítése a forrásportok eloszlásának, amivel a NATxy átjáró az interneten találkozhat. [15]

### 3.1.4 Maximális kapcsolatfelépítési sebesség mérése

A maximális kapcsolatfelépítési sebesség (maximum connection establishment rate) az állapottartó NATxy átjáróknak egy fontos jellemzője, és meghatározása szükséges az előzetes tesztfázis biztonságos futtatásához (keretvesztés nélkül) a valódi tesztfázis előtt. [15]

A mérés során a kezdeményező meghatározott számú teszt keretet küld különböző forrás- és célport kombinációk használatával egy meghatározott sebességgel a DUT-on keresztül. A válaszadó megszámlálja a DUT által sikeresen fordított kereteket. Ha a kapott keretek száma megegyezik a

küldöttek számával, akkor a sebesség emelésre kerül, és a tesztet újra futtatjuk. Ha kevesebb keret érkezik meg, mint amennyi küldésre került, a sebességet csökkentjük, és újra elvégezzük a tesztet. [15]

A maximális kapcsolatépítési sebesség a legnagyobb érték, amelyen még sikeresen fordított a DUT, tehát a küldött és a megérkezett csomagok száma egyenlő. A gyakorlatban ennek a meghatározására bináris keresést használunk. [15]

### 3.1.5 Valódi tesztfázis

A forgalom iránya szerint háromféle lehetőséget különböztetünk meg a valódi tesztfázisban:

- Kétirányú forgalom: A kezdeményező teszt kereteket küld a válaszadónak és fordítva.
- Egyirányú forgalom a kezdeményezőtől a válaszadónak: A kezdeményező teszt kereteket küld a válaszadónak, de a válaszadó nem küld teszt kereteket visszafelé.
- Egyirányú forgalom a válaszadótól a kezdeményezőnek: A válaszadó teszt kereteket küld a kezdeményezőnek, de a kezdeményező nem küld teszt kereteket visszafelé.

Ha a kezdeményező küld teszt kereteket, akkor álvéletlen forrás- és célport számokat használ egy korlátozott tartományból. A válaszadóhoz megérkeznek a keretek, frissíti az állapottábláját, és feldolgozza a teszt kereteket az adott mérési procedúra alapján (például megszámlálja őket az áteresztőképesség vizsgálathoz, kezeli az időbélyegeket a válaszidő tesztnél). [15]

Ha a válaszadó küld teszt kereteket, akkor a számnégyeseket használja az állapottáblájából. A kezdeményezőhöz megérkeznek a csomagok, és feldolgozza a megfelelő mérési eljárás szerint.

## 3.2 Siitperf

A Siitperf eredetileg egy RFC 8219 kompatibilis SIIT (állapotmentes NAT64) teszter, C++ nyelven íródott DPDK (Intel Data Plane Development Kit) segítségével. Linux operációs rendszer alatt használható. Az RFC 8219 legfontosabb mérési eljárásait implementálja, nevezetesen az áteresztőképesség (throughput), a keretvesztés arány (frame loss rate), a késleltetés (latency) és a csomagkésleltetés változás (PDV – Packet Variation Delay) méréseket. [12]

A program kutatási célokra készült, elég rugalmas és hangolható, nincsenek állandó értékek beleírva. A paramétereket a `siitperf.conf` konfigurációs fájlból vagy a parancssorból veszi. A csomag három hasonló, de mégis különböző programból áll: `siitperf-tp`, `siitperf-lat`, `siitperf-`

pdv. A `siitperf-tp` szolgál az áteresztőképesség és keretveszteség arány, a `siitperf-lat` a késleltetés mérésekre, a `siitperf-pdv` pedig a csomagkésleltetés változás plusz speciális áteresztőképesség és keretveszteség arány mérésekre, amelyek minden egyes keret időtúllépését ellenőrzik. Mindhárom program egy elemi tesztet hajt végre (60 másodpercig, vagy amit beállítunk), ha többszöri futtatásra van szükség, például az áteresztőképesség mérésénél bináris kereséséhez, akkor a programokat többször is végrehajtják a `bash` shell szkriptek. [12]

Mindegyik szoftver ugyanazt a `siitperf.conf` konfigurációs fájlt használja, egy minta konfigurációs fájl megtalálható az függelékben. A parancssoros paramétereik is nagyon hasonlóak:

```
./build/siitperf-tp <IPv6 méret> <sebesség> <időtartam> <globális  
időtúllépés> <n> <m>  
./build/siitperf-lat <IPv6 méret> <sebesség> <időtartam> <globális  
időtúllépés> <n> <m> <késleltetés> <időbélyegek>  
./build/siitperf-pdv <IPv6 méret> <sebesség> <időtartam> <globális  
időtúllépés> <n> <m> <keret időtúllépés>
```

*IPv6 méret* alatt az IPv6 keret méretét értjük bájtokban (84-1518), az IPv4 keretek automatikusan 20 bájtal rövidebbek.

A *sebesség* a keretek küldési gyakoriságát adja meg keret/másodpercben.

Az *időtartam* alatt a tesztelés időtartamát értjük másodpercben (1-3600).

A *globális időtúllépés* (milliszekundumban) a Tester befejezi a keretek fogadását, miután ez az időtartam letelik.

*n* és *m* két relatív prímszám, ami meghatározza az elő- és háttér forgalom arányát.

*Késleltetés* meghatározza azt az időeltolást, mielőtt az első időbélyeges keretet kiküldi (másodpercekben, 0-3600).

*Időbélyegek* paraméter megadja az időbélyeges keretek számát (1-50 000)

*Keret időtúllépés* (milliszekundumban): Ha az értéke 0, akkor PDV mérést végez. Ha nagyobb, mint 0, akkor pedig speciális áteresztőképesség (vagy keretveszteség arány) méréseket végez, ahol a Tester ellenőrzi ezt az időtúllépést minden keretnél egyesével. [12]

A programnak létezik stateful [16] verziója is, amellyel állapottartó NAT44 és NAT64 megoldásokat is képes mérni, ehhez további parancssoros paraméterek tartoznak:

```
./build/siitperf-tp <IPv6 méret> <sebesség> <időtartam> <globális  
időtúllépés> <n> <m> <N> <M> <R> <T> <D>
```

N: A teszt keretek száma az előzetes fázisban ( $1 - 2^{32} - 1$ )

M: A Tester állapot táblájában lévő bejegyzéseinek száma ( $1 - 2^{32} - 1$ )

R: A keretsebesség, amin a teszt kereteket továbbítja az előzetes fázisban (keret/másodperc)

T: Globális időtúllépés az előzetes fázisban (ezredmásodpercben 1-2000)

D: A teljes késleltetés az előzetes fázis miatt (ezredmásodpercben, 1-100 000 000)

A mérések végrehajtását különböző mérő szkriptek segítik, ezek mindegyikét testre kell szabni az adott méréshez, a letölthető fájlok inkább mintának használhatók. [12]

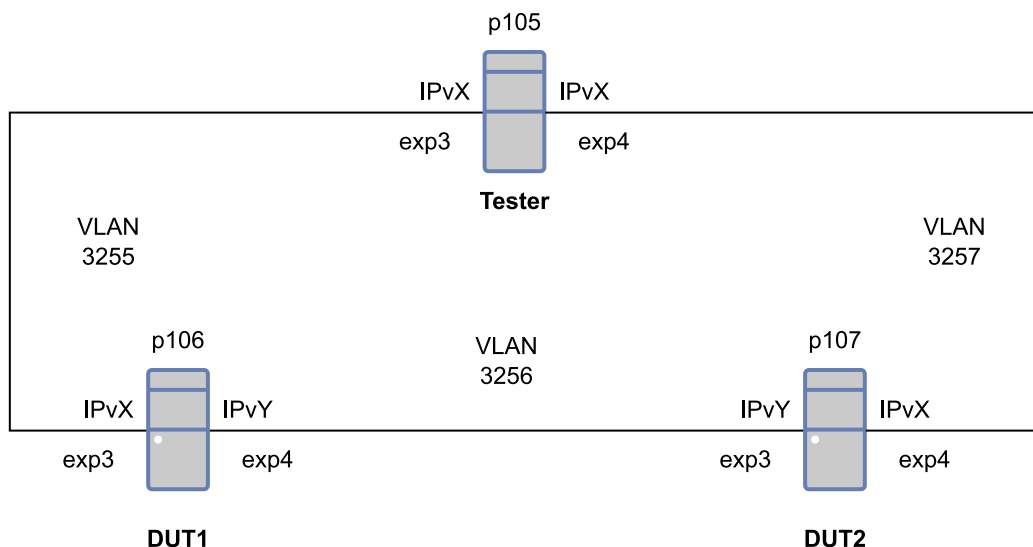
### 3.3 Mérési környezet

A méréseket a Japánban elhelyezkedő NICT Hokuriku StarBED Technology Centerben megtalálható szervereken végeztem. Munkámhoz összesen 7 darab Dell PowerEdge R430 típusú szervert használtam, amelyek egyenként 2 darab Intel Xeon CPU E5-2683 v4 2,1 GHz 16 magos processzorral rendelkeztek így gépenként 32 processzormag állt rendelkezésre. A szerverek egyenként 12 db 32GB-os DDR4 24MHz-es RAM modulokkal voltak felszerelve, így gépenként 384 GB memória állt rendelkezésemre. Mindegyik szerverben volt egy Intel 10G 2P X540-t Adapter, amelyen 2 db 10 Gigabites port található, ezekkel végeztem méréseimet. Az alaplapon még 4 db Broadcom Gigabit Ethernet BCM5720 típusú Gigabites port helyezkedett el. Az összes interfész egy VLAN switch-csel van összekötve, amely lehetővé teszi, hogy távolról egy kezelőfelület segítségével hozzáadja vagy eltávolítsa az egyes portokat az adott VLAN-ból, így nem kell fizikailag átkábelezni a szervereket ahhoz, hogy össze lehessen kötni a mérési összeállításhoz. A mérésekhez a gépek BIOS-ában a Turbo Boost és a Hyper-Threading funkció kikapcsolásra került, hogy stabilabb eredmények legyenek. A mérési eredmények szórásának csökkentése érdekében minden szerveren a CPU frekvencia stabil 2100MHz-re került beállításra. Ezt a CPU frekvenciaskálázás „performance” értékre állításával, az `idle=poll` kernel paraméter és a `tlp` csomag segítségével sikerült elérni. A `/etc/default/tlp` fájlban a következő sorokat használtam:

- `TLP_ENABLE=1`
- `CPU_SCALING_GOVERNOR_ON_AC=performance`
- `CPU_SCALING_MIN_FREQ_ON_AC=2100000`
- `CPU_SCALING_MAX_FREQ_ON_AC=2100000`

Két szerveren Debian GNU/Linux 10 (buster) operációs rendszer van, 4.19.0-18-amd64 verziójú kernellel. Ezek a mérendő eszközök, angolul DUT (Device Under Test). Egy gépen Debian GNU/Linux 9 (stretch) operációs rendszer fut, 4.9.0-14-amd64 verziójú kernellel. Ez a mérőgép (Tester). A Linux disztribúciókat és verziókat azért így választottam meg, mert a készítő(k) Debian 9-en tesztelték a mérőprogramot a `siitperf`-et [12], Debian 10-en pedig az implementációt, azaz a

Joolt [7]. Méréseimhez a p105, p106, p107-es nevű szervereket kaptam meg. A p105-ös a mérőeszköz, a p106 és a p107 pedig a két DUT. Mivel a mért áttérési technológiák kialakítása páros, tehát az egyik IPv4 – IPv6 fordítást a másik meg épp ellenkezőleg IPv6 - IPv4 címfordítást hajt végre, ezért az RFC 8219 szerinti Dual DUT összeállításban mértem őket a 12. ábra szerint.



12. ábra: Dual DUT mérési összeállítás [13]

A szerverek elérése FortiClient [14] segítségével VPN-en keresztül SSH-val történik. Közvetlenül az eszközökre nem lehet SSH-zni, hanem egy úgynevezett gateway-re kell bejelentkezni, majd onnan tovább SSH-zni a megfelelő gépre. A 12. ábrán szereplő VLAN beállításokat a StarBED sbeasy kezelőfelületén (sbeasy.lab.starbed.org) állítottam be. Az exp3 és exp4 interfészek jelölik a már említett 10 gigabites hálózati interfészeket.



13. ábra: StarBED sbeasy kezelőfelület

### 3.4 Siitperf telepítése a mérőgépre

A hardware-követelmények között szerepel, hogy DPDK kompatibilis hálózati kártyát használjunk, ezeknek a listája <https://core.dpdk.org/supported/> linken elérhető. A Siitperf külön processzormagot használ a fogadásra és a küldésre mindkét irányban, ezért 4 CPU magra van szüksége a kétirányú forgalomhoz a fő mag mellett, ami a főprogramot futtatja. A 4 magot már a rendszer bootolásánál le kell foglalni, hogy megelőzzünk bármiféle interferenciát más szoftverekkel. Ezt a következő sorral lehet megtenni a `/etc/default/grub` fájlban [12]:

```
GRUB_CMDLINE_LINUXDEFAULT="quiet isolcpus=2,4,6,8 hugepagesz=1G
hugepages=32"
```

Majd `update-grub` parancsot kell futtatni, hogy érvényesítse a grub rendszerbetöltőben a beállításokat. Ahogy a fenti sorból is látni lehet, 1 GB-os hugepage-eket használtam. Emiatt a `/etc/fstab` fájlban fel kell venni a következő sort [12]:

```
nodev    /mnt/huge_1GB hugetlbfs pagesize=1GB 0 0
```

És természetesen létre kell hozni a `/mnt/huge_1GB` könyvtárat is.

A `siitperf` Debian 9-en 16.11.9-1+deb9u1 DPDK verzióval került tesztelésre [12], így mérőgépemén is ezeket a verziókat használtam. Miután az előbbiek beállítása megtörtént és `reboot` paranccsal újraindítottuk a gépet, fel kell telepíteni néhány szükséges csomagot, hogy probléma nélkül lefusson az installálás a következő parancsok kiadásával:

```
apt update
apt install cpufrequtils git dpdk dpdk-dev libnuma-dev build-essential
```

Mérőprogramot letöltjük a githubról a következő paranccsal:

```
git clone https://github.com/lencsegabor/siitperf stateful verzió esetén
git clone https://github.com/lencsegabor/siitperf -b stateful
```

Így az aktuális könyvtárunkban `siitperf` mappa keletkezett, benne a mérőprogram fájljaival. Még telepítés előtt a `scripts` mappában találhatunk beállító parancsokat, amiket le kell futtassunk az installálás előtt. Be kell állítani a `dpdk sdk` (Software Development Kit) környezeti változóit a következő parancsokkal:

```
export RTE_TARGET="x86_64-default-linuxapp-gcc"
export RTE_SDK="/usr/share/dpdk/"
export RTE_INCLUDE="/usr/include/dpdk"
```

Majd a `/usr/lib/x86_64-linux-gnu` mappában lévő `libxenstore.so.3.0` fájlra létre kell hozni egy symbolic linket `libxenstore.so` néven az alábbi módon:

```
cd /usr/lib/x86_64-linux-gnu
ln -s libxenstore.so.3.0 libxenstore.so
```

Amint ezek elkészültek, visszatérhetünk a `siitperf` főkönyvtárába, és a `make` paranccsal lefordíthatjuk a programot. Ahhoz, hogy a mérőprogram a megfelelő MAC címeket használja a méréshez, a `siitperf.conf` fájlban be kell állítani a Tester portok MAC címeit, és ezeket le kell foglalni a `dpdk-devbind` paranccsal, ezzel átadva az interfészek kezelését a kerneltől a DPDK-nak, esetemben pl.:

```
dpdk-devbind --bind=uio_pci_generic 0000:05:00.0
dpdk-devbind --bind=uio_pci_generic 0000:05:00.1
```

A `siitperf.conf` fájlban be kell még állítani az előzőleg a `/etc/default/grub` fájlban lefoglalt processzormagokat a `CPU-L-Send`, `CPU-R-Recv`, `CPU-R-Send`, `CPU-L-Recv` sorokkal. Esetemben ez a 2,4,6,8-as számú magok lesznek.

### 3.5 Jool telepítése és működési módjai

Ahogy azt már említettem, a mérendő eszközökön a NICT Mexico által fejlesztett Joolt [7] alkalmaztam. Ennek is a 4.2.0-rc2 verzióját, mivel ebben található meg az állapotmentes, állapottartó címfordítás és a MAP-T modul is. Első lépésként a StarBED gateway-re letöltöttem `wget` segítségével a Jool kernel modult és a felhasználói eszköztárat a következő parancsokkal:

```
wget https://www.jool.mx/download/jool-dkms\_4.2.0.rc2-1\_all.deb
wget https://www.jool.mx/download/jool-tools\_4.2.0.rc2-1\_amd64.deb
```

Majd ezeket `scp` segítségével eljuttattam a p106 és p107 gépekre. Itt `apt update`-tel újra szinkronizáltam a csomag index fájlokat, majd `apt install linux-headers-$(uname -r)` paranccsal letöltöttem a legfrissebb kernel fejleceket. Ezek után `apt install ./jool-dkms_4.2.0.rc2-1_all.deb ./jool-tools_4.2.0.rc2-1_amd64.deb` módon telepítettem az implementációt.

Egy Jool instance két különböző forgalomelfogó, beépülőmodulokat engedélyező kernel alapú keretrendszer egyikéhez csatolható. Ez a netfilter és az iptables.[7]

**Netfilter Jool:** A netfilter egy „csapda” a Linux kernelben, ahová bizonyos modulok kódot tudnak beszúrni. Akármikor egy csomag eléri ezt a csapdát, a kernel végigfuttatja az összes megfelelő regisztrált modulon. Netfilter Joolt egyszerű konfigurálni, de a működése „mohó”. Ez azt jelenti, hogy nincsenek feltételek, mindent fordít, és ez az elsődleges prioritása. Mindent megpróbál lefordítani, és csak a maradékot kapja meg a többi hálózati alrendszer. Csak egy darab Netfilter SIIT Jool instance és egy Netfilter NAT64 instance fordulhat elő egy névtérben. A netfilter Jool instance egyből elkezdi fordítani a csomagot, amint az létrejött, és eldobja a sérülteket. Lefordítja, amit le tud, és minden mást visszaad a kernelnek. [7]

**Iptables Jool:** Az iptables a Netfilter fölé épült, NAT-olási és tűzfalas képességeiről ismert. Itt most az a funkciója a lényeges, amellyel meg tudjuk határozni, hogy az adott csomagot melyik cél-bővítmény kezelje. Bármennyi iptables Jool lehet bármelyik névtérben, és akárhány iptables szabállyal lehet hivatkozni rájuk. Ezek az instance-ok tétlenül „ülnek”, amíg valamilyen iptables szabály nem küld feléjük egy csomagot, ezek után úgy működik, mint a Netfilter Jool: lefordítja, amit le tud, eldobja a hibásnak vélt csomagokat, és minden mást visszaad a kernelnek. [7]

Amint később bemutatom majd, a két megoldás teljesítménye között nem találtam lényeges különbséget.

### **3.6 Csomagtovábbítás mérések**

Mielőtt rátérnék a technológiák mérésére, le kell tesztelni, mire képes a rendszerem. Ehhez csomagtovábbítás mérést alkalmaztam, amit lemértem IPv4, majd IPv6 használatával is. A csomagtovábbítás mérés annyit tesz, hogy megadom a megfelelő útvonalakat a mérendő eszközökön, és egy áteresztőképesség mérést hajtok végre. Ezt elvégeztem a 3 szerveres és a 4 szerveres mérési összeállításban is egytől egészen tizenhat processzormagig. Csomagtovábbítás mérésnél csak a CPU magszám szerinti skálázódás vizsgálatának van értelme, mivel a csomagtovábbítás kapcsolatmentes volta miatt a kapcsolatszám ebben az esetben nem értelmezhető.

A 14. ábraábrán látható a 3 szerveres mérési konfiguráció az IPv4 és IPv6 címekkel.



14. ábra: Mérési összeállítás csomagtovábbítás méréshez 3 szerverrel

A p106-os gépen IPv4 esetében az útválasztási táblázathoz (routing table) a 198.19.0.0/24-es hálózatot kellett hozzáadni, ehhez az `ip route add 198.19.0.0/24 via 198.18.50.2` parancsot alkalmaztam, a p107-es gépen pedig a 198.18.0.0 hálózatot, ehhez az `ip route add 198.18.0.0/24 via 198.18.50.1` parancsot alkalmaztam.

IPv6 esetében is hasonlóképpen nézek ki a parancsok, p106-os gép esetén `ip -6 route add 2001:2:0:3::/64 via 2001:2:0:2::2` parancssal adtam hozzá az útválasztó táblához a 2001:2:0:3::/64-es hálózatot, p107-es szerver esetén pedig `ip -6 route add 2001:2:0:1::/64 via 2001:2:0:2::1` parancssal a 2001:2:0:1::/64-es hálózatot.

Továbbá minden gépen engedélyeztem a csomagtovábbítást az `echo 1 > /proc/sys/net/ipv4/ip_forward`, illetve az `echo 1 > /proc/sys/net/ipv6/conf/all/forwarding` parancsok segítségével. Ahhoz, hogy a hálózati interfész megszakításkéréseit a kernel szétossza a magok között, az `ethtool -N enp5s0f{0,1} rx-flow-hash udp{4,6} sdfn` parancsot használtam a mérendő eszközökön. Minden mérésnél „négy kilencest” alkalmaztam: ez azt jelenti, hogy ahhoz, hogy egy tesztet sikeresnek tekintsek, legalább a keretek 99,99%-ának meg kellett értekeznie, másképpen szólva 0,01% keretvesztést engedtem meg, mivel az előzetes mérések során a rendszerben kismértékű veszteséget tapasztaltam. A 3 szerveres IPv4 csomagtovábbítás tesztek eredményeit a 3. táblázat mutatja be.

| IPv4 routing test |           |           |             |             |             |
|-------------------|-----------|-----------|-------------|-------------|-------------|
| num. cores        | 1         | 2         | 4           | 8           | 16          |
| error             | 1         | 1         | 1           | 1           | 1           |
| frame size        | 64        | 64        | 64          | 64          | 64          |
| num. Exp.         | 20        | 20        | 20          | 20          | 20          |
| throughput median | 441 776,5 | 770 016,0 | 1 561 423,0 | 3 046 874,0 | 5 742 186,0 |
| throughput min    | 430 657,0 | 749 937,0 | 1 519 006,0 | 2 968 744,0 | 5 687 499,0 |
| throughput max    | 442 969,0 | 773 438,0 | 1 564 819,0 | 3 070 678,0 | 5 742 186,0 |
| tp. rel. scale up | 1         | 0,871     | 0,884       | 0,862       | 0,812       |
| IPv6 routing test |           |           |             |             |             |
| num. cores        | 1         | 2         | 4           | 8           | 16          |
| error             | 1         | 1         | 1           | 1           | 1           |
| frame size        | 84        | 84        | 84          | 84          | 84          |
| num. Exp.         | 20        | 20        | 20          | 20          | 20          |
| throughput median | 381 571,0 | 612 985,0 | 1 215 935,0 | 2 316 997,0 | 2 788 931,0 |
| throughput min    | 374 999,0 | 601 561,0 | 1 195 311,0 | 2 279 783,0 | 2 777 342,0 |
| throughput max    | 382 032,0 | 614 258,0 | 1 218 751,0 | 2 324 234,0 | 2 794 433,0 |
| tp. rel. scale up | 1         | 0,803     | 0,797       | 0,759       | 0,457       |

3. táblázat: 3 szerveres csomagtovábbítás mérés eredményei

A négy szerveres összeállítást a 15. ábra mutatja IPv4 és IPv6 címekkel.



15. ábra: Mérési összeállítás csomagtovábbítás méréshez 4 szerverrel

Itt is ugyanúgy be kellett állítanom az útvonalakat az egyes mérendő eszközökön. Így a korábbi paranccsal IPv4 esetében hozzáadtam a DUT1-hez a 198.19.0.0/24 hálózatot, a DUT2-höz a 198.18.0.0/24 és a 198.19.0.0/24 hálózatokat, a DUT3-hoz pedig a 198.18.0.0/24-es hálózatot.

IPv6 esetében a DUT1-hez a 2001:2:0:4::/64-es, DUT2-höz a 2001:2:0:1::/64 és a 2001:2:0:4::/64-es, DUT3-hoz pedig a 2001:2:0:1::/64-es hálózatokat. Ezek után itt is engedélyeztem a

csomagtovábbítást a mérendő eszközökön, majd beállítottam a megszakításkérések szétosztását a processzormagok között. A 4. táblázat mutatja a mérések eredményeit.

| <b>IPv4 routing test</b> |                  |                  |                    |                    |                    |
|--------------------------|------------------|------------------|--------------------|--------------------|--------------------|
| num. cores               | 1                | 2                | 4                  | 8                  | 16                 |
| error                    | 1                | 1                | 1                  | 1                  | 1                  |
| frame size               | 64               | 64               | 64                 | 64                 | 64                 |
| num. Exp.                | 20               | 20               | 20                 | 20                 | 20                 |
| <b>throughput median</b> | <b>446 902,5</b> | <b>787 604,0</b> | <b>1 534 675,0</b> | <b>2 980 493,0</b> | <b>5 687 499,0</b> |
| throughput min           | 424 999,0        | 781 249,0        | 1 497 034,0        | 2 812 499,0        | 5 646 162,0        |
| throughput max           | 447 488,0        | 789 207,0        | 1 538 452,0        | 2 988 282,0        | 5 687 499,0        |
| tp. rel. scale up        | 1                | 0,881            | 0,859              | 0,834              | 0,795              |
| <b>IPv6 routing test</b> |                  |                  |                    |                    |                    |
| num. cores               | 1                | 2                | 4                  | 8                  | 16                 |
| error                    | 1                | 1                | 1                  | 1                  | 1                  |
| frame size               | 84               | 84               | 84                 | 84                 | 84                 |
| num. Exp.                | 20               | 20               | 20                 | 20                 | 20                 |
| <b>throughput median</b> | <b>381 338,5</b> | <b>598 834,5</b> | <b>1 176 820,0</b> | <b>2 331 845,0</b> | <b>2 906 249,0</b> |
| throughput min           | 374 999,0        | 593 749,0        | 1 152 188,0        | 2 292 444,0        | 2 904 295,0        |
| throughput max           | 382 032,0        | 602 417,0        | 1 179 688,0        | 2 337 891,0        | 2 910 157,0        |
| tp. rel. scale up        | 1                | 0,785            | 0,772              | 0,764              | 0,476              |

4. táblázat: 4 szerveres csomagtovábbítás mérés eredményei

IPv4 esetében az eredményekből azt lehet látni, hogy mindkét tesztrendszer jól skálázódik, csak egészen kis különbség figyelhető mind a teljesítményük, mint a skálázódásuk között. Egy CPU magon a négy gépes, tizenhat processzormagon pedig a három szerveres rendszer teljesítménye volt nagyobb, így (kis mértékben) a három gépes rendszer skálázódott jobban.

IPv6 mérések esetén egy processzormagon kis eltéréssel egyeznek az eredmények, majd mindkét rendszere teljesítménye szépen skálázódik egészen nyolc magig, ezt követően tizenhat CPU magnál a skálázódás jelentősen romlik. (Ennek az okának a vizsgálata meghaladja a dolgozatom kereteit, viszont ez a jelenség magyarázatul szolgál majd a megvizsgált 464XLAT és MAP-T implementációk skálázódásának a viselkedésére.)

### 3.7 464XLAT konfigurálása

A 464XLAT méréseket a 16. ábraábrán látható összeállításban végeztem.



16. ábra: 464XLAT mérési konfiguráció

A Tester IP címét a `siitperf.conf` fájlban állítottam be. A Tester CLAT felőli hálózati interfésze a 198.18.0.2-es címet kapta, a PLAT oldal felőli pedig a 198.19.0.2-es címet. Ezeket az *A függelékben* található `siitperf.conf` fájl `IPv4-L-Real` és az `IPv4-R-Real` soraiban lehet megtalálni. Az előbbi a bal oldali, tehát a CLAT felőli címet állítja, utóbbi pedig a jobboldali, azaz PLAT felőli címet adja meg. Ezen kívül a konfigurációs fájlban még be kell állítani a mérőgép használt interfészeinek MAC címét, esetemben az `enp5s0f0` és az `enp5s0f1`-ét. Ezt a `MAC-L-Tester` és a `MAC-R-Tester` sorokkal tehetjük meg. Ugyanígy meg kell adni a DUT MAC címeit, itt most a vizsgált 464XLAT rendszer PLAT és CLAT része a „mérendő eszköz” így mindkettőnek a Tester oldali interfész MAC címeit kell a `MAC-L-DUT` és a `MAC-R-DUT` sorokhoz beírunk. Nálam ez a következőképp nézett ki:

```
IPv4-L-Real 198.18.0.2
IPv4-R-Real 198.19.0.2
```

```
MAC-L-Tester a0:36:9f:c5:e9:74 # StarBED p105 enp5s0f0
MAC-R-Tester a0:36:9f:c5:e9:76 # StarBED p105 enp5s0f1
MAC-L-DUT a0:36:9f:c5:d3:1c # StarBED p106 enp5s0f0
MAC-R-DUT a0:36:9f:c5:d2:e2 # StarBED p107 enp5s0f1
```

A CLAT IP címeit a `/etc/network/interfaces` fájlban állítottam be, hogy az esetleges újraindítások miatt ne kelljen parancssorral mindig beállítani a címeket. A Tester oldali porton egy

IPv4-es címet konfiguráltam: az interfésznek a 198.18.0.1-es címet adtam /24-es alhálózati maszkkal. A CLAT és PLAT közötti hálózat tisztán IPv6-on működik, ezzel a szolgáltató IPv6 hozzáférési- és gerinchálózatát szimuláltam. Itt a 2001:2:0:100::1-es címet kapta a CLAT. Továbbá itt beállítottam a *Joolt* stateless NAT64 funkcióra a következők szerint:

```
1) modprobe jool_siit
2) ip route add 64:ff9b::/96 via 2001:2:0:100::2
3) jool_siit instance add --netfilter --pool6 64:ff9b::/96
4) jool_siit eamt add 198.18.0.2/24 2001:2:0:2::2/120
5) ip neigh add 198.18.0.2 lladdr a0:36:9f:c5:e9:74 dev enp5s0f0 nud
    permanent
```

Az 1) pontban betöltöttem a Jool állapotmentes modulját. A 2) pontban hozzáadom a 64:ff9b::/96 prefixet az útválasztó táblához, hogy a PLAT felé irányítsa. A 3) pontban a Jool állapotmentes moduljához is hozzáadom ezt a prefixet alapértelmezett Jool instance-ként, majd a 4) pontban a Tester IPv4-es címének fordítására állítunk be egy IPv6-os címet. Az 5) pontban pedig, mivel a mérőgép interfészei csak a mérés alatt aktívak, és a *siitperf* nem képes ARP üzenetekre válaszolni, statikusan hozzá kell adnunk az ARP táblához a Tester címeit. Ezzel a 464XLAT CLAT hálózatelemét beállítottam.

A PLAT IP címeit ugyanúgy, mint a CLAT-nál, a `/etc/network/interfaces` fájlban beállítottam a 16. ábra szerint. CLAT felől IPv6 hálózatban a 2001:2:0:100::2 IP címet kapta /64-es maszkkal. Tester oldalon a 198.19.0.1-es címet /24-es alhálózati maszkkal. A PLAT-on a Joolt a következőképp konfiguráltam:

```
1) modprobe jool
2) ip route add 2001:2:0:2::/64 via 2001:2:0:100::1
3) jool instancece add --netfilter --pool6 64:ff9b::/96
4) jool pool4 add 198.19.0.1 --udp 1-65535
5) ip neigh add 198.19.0.2 lladdr a0:36:9f:c5:e9:76 dev enp5s0f1 nud
    permanent
```

Az 1) pontban betöltöttem a Jool állapottartó modulját. A 2) pontban hozzáadom az útválasztó táblához a CLAT-on található hálózatokat. A 3) pontban hozzáadom a címfordító algoritmusoknak fenntartott prefixet a Joolhoz, mint IPv6 címtartományt. A 4) pontban hozzáadom a PLAT Tester oldali címét az IPv4-es címtartományhoz. Végül az 5) pontban elvégzem az ARP táblába történő statikus bejegyzést. Ezzel a 464XLAT működőképes lett.

### 3.8 464XLAT mérése

A 464XLAT már működőképes, de még hiányzik néhány beállítás a mérés elindításához. Két mérést végeztem a hálózaton. Először a technológia teljesítőképességét vizsgáltam a PLAT CPU mag számának függvényében, de azonos kapcsolat szám mellett, majd 16 processzormaggal a PLAT-on megmértem hogyan változik a teljesítőképesség a kapcsolatok számának változásával. A 464XLAT architektúrában található állapottartó rész, ezért a siitperf stateful verzióját használtam hozzá. Mivel a siitperf még erősen fejlesztés alatt áll, ezért lényeges, hogy a felhasznált verzió a stateful branch, latest commit: 2021. 08. 16., d48e33b.

Ahhoz, hogy az első mérést el tudjam indítani, a következő sorokat be kell állítani a `siitperf.conf` fájlban:

```
Fwd-var-sport 3 # álvéletlen a megadott tartományban
Fwd-var-dport 3 # álvéletlen a megadott tartományban
Fwd-sport-min 1
Fwd-sport-max 4000
Fwd-dport-min 1
Fwd-dport-max 1000

Stateful 1 # stateful mérés, Initiator a bal oldalon van
Enumerate-ports 4 # A későbbi verzióban ez változott, 3 lett.
Responder-ports 3 # álvéletlen választás az állapottáblából
```

Az `Fwd-var-sport` és `Fwd-var-dport` sorokkal a forrás- és célportokat állítjuk előre (forward, fwd: balról jobbra) irányban, 0 értékkel fix, 1-esnél növekedő, 2-esnél csökkenő, 3-asnál pedig véletlenszerű portszámokat használ a mérőprogram. Az `Fwd-sport-min` beállítással az előre irány minimum forrásportjának számát adjuk meg, a `Fwd-sport-max` meghatározza ennek a maximum értékét. Az `Fwd-dport-min` és `Fwd-dport-max` ugyanezeket állítjuk, csak a célporttal.

A `Stateful` sorral a `siitperf stateful` részét kezeljük, ha ez az érték 0, az állapottartó rész kikapcsolt állapotban van, ha 1, akkor állapottartó mérést indít úgy, hogy a kezdeményező a baloldalon, a válaszadó pedig a jobb oldalon található. Ilyenkor új parancssoros paramétereket vár a program. Ha ez az érték 2, akkor ugyanúgy stateful mérést indít csak épp ellenkezőleg, a kezdeményező van a jobboldalon, és a válaszadó pedig a baloldalon.[16]

`Enumerate-ports` sorral az előzetes fázisban portszámok generálása és sorrendje állítható. Ha az értéke 0, a funkció kikapcsolt állapotban van, 1-es értéknél növekvő sorrendben generálja le a portokat, 2-es értéknél csökkenő sorrendben. A 3-as érték egy kísérleti verzió, jelenleg nem működik. 4-es értéknél véletlenszerű permutációt használ, az összes portszámot legenerálja, majd véletlen sorrendbe rakja a mérőprogram. (A program későbbi verziójában ezt a 3-as érték jelenti.) [16]

A `Responder-ports` sorral a legenerált portszámok (pontosabban számnégyesek) kiválasztásának irányelve adható meg. Ha az értéke 0, akkor a válaszadó fixen mindig ugyanazt a számnégyest használja, amit az előzetes tesztfázis során legelsőként vett tesztkeretben talált. 1-es értéknél növekvő sorrendben, 2-es értéknél csökkenő sorrendben olvassa ki a számnégyeseket az állapotablájából. Ha az érték 3, akkor pedig véletlenszerűen választ számnégyeseket az állapotablájából. [16]

Amint ezeket beállítottuk, áttérhetünk a mérőszkriptekre. Kettőt használtam, ezek a `binary-rate-alg.sh-R` és a `binary-rate-alg.sh-sf` nevű fájlok. Az előbbi bináris kereséssel a *maximum connection establishment rate*-et határozza meg, tehát hogy mennyi keret/másodperc sebességgel tölthető fel a DUT állapotablája. Az utóbbi szkript az előző adat birtokában bináris kereséssel megméri a hálózat áteresztőképességét. Mind a két szkriptben vannak alapból beállított értékek, én azokat fogom most kiemelni, amiket átállítottam.

A maximum connection establishment rate-nél be kell állítani a mérésnek egy nevet, ezt az `impl` változóval lehet megtenni; egy maximum csomag sebességet, itt meg kell határozni egy alkalmas felső korlátot keret/másodperc értékre, ami a méréskor a bináris keresés intervallumának a felső határa lehet, ez a `max` változó. A `siitperf.conf` fájlban megadott beállítások alapján meg kell adni az előzetes fázisban elküldött csomagok számát, ez az `N` változó, és az állapotábla méretét, ez az `M` változó. A `DUT_IP` változóval a mérendő eszköz IP címét lehet beállítani, ahol a szkript `ssh` segítségével lefuttat további szkripteket, amivel a mérés elején állít be timeout értéket, illetve a mérés végén újraindítja az adott implementációt, és törli az állapotablát. Ezt a `startcmd` és a `stopcmd` változókkal lehet megadni. A `binary-rate-alg.sh-sf`-nél kiegészül még egy sorral a beállítás, ami az `R`-es végű szkripttel mért connection establishment rate (illetve annál valamivel kisebb, „biztonságos” érték). Tehát az előzőleg mérték alapján kell megadni az `R` változóval, hogy mennyi keret/másodperccel töltsse fel az állapotablát. Ennek megfelelően állítottam be a méréshez a szkripteket. Az összes mérést 84 bájtos keretméret mellett mértem, ezt az `fs` változó adja meg. (Ez mindig az IPv6 csomagot tartalmazó keret méretét jelenti. A neki megfelelő IPv4 csomagot tartalmazó keret méret 20 oktettel kisebb, tehát jelen esetben 64 byte.)

Először 464XLAT-on a CPU magok száma szerinti skálázódás mérést végeztem el. Itt az aktív processzormagok számának beállításához a `p107` (PLAT) serveren az `/etc/default/grub` fájlban a `GRUB_CMDLINE_LINUX_DEFAULT="quiet idle=poll maxcpus=X"` soron belül a `maxcpus=X` beállítással állítottam a magok számát  $X = 1, 2, 4, 8$  és  $16$  értékekre. Azért ezen a gépen állítottam a magok számát, mert ennek a teljesítményére voltam kíváncsi, ezért az volt a célom, hogy

ez legyen a szűk keresztmetszet a mérések során. Továbbá mindkét DUT-on bekapcsoltam az RSS-t (Receive-Side Scaling, fogadó oldali skálázódás), ami a hálózati interfész megszakítás kéréseit osztja szét az egyes processzormagok között a következő parancs megfelelő variációjának használatával:

```
ethtool -N enp5s0f{0,1} rx-flow-hash udp{4,6} sdfn
```

A CPU szám skálázódásnál minden mérés végeztekor, a processzormag számának változása miatt egy power cycle (cold boot)-tal újraindítottam a gépeket, kapcsolatszám skálázódáskor, mivel nem történt magszám változás, ezt nem tettem meg. A további méréseim során is így jártam el.

Az 5. táblázat: 464XLAT CPU skálázódás mérési eredményekben láthatók a CPU skálázódás mérési eredményei:

| num. cores        | 1               | 2               | 4               | 8               | 16              |
|-------------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| src ports         | 4000            | 4000            | 4000            | 4000            | 4000            |
| dst ports         | 1000            | 1000            | 1000            | 1000            | 1000            |
| num. exp.         | 20              | 20              | 20              | 20              | 20              |
| error             | 1               | 1               | 1               | 1               | 1               |
| cps median        | <b>210343,5</b> | <b>328212,0</b> | <b>423903,0</b> | <b>448816,5</b> | <b>479480,0</b> |
| cps min           | 208396,0        | 324595,0        | 417039,0        | 440014,0        | 468751,0        |
| cps max           | 212501,0        | 331349,0        | 428980,0        | 487501,0        | 487501,0        |
| cps rel. scale up | 1               | 0,780           | 0,504           | 0,267           | 0,142           |
| throughput median | <b>108973,0</b> | <b>186371,0</b> | <b>236067,5</b> | <b>245029,5</b> | <b>246902,0</b> |
| throughput min    | 107811,0        | 182774,0        | 232804,0        | 242124,0        | 242965,0        |
| throughput max    | 109389,0        | 187501,0        | 239063,0        | 247657,0        | 250001,0        |
| tp. rel. scale up | 1               | 0,855           | 0,542           | 0,281           | 0,142           |

5. táblázat: 464XLAT CPU skálázódás mérési eredmények

A táblázatból lehet látni, hogy 4 magig viszonylag jól skálázódnak az eredmények, felette lassabb növekedés tapasztalható.

Ezekután megnéztem egy munkaponton, hogy tapasztalható-e különbség a Jool netfilteres beállítása és az iptables-es konfigurációja között. Ehhez a jool instance add --netfilter --pool6 64:ff9b::/96 parancs helyett a következő parancsokat kellett használni:

- 1) jool instance add --iptables --pool6 64:ff9b::/96
- 2) iptables -t mangle -A PREROUTING -j JOOL --instance "default"
- 3) ip6tables -t mangle -A PREROUTING -j JOOL --instance "default"

Az 1) paranccsal hozzáadom a WKP-t a default Jool instance-hoz a 2) és 3) paranccsal a default Jool instance-ra irányítom a csomagokat iptables segítségével. Ezt a mérést 16 magon futtattam, 4000 forrás- és 1000 célport használatával, 84 bájtos keretmérettel. A táblázatban látható eredmények születtek.

|                   | netfilter       | netfilter       | iptables        |
|-------------------|-----------------|-----------------|-----------------|
| num. cores        | 1               | 16              | 16              |
| src ports         | 4000            | 4000            | 4000            |
| dst ports         | 1000            | 1000            | 1000            |
| num. exp.         | 20              | 20              | 20              |
| error             | 1               | 1               | 1               |
| cps median        | <b>202630,0</b> | <b>488387,5</b> | <b>488825,5</b> |
| cps min           | 199997,0        | 481053,0        | 482699          |
| cps max           | 206446,0        | 492190,0        | 500391          |
| cps rel. scale up | 1               | 0,151           | 0,151           |
| throughput median | <b>118043,5</b> | <b>247394,5</b> | <b>246852,5</b> |
| throughput min    | 117173,0        | 245283,0        | 242954          |
| throughput max    | 118897,0        | 264844,0        | 250001          |
| tp. rel. scale up | 1               | 0,131           | 0,131           |

6. táblázat: netfilter Jool és iptables jool összehasonlítása

Ezek alapján a kétfajta megvalósítás között érdemi különbség nem tapasztalható, így én az egyszerűbb beállítás miatt mindig a netfilteres megoldást használtam a méréseim során.

Ezután megmértem, 16 CPU maggal, hogyan skálázódik a 464XLAT teljesítménye a kapcsolatok számának változásával. Itt a Testeren a `siitperf.conf` fájlban a `Fwd-sport-min`, `Fwd-sport-max`, `Fwd-dport-min`, `Fwd-dport-max` sorokat kellett a megfelelő értékre állítanom, hogy megfelelő számú portszám kombinációkat generáljon a mérőprogram. Továbbá figyelnem kellett, hogy elég nagy UDP Timeout értéket válasszak az egyes mérések során. Ezt a mérés elején lévő `startcmd` változóval állítottam, ami `ssh-zik` a `p107-es` gépre és az ott lévő `set-jool` szkripttel beállította az általam meghatározott értéket. A `set-jool` szkript tartalma a következőképp nézett ki:

```
jool global update udp-timeout $1
```

Minden méréshez használtam egy újraindító szkriptet, amit a mérés végén a `stopcmd` változó futtatott. Ezzel a Joolt indítottam újra, majd indítás után beállítottam az előzőekben leírtak alapján a 464XLAT-ot. Ezt a következőképp írtam meg:

- 1) `modprobe -r jool`
- 2) `modprobe jool`
- 3) `jool instance add --netfilter --pool6 64:ff9b::/96`
- 4) `jool pool4 add 198.19.0.1 --udp 1-65535`

A mérések során a fix méretű forrásportszám tartományt használtam annak érdekében, hogy a 464XLAT eredményei összevethetők legyenek a MAP-T eredményeivel. A kapcsolatszám növekedését a célportszám tartomány méretének duplázásával értem el.

A kapcsolatszám szerinti skálázódás mérésének eredményei a 7. táblázatban láthatók. A növekvő kapcsolatszám függvényében mind a maximális kapcsolatfelépítési sebesség, mind a throughput monoton csökken, a romlás viszonylag egyenletes, a vizsgált tartományban semmilyen értéknél sem található hirtelen nagy csökkenés.

|                    |                   |                   |                    |                    |                   |
|--------------------|-------------------|-------------------|--------------------|--------------------|-------------------|
| num. cores         | 16                | 16                | 16                 | 16                 | 16                |
| src ports          | 4 000             | 4 000             | 4 000              | 4 000              | 4 000             |
| dst ports          | 250               | 500               | 1 000              | 2 000              | 4 000             |
| num. conn.         | <b>1 000 000</b>  | <b>2 000 000</b>  | <b>4 000 000</b>   | <b>8 000 000</b>   | <b>16 000 000</b> |
| initial R rate     | 500 000           | 450 000           | 400 000            | 400 000            | 350 000           |
| calculated timeout | 5,000             | 7,444             | 13,000             | 23,000             | 48,714            |
| used timeout       | 300               | 300               | 300                | 300                | 300               |
| num. exp.          | 20                | 20                | 20                 | 20                 | 20                |
| error              | 1                 | 1                 | 1                  | 1                  | 1                 |
| cps median         | <b>576 789,5</b>  | <b>514565,5</b>   | <b>459088,5</b>    | <b>425 400</b>     | <b>394 229</b>    |
| cps min            | 549 999,0         | 506233            | 451170             | 419 749            | 388 181           |
| cps max            | 593 751,0         | 519598            | 468751             | 432 617            | 398 804           |
| throughput median  | <b>305 570,5</b>  | <b>277301</b>     | <b>247 807</b>     | <b>226 836</b>     | <b>210 673</b>    |
| throughput min     | 298 423,0         | 273832            | 243 553            | 224 119            | 206 249           |
| throughput max     | 307 946,0         | 281251            | 250 199            | 229 706            | 213 135           |
| num. cores         | 16                | 16                | 16                 | 16                 |                   |
| src ports          | 4 000             | 4 000             | 4 000              | 4 000              |                   |
| dst ports          | 8 000             | 16 000            | 32 000             | 64 000             |                   |
| num. conn.         | <b>32 000 000</b> | <b>64 000 000</b> | <b>128 000 000</b> | <b>256 000 000</b> |                   |
| initial R rate     | 350 000           | 300 000           | 250 000            | 250 000            |                   |
| calculated timeout | 94,429            | 216,333           | 515,000            | 1 027,000          |                   |
| used timeout       | 300               | 600               | 1 200              | 1 200              |                   |
| num. exp.          | 20                | 20                | 20                 | 20                 |                   |
| error              | 1                 | 1                 | 1                  | 1                  |                   |
| cps median         | <b>357 846</b>    | <b>327 910</b>    | <b>290 325</b>     | <b>272 061</b>     |                   |
| cps min            | 351 561           | 324 217           | 286 861            | 261 868            |                   |
| cps max            | 369 690           | 337 524           | 303 223            | 276 563            |                   |
| throughput median  | <b>192 187</b>    | <b>170 340</b>    | <b>152 255</b>     | <b>139 526</b>     |                   |
| throughput min     | 189 676           | 168 379           | 149 999            | 138 275            |                   |
| throughput max     | 193 946           | 173 996           | 153 552            | 140 653            |                   |

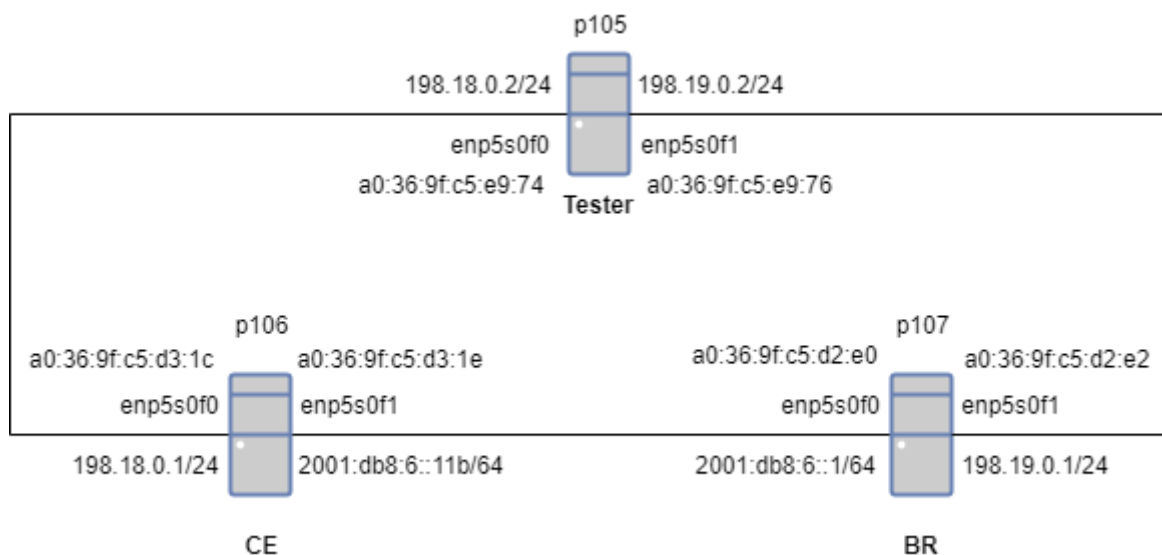
7. táblázat: 464XLAT kapcsolatszám skálázódás eredmények

### 3.9 MAP-T Tervezése

Még mielőtt elkezdeném konfigurálni a MAP-T technológiát, tekintsük át, hogyan is kaptam meg a beállításokhoz szükséges címeket. Méréseimhez a 192.0.2.0/24-es publikus IPv4 címtartományt választottam. Ebből adódik az IPv4 suffix (utótag), amit 8 biten tudunk ábrázolni. Az eddigi mérésekhez 4000 forrás- és 1000 célportot használtam, hogy ne legyen kihasználva teljesen ez a tartomány (mivel itt most a forrásport a lényeges) eggyel nagyobb intervallumot választottam, így 8192 széles tartományt alkalmaztam. Ezzel a teljes portszám tartományt 8 részre osztottam. Ezt a 8 *Port Setet* 3 biten lehet ábrázolni. Így az EA (Embedded Address, beágyazott cím) bitek hossza  $8+3=11$  lesz. A CE eszköz azonosítója 11b hexadecimálisan megadva, ezt átírva binárisba: **00100011011**. Ebből az első 8 bit az **IPv4 suffix** az utolsó 3 pedig a *PSID*. Ezeket visszaírva a megfelelő alakokba IPv4 címként a 192.0.2.35/32-es címet kapjuk a 3-as Port Settel. Mivel 8192 méretű Port Seteket alkalmazok, ez azt jelenti, hogy a portszám tartomány az 24576-tól kezdődik, és 32767-nél ér véget. A BMR a következőképp alakul: 2001:db8:ce::/53, 192.0.2.0/24, EA bitek: 11. A DMR pedig a 64:ff9b::/96 lesz.

### 3.10 MAP-T konfigurálása három szerverrel

A MAP-T megoldást a 17. ábra alapján konfiguráltam.



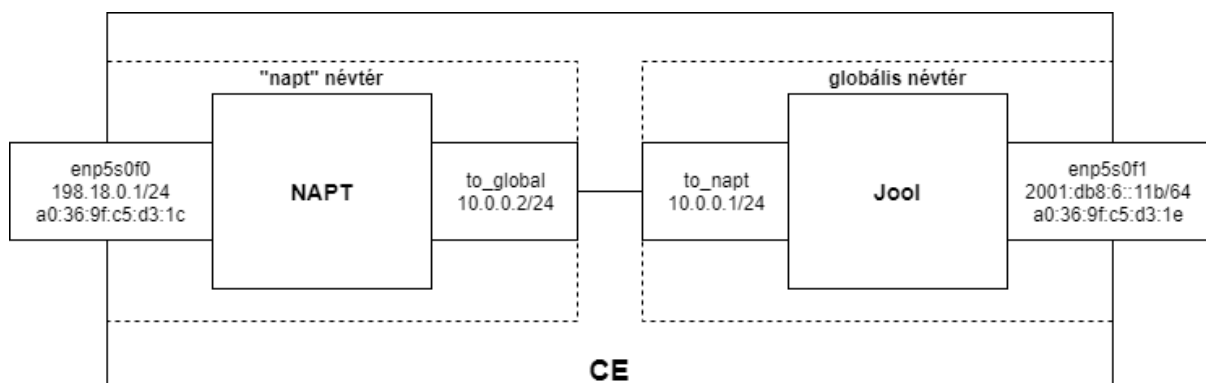
17. ábra: MAP-T konfiguráció

A Tester IP címeit a `siitperf.conf` fájlban állítottam be. A CE felőli portja a 198.18.0.2/24-es címet, a BR felőli pedig a 198.19.0.2/24-es címet kapta. Ugyanúgy, mint a 464XLAT-nál, itt is megadtam a Tester és a DUT MAC címeit a megfelelő sorokban. Ami a következőképp nézett ki:

```
IPv4-L-Real 198.18.0.2
IPv4-R-Real 198.19.0.2
```

```
MAC-L-Tester a0:36:9f:c5:e9:74 # StarBED p105 enp5s0f0
MAC-R-Tester a0:36:9f:c5:e9:76 # StarBED p105 enp5s0f1
MAC-L-DUT a0:36:9f:c5:d3:1c # StarBED p106 enp5s0f0
MAC-R-DUT a0:36:9f:c5:d2:e2 # StarBED p107 enp5s0f1
```

A CE beállításai a 18. ábrán láthatók.



18. ábra: MAP-T CE konfiguráció (készült [7] alapján)

A Jool dokumentációjában leírt eljárást [7] követve, először csináltam egy „napt” névtérrel az `ip netns add napt` paranccsal. Majd létrehoztam két virtuális hálózati interfészt, amivel összekötöttem a globális névtérrel az előbb megalkotott „napt” névtérrel. Ezt az `ip link add to_napt type veth peer name to_global` parancs kiadásával oldottam meg. Ezt követően az `enp5s0f0` és a `to_global` interfészt hozzáadom a „napt” névtérhez az `ip link set enp5s0f0 netns napt` és az `ip link set to_global netns napt` beállítások végrehajtásával. Az alábbi sorokkal aktív állapotba helyeztem az interfészeket, majd beállítottam az IP címüket:

- 1) `ip netns exec napt ip link set to_global up`
- 2) `ip netns exec napt ip link set enp5s0f0 up`
- 3) `ip link set to_napt up`
- 4) `ip addr add 10.0.0.1/24 dev to_napt`
- 5) `ip netns exec napt ip addr add 10.0.0.2/24 dev to_global`
- 6) `ip netns exec napt ip addr add 198.18.0.1/24 dev enp5s0f0`
- 7) `ip link set enp5s0f1 up`
- 8) `ip addr add 2001:db8:6::11b/64 dev enp5s0f1`

Ezután következett az útvonalak beállítása:

- 1) `ip netns exec napt ip route add default via 10.0.0.1`
- 2) `ip route add 64:ff9b::/96 via 2001:db8:6::1`
- 3) `ip route add 192.0.2.35/32 via 10.0.0.2`

A „napt” névtérben beállítottam az alapértelmezett útvonalat a to\_napt interfész felé, ez az 1) jelű parancs. A 2) paranccsal hozzáadtam a WKP-t a BR-en keresztül. A 3) parancs pedig a NAPT névtérben található NAT44 alrendszer felé irányítja a to\_global felől érkező csomagokat.

Ezután a NAT-olást állítottam be iptables segítségével, majd engedélyeztem az IPv4-es és IPv6-os csomagok továbbítását mind a „napt”, mind a globális névtérben.

- 1) `ip netns exec napt iptables -t nat -A POSTROUTING -s 198.18.0.0/24 -o to_global -p udp -j SNAT --to-source 192.0.2.35:24576-32767`
- 2) `ip netns exec napt sysctl -w net.ipv4.conf.all.forwarding=1`
- 3) `ip netns exec napt sysctl -w net.ipv6.conf.all.forwarding=1`
- 4) `sysctl -w net.ipv4.conf.all.forwarding=1`
- 5) `sysctl -w net.ipv6.conf.all.forwarding=1`

A 192.0.2.35 IPv4 cím és a hozzá tartozó portszám tartomány a 24576-32767 számított eredmények, amelyek függenek a szabály IPv6 prefixtől, a CE interfész azonosítójától és az EA bitek hosszától.

Amint ezeket beállítottam, jöhet a Jool konfigurálása:

- 1) `modprobe jool`
- 2) `modprobe jool_mapt`
- 3) `jool_mapt instance add "CE 11b" --netfilter --dmr 64:ff9b::/96`
- 4) `jool_mapt -i "CE 11b" global update end-user-ipv6-prefix 2001:db8:ce:11b::/64`
- 5) `jool_mapt -i "CE 11b" global update bmr 2001:db8:ce::/53 192.0.2.0/24 11 0`
- 6) `jool_mapt -i "CE 11b" global update map-t-type CE`

Az 1) és 2) paranccsal betöltöttem a Joolt és a MAP-T modulját. A 3) parancs segítségével létrehozok egy Jool instance-ot „CE 11b” néven, ami a WKP-t használja. A 4) pontban megadom a végfelhasználói IPv6 prefixet. Az 5) pontban beállítom az alapvető leképzési szabályt (BMR), melyben a sor végén egy 0 áll, ami azt jelenti, hogy az első port set magában foglalja az ún. well-known portokat is, majd a 6) pontban megadom az eszköz típusát, ami a CE.

Mivel a mérőgép interfészei csak a mérés alatt aktívak, és a siitperf nem képes ARP üzenetekre

válaszolni, hozzá kell adnunk az ARP táblához a Tester címeit, viszont itt speciális módszerrel, mivel az enp5s0f0 interfész már a „napt” névtérben van. Ez a következőképp alakul:

```
ip netns exec napt ip neighbor add 198.18.0.2 lladdr a0:36:9f:c5:e9:74 dev enp5s0f0 nud permanent.
```

Ezzel a CE konfigurációja elkészült. A BR-t lényegesen egyszerűbb konfigurálni, ennek érdekében a /etc/network/interfaces fájlban megadtam az egyes interfészek IP címeit. A baloldali hálózati interfésze a 2001:db8:6::1/64-es címet kapta, a jobboldali pedig a 198.19.0.1/24-es címet. Ezután a Joolt konfiguráltam az alábbi módon.

- 1) modprobe jool
- 2) modprobe jool\_mapt
- 3) jool\_mapt instance add "BR" --netfilter --dmr 64:ff9b::/96
- 4) jool\_mapt -i "BR" fmrt add 2001:db8:ce::/53 192.0.2.0/24 11 0
- 5) jool\_mapt -i "BR" global update map-t-type BR
- 6) ip route add 2001:db8:ce:11b::/64 via 2001:db8:6::11b

Az 1) és 2) parancsokkal a már látott módon betöltöttem a Joolt és MAP-T modulját. Ezt követően a 3) pontban hozzáadtam egy Jool instance-t „BR” néven, ami a WKP-t használja. A 4) pontban a „BR” instance-ben beállítom a továbbító leképzési szabályt (FMR) és az EA biteket, a sor végén a 0-val, ami azt jelenti, hogy az első port set magában foglalja az ún. well-known portokat is. Az 5) pontban pedig az eszköz típusát adom meg. A 6) pontban pedig az útválasztó táblához hozzáadom a CE által fordított IPv6 prefixet.

Még az előbbi módon, a mérőprogram működése miatt, hogy nem kezeli az ARP üzeneteket, a következő sort le kell futtatni.

```
ip neighbor add 198.19.0.2 lladdr a0:36:9f:c5:e9:76 dev enp5s0f1 nud permanent
```

Ezzel a BR-t konfiguráltam, így a MAP-T működik.

### 3.11 MAP-T mérése három szerverrel

Habár a MAP-T működőképes, a mérőcsomagok megérkeznek az egyik oldalról a másikra, itt is van néhány beállítás, amit még meg kell tenni. Ahhoz, hogy a hálózati interfészek megszakítás kéréseit elosszam a magok között, a következő beállításokat alkalmaztam a CE-n:

```
ip netns exec napt ethtool -N enp5s0f0 rx-flow-hash udp4 sdfn
ethtool -N enp5s0f1 rx-flow-hash udp6 sdfn
```

Mivel virtuális interfészeknél ez a módszer nem működik, ezért ezt máshogyan oldottam meg a to\_napt és a to\_global interfészeknél:

```
echo "ff000000" > /sys/class/net/to_napt/queues/rx-0/rps_cpus
echo "00ff0000" > /sys/class/net/to_global/queues/rx-0/rps_cpus
```

Ezzel azt értük el, hogy az enp5s0f0 és az enp5s0f1 megszakításai a 0-15 magokon történnek, a to\_napt és a to\_global interfészeké pedig a 16-32 magokon. Ezek után a siitperf.conf fájlban állítottam be a méréshez a következő sorokat:

```
Fwd-var-sport 3
Fwd-var-dport 3
Fwd-sport-min 1
Fwd-sport-max 4000
Fwd-dport-min 1
Fwd-dport-max 1000

Stateful 1
Enumerate-ports 4
Responder-ports 3
```

Mivel előzetes mérések azt mutatták, hogy a hálózatban csomagvesztés tapasztalható, ezért a mérőszkripteket úgy módosítottam, hogy ha megérkezik a csomagok 99,99%-a, akkor már fogadja el a mérést. Továbbá minden mérés elején a CE eszközön egy set-iptables nevű szkripttel beállítottam az állapottábla méretét és az UDP csomagok timeout értékét, a mérések végén pedig töröltem, és kiírtam az állapottábla tartalmát (ellenőrzésképpen). Ezt az alábbi módon valósítottam meg:

- 1) /sbin/ip netns exec napt /usr/sbin/conntrack -F
- 2) /sbin/ip netns exec napt /usr/sbin/conntrack -L

Így mértem a MAP-T skálázódását a BR CPU magjainak függvényében 1, 2, 4, 8, 16 maggal. Ezt a BR-en a /etc/default/grub fájlban a maxcpus=X módosításával változtattam. Az eredményeket a 8. táblázatban összefoglaltam.

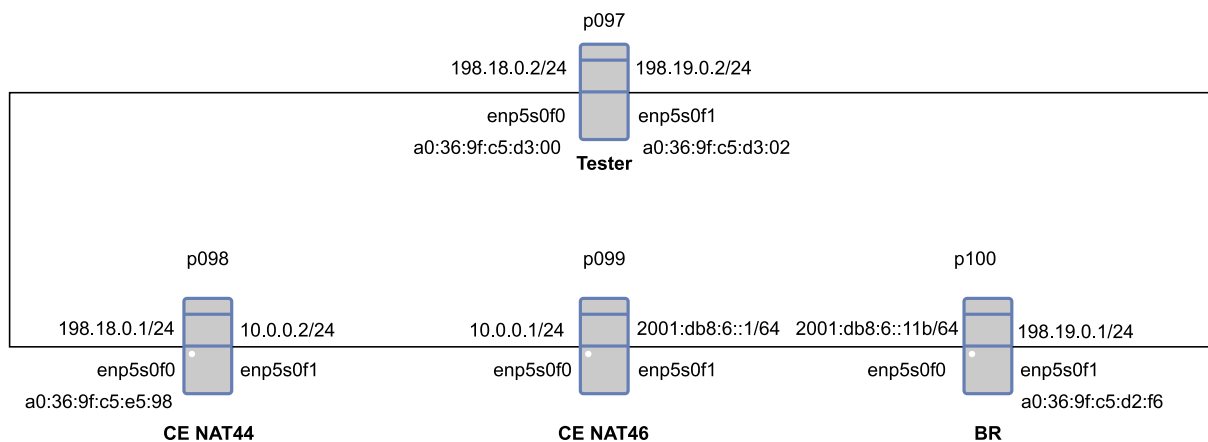
| num. cores        | 1               | 2               | 4                | 8                | 16               |
|-------------------|-----------------|-----------------|------------------|------------------|------------------|
| src ports         | 4000            | 4000            | 4000             | 4000             | 4000             |
| dst ports         | 1000            | 1000            | 1000             | 1000             | 1000             |
| num. exp.         | 20              | 20              | 20               | 20               | 20               |
| error             | 1               | 1               | 1                | 1                | 1                |
| cps median        | <b>443388,0</b> | <b>693165,0</b> | <b>1054225,0</b> | <b>1050862,0</b> | <b>1053726,0</b> |
| cps min           | 439452,0        | 681249,0        | 1007805,0        | 1023418,0        | 999022,0         |
| cps max           | 445328,0        | 694922,0        | 1125001,0        | 1125065,0        | 1125123,0        |
| cps rel. scale up | 1               | 0,782           | 0,594            | 0,296            | 0,149            |
| throughput median | <b>217110,0</b> | <b>375410,0</b> | <b>672731,0</b>  | <b>1040534,0</b> | <b>1040995,0</b> |
| throughput min    | 216403,0        | 337499,0        | 671377,0         | 1039037,0        | 1039061,0        |
| throughput max    | 217492,0        | 375668,0        | 673829,0         | 1041049,0        | 1042262,0        |
| tp. rel. scale up | 1               | 0,865           | 0,775            | 0,599            | 0,300            |

8. táblázat: MAP-T skálázódása CPU magok számának függvényében

A táblázatból látható, hogy mind a maximális kapcsolatfelépítési sebesség, mind a throughput viszonylag jól skálázódik 4 CPU magig. Viszont a kapcsolatfelépítési sebesség 4 mag fölött, a throughput pedig 8 mag fölött már egyáltalán nem növekszik. A jelenség egyik elképzelhető oka lehet az, hogy a CE gépen két névteret használtam, amelyet virtuális Ethernet interfészek kötnek össze. A csomagot továbbítása így a CE gépnek dupla munkát okoz: minden egyes csomagot kétszer kell vennie, ami a megszakítások miatt túlságosan nagy terhelést okozhat. Ennek a hipotézisnek az ellenőrzésére a mérést úgy is elvégzem, hogy a CE gép két funkcióját (állapottartó NAT44 és állapotmentes NAT46) két külön szerverrel valószínűsítem meg.

### 3.12 MAP-T konfigurálása négy szerverrel

A négy szerveres MAP-T mérési összeállítást a 19. ábra mutatja be.



19. ábra: MAP-T mérési összeállítás 4 szerverrel

A konfiguráció hasonló, mint a három szerveres megoldásnál, csak itt a CE funkcióit (állapottartó NAT44, állapotmentes NAT46) két külön gépre osztottam. A Tester beállításait a `siitperf.conf` fájlban végeztem, baloldalon a szokásos 192.18.0.2/24-es, jobboldalon a 198.19.0.2/24-es címet adtam az interfészeknek. Ezen kívül a Tester és a DUT MAC címeit is beállítottam ebben a konfigurációs fájlban, az alábbi módon:

```
IPv4-L-Real 198.18.0.2
IPv4-R-Real 198.19.0.2
MAC-L-Tester a0:36:9f:c5:d3:00 # p097 enp5s0f0
MAC-R-Tester a0:36:9f:c5:d3:02 # p097 enp5s0f1
MAC-L-DUT a0:36:9f:c5:e5:98 # p098 enp5s0f0
MAC-R-DUT a0:36:9f:c5:d2:f6 # p100 enp5s0f1
```

A CE NAT44 IP címeit a `/etc/network/interfaces` fájlban konfiguráltam, a Tester felőli kapta a 198.18.0.1/24-es címet, a NAT46 felőli pedig a 10.0.0.2/24-es IP címet. Ezek után beállítottam a NAT-olást `iptables` segítségével, majd az útvonalat a 198.19.0.0/24-es hálózatba, és engedélyeztem az IPv4-es csomagtovábbítást. Mivel a mérőprogram nem képes az ARP kérésekre válaszolni, ezért hozzáadtam az ARP táblához a Tester IP-címének feloldását:

- 1) `iptables -t nat -A POSTROUTING -s 198.18.0.0/24 -o enp5s0f1 -p udp -j SNAT --to-source 192.0.2.35:24576-32767`
- 2) `ip route add 198.19.0.0/24 via 10.0.0.1`
- 3) `sysctl -w net.ipv4.conf.all.forwarding=1`

```
4) ip neighbor add 198.18.0.2 lladdr a0:36:9f:c5:d3:00 dev enp5s0f0 nud
   permanent
```

A CE NAT46 IP címeit is a `/etc/network/interfaces` fájlban konfiguráltam, a NAT44 felőli kapta a 10.0.0.1/24-es címet, a BR felőli pedig a 2001:db8:6::1/64-es címet. Itt hozzáadtam a 64:ff9b::/96-os WKP-t és a korábban kiszámított IPv4-es címet az útválasztó táblához, majd konfiguráltam a Joolt:

```
1) ip route add 64:ff9b::/96 via 2001:db8:6::1
2) ip route add 192.0.2.35/32 via 10.0.0.2
3) modprobe jool_mapt
4) jool_mapt instance add "CE 11b" --netfilter --dmr 64:ff9b::/96
5) jool_mapt -i "CE 11b" global update end-user-ipv6-prefix
   2001:db8:ce:11b::/64
6) jool_mapt -i "CE 11b" global update bmr 2001:db8:ce::/53 192.0.2.0/24
   11 0
7) jool_mapt -i "CE 11b" global update map-t-type CE
```

Az 1) és 2) paranccsal hozzáadom a hálózatokat az útválasztó táblához. A 3) sorral a Jool MAP-T modulját töltöm be. A 4) -es egy „CE 11b” nevű instance-ot hozok létre, az alapértelmezett leképzési szabállyal. Az 5) pontban hozzáadom a Jool instance-hoz a végfelhasználói prefixet. A 6) sorban megadom az alapvető leképzési szabályt a végén 0 értékkel, ami azt jelenti, hogy az első port set magában foglalja az ún. well-known portokat is. A 7) pontban pedig beállítom az eszköz típusát.

Ezek után a BR beállításai következtek. Itt is a szokott módon a `/etc/network/interfaces` fájlban beállítottam az IP címeket. A NAT46 felőli kapta a 2001:db8:6::11b/64-es címet, a Tester felőli pedig a 198.19.0.1/24-es címet. Hozzáadtam a BMR-t az útválasztó táblához, majd betöltöttem és beállítottam a Joolt, és engedélyeztem mind az IPv4-es és IPv6-os csomagtovábbítást. Még hozzáadtam az ARP táblához a Tester IP és MAC címeit:

```
1) ip route add 2001:db8:ce:11b::/64 via 2001:db8:6::11b
2) modprobe jool_mapt
3) jool_mapt instance add "BR" --netfilter --dmr 64:ff9b::/96
4) jool_mapt -i "BR" fmrt add 2001:db8:ce::/53 192.0.2.0/24 11 0
5) jool_mapt -i "BR" global update map-t-type BR
6) sysctl -w net.ipv4.conf.all.forwarding=1
   sysctl -w net.ipv6.conf.all.forwarding=1
7) ip neighbor add 198.19.0.2 lladdr a0:36:9f:c5:d3:02 dev enp5s0f1
   nud permanent
```

Minden szervernél alkalmaztam a következő sor megfelelő verzióját, hogy mérés közben a hálózati interfészek megszakításai szét legyenek osztva a magok között:

```
ethtool -N enp5s0f{0,1} rx-flow-hash udp{4,6} sdfn
```

### **3.13 MAP-T mérése négy szerverrel**

Itt is kétféle mérést végeztem. Elsőként a MAP-T technológia teljesítőképességét vizsgáltam a BR CPU magjai számának függvényében, majd fixen 16 CPU maggal megmértem, hogyan alakul a teljesítőképesség a kapcsolatszám változásának függvényében.

Minden méréshez szükséges a CE-n egy szkript, ami a kapcsolatkövető tábla törlésére szolgál, és a mérőszkript minden egyes mérés végén lefuttatja:

- 1) /usr/sbin/conntrack -F
- 2) /usr/sbin/conntrack -L

Az első sorral törlöm a tábla tartalmát, a második sorral pedig kiíratom ellenőrzésképp. Ezen kívül szükséges egy szkript, ami minden mérés elején beállítja az UDP csomagok időtúllépését és a kapcsolatkövető tábla méretét. A magszámot a /etc/default/grub fájl maxcpus=X sorával változtattam.

Ezután a siitperf.conf fájlban beállítottam a megfelelő portszám tartományt, és elvégeztem a méréseket. Az eredmények a 9. táblázatban láthatók. Az eredményekben (és a skálázódásban is) látható ugyan kismértékű javulás a három szerveres összeállítás 8. táblázatban szereplő eredményeihez képest, de ez nem átütő jellegű: nem emeli a plafont a duplájára (amint azt korábban elképzelhetőnek tartottam).

|                   |                 |                 |                  |                  |                  |
|-------------------|-----------------|-----------------|------------------|------------------|------------------|
| num. cores        | 1               | 2               | 4                | 8                | 16               |
| src ports         | 4000            | 4000            | 4000             | 4000             | 4000             |
| dst ports         | 1000            | 1000            | 1000             | 1000             | 1000             |
| num. exp.         | 20              | 20              | 20               | 20               | 20               |
| error             | 1               | 1               | 1                | 1                | 1                |
| cps median        | <b>458581,5</b> | <b>767220,5</b> | <b>1187629,0</b> | <b>1198302,0</b> | <b>1196114,0</b> |
| cps min           | 451170,0        | 749999,0        | 1049999,0        | 1114940,0        | 1111803,0        |
| cps max           | 459526,0        | 768770,0        | 1235938,0        | 1246961,0        | 1242244,0        |
| cps rel. scale up | 1               | 0,837           | 0,647            | 0,327            | 0,163            |
| throughput median | <b>223631,0</b> | <b>364183,0</b> | <b>651747,5</b>  | <b>998264,5</b>  | <b>1204255,0</b> |
| throughput min    | 222655,0        | 360350,0        | 650975,0         | 996303,0         | 1203824,0        |
| throughput max    | 224219,0        | 365251,0        | 652735,0         | 999005,0         | 1204559,0        |
| tp. rel. scale up | 1               | 0,814           | 0,729            | 0,558            | 0,337            |

9. táblázat: MAP-T 4 szerver CPU skálázódás eredményei

|                    |                    |                  |                  |                  |                  |
|--------------------|--------------------|------------------|------------------|------------------|------------------|
| num. cores         | 16                 | 16               | 16               | 16               | 16               |
| src ports          | 4 000              | 4 000            | 4 000            | 4 000            | 4 000            |
| dst ports          | 250                | 500              | 1 000            | 2 000            | 4 000            |
| num. conn.         | 1 000 000          | 2 000 000        | 4 000 000        | 8 000 000        | 16 000 000       |
| initial R rate     | 500 000            | 700 000          | 1 000 000        | 1 500 000        | 1 500 000        |
| calculated timeout | 5,000              | 5,857            | 7,000            | 8,333            | 13,667           |
| used timeout       | 300                | 300              | 300              | 300              | 300              |
| num. exp.          | 20                 | 20               | 20               | 20               | 20               |
| error              | 1                  | 1                | 1                | 1                | 1                |
| cps median         | <b>534 680,0</b>   | <b>754 825,5</b> | <b>1 150 865</b> | <b>1 822 311</b> | <b>2 004 725</b> |
| cps min            | 496 874,0          | 687 499          | 1 099 217        | 1 756 300        | 2 003 669        |
| cps max            | 575 002,0          | 812 501          | 1 225 091        | 1 876 489        | 2 005 737        |
| throughput median  | <b>1 025 310,0</b> | <b>1 025 630</b> | <b>1 025 702</b> | <b>1 025 689</b> | <b>1 025 801</b> |
| throughput min     | 1 024 500,0        | 1 024 827        | 1 025 327        | 1 025 118        | 1 025 458        |
| throughput max     | 1 025 861,0        | 1 026 099        | 1 026 099        | 1 026 123        | 1 026 233        |
| num. cores         | 16                 | 16               | 16               | 16               |                  |
| src ports          | 4 000              | 4 000            | 4 000            | 4 000            |                  |
| dst ports          | 8 000              | 16 000           | 32 000           | 64 000           |                  |
| num. conn.         | 32 000 000         | 64 000 000       | 128 000 000      | 256 000 000      |                  |
| initial R rate     | 1 800 000          | 1 800 000        | 1 800 000        | 1 800 000        |                  |
| calculated timeout | 20,778             | 38,556           | 74,111           | 145,222          |                  |
| used timeout       | 300                | 300              | 300              | 300              |                  |
| num. exp.          | 20                 | 20               | 20               | 5                |                  |
| error              | 1                  | 1                | 1                | 1                |                  |
| cps median         | <b>2 005 831</b>   | <b>2 005 818</b> | <b>2 006 344</b> | <b>2 006 614</b> |                  |
| cps min            | 2 004 908          | 2 005 003        | 2 005 735        | 2 006 458        |                  |
| cps max            | 2 006 841          | 2 006 491        | 2 006 881        | 2 007 568        |                  |
| throughput median  | <b>1 026 024</b>   | <b>1 025 737</b> | <b>1 025 818</b> | <b>1 025 887</b> |                  |
| throughput min     | 1 025 680          | 1 019 530        | 1 025 668        | 1 025 599        |                  |
| throughput max     | 1 026 270          | 1 026 123        | 1 026 129        | 1 026 233        |                  |

10. táblázat: MAP-T kapcsolatszám skálázódás eredményei

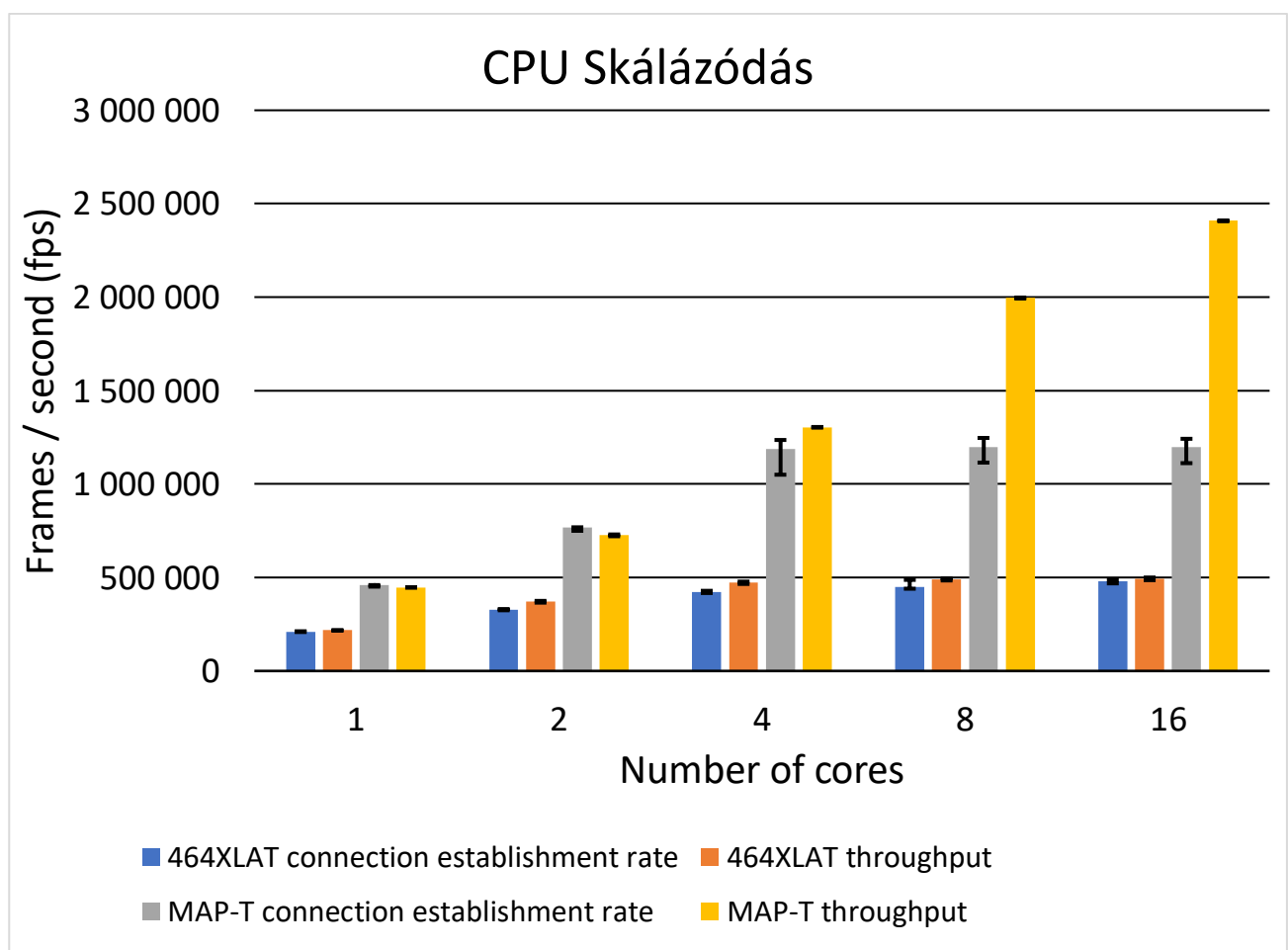
A kapcsolatszám skálázódás mérésnél látható, hogy a maximális kapcsolatépítési sebesség (cps median) növekszik 16 millió kapcsolatig, majd stagnál. Véleményem szerint ez a négy kilencessel történő mérés miatt van, a kapcsolatszám növelésével együtt emelkedik az elveszthető keretek száma így még 1 millió kapcsolatnál 100 keret veszhet el, 128 milliónál 12 800.

Az viszont egyértelmű, hogy a maximális kapcsolatfelépítési sebesség 16 millió kapcsolat fölött, és a throughput az teljes tartományban gyakorlatilag állandó, tehát a kapcsolatok számának növekedésével egyáltalán nem mutat romlást.

### 3.14 Eredmények összehasonlítása

A méréseim során mindkét technológia esetén megvizsgáltam, hogy a teljesítményük hogyan skálázódik a CPU magok számának, illetve a kapcsolatok számának függvényében. Az alábbiakban ezeket hasonlítom össze.

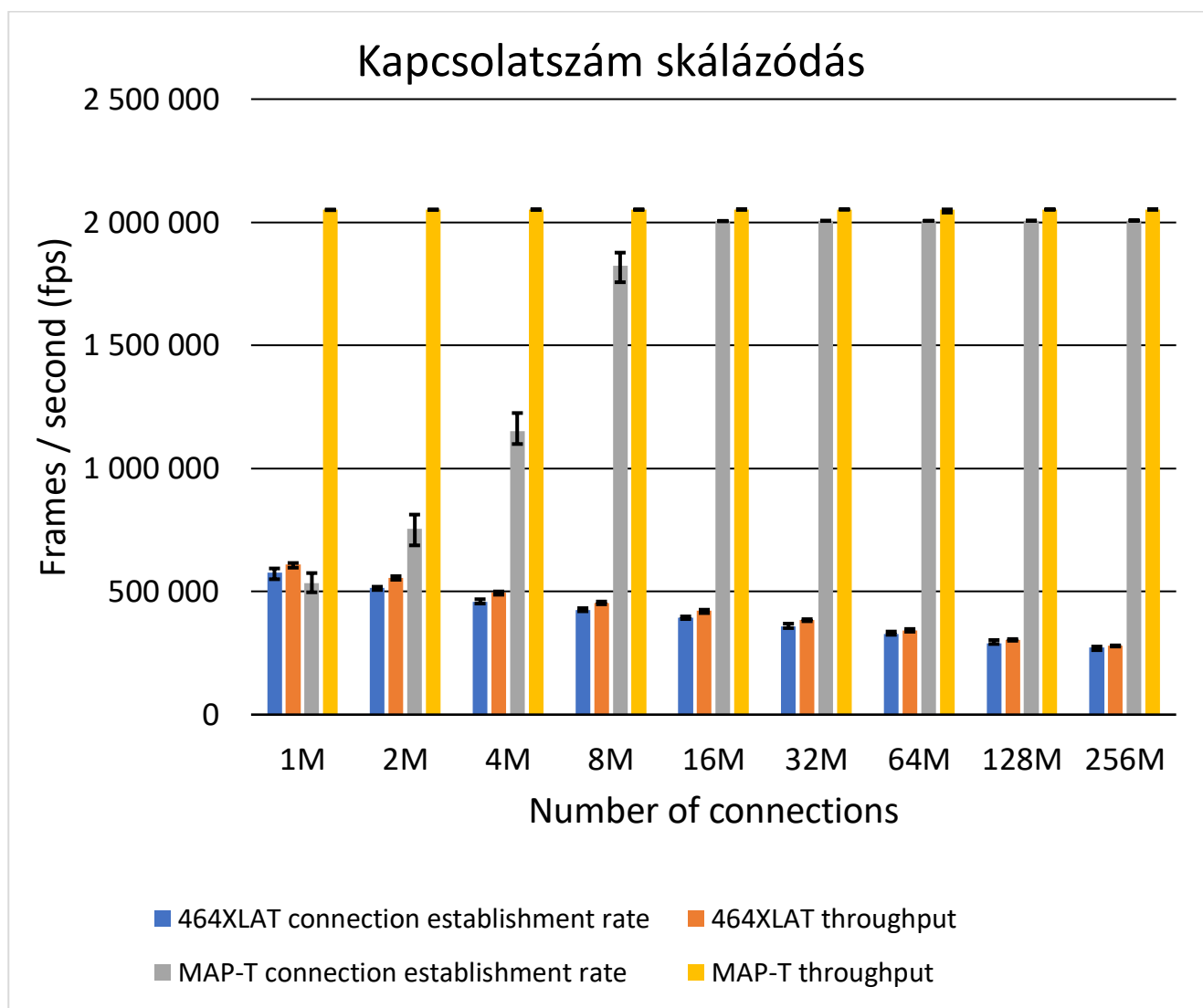
#### 3.14.1 CPU skálázódás



20. ábra: CPU skálázódás diagram

A 20. ábra mutatja az aktív CPU magok száma szerinti skálázódást oszlop diagramon ábrázolva. A kék oszlopok mutatják a 464XLAT maximális kapcsolatfelépítési (connection establishment rate) sebességét, a narancssárgák a 464XLAT áteresztőképességét (throughput), a szürkék a MAP-T maximális kapcsolatépítési sebességét, a citromsárgák pedig a MAP-T áteresztőképességét. A diagramon látható, hogy a 464XLAT 4 CPU magig skálázódik, utána már csak minimális növekedés tapasztalható az eredményekben. MAP-T megoldásnál, viszont a maximális kapcsolatépítési sebesség 4 processzormagig skálázódik, utána stagnál, az áteresztőképesség pedig egészen 16 magig növekszik. Az is jól észrevehető, hogy a MAP-T technológia minden processzormagon magasabb teljesítmény mutat a 464XLAT-nál. Ez a különbség a 464XLAT azon tulajdonságából adódik, hogy a szolgáltatói hálózatban található egy állapottartó NAT64, ami erőforrásigényesebb, mint a MAP-T-ben található állapotmentes megoldás.

### 3.14.2 Kapcsolatszám skálázódás



21. ábra: Kapcsolatszám skálázódás diagram

A 21. ábra mutatja oszlop diagramban ábrázolva a kapcsolatszám skálázódás mérés eredményeit. A kék oszlopok mutatják a 464XLAT maximális kapcsolatépítési sebességét, a narancssárgák a 464XLAT áteresztőképességét, a szürkék a MAP-T kapcsolatépítési sebességét, a citromsárgák pedig a MAP-T áteresztőképességét. 464XLAT-nál a kapcsolatszám növekedésével csökkenő eredmény tapasztalható. MAP-T-nél 1 milliótól 8 millió kapcsolatig látható bizonyos skálázódás, azonban ez, ahogy a 3.12-es fejezetben is említettem, a négy kilenccsel történő mérésből adódik. Innentől felfelé az látható a maximális kapcsolatépítési sebességen (szürke oszlop) és az áteresztőképességen (citromsárga oszlop), hogy ezek nem függnek a kapcsolatszámtól.

### **3.15 Összegzés**

Összesítve elmondható, hogy már egy CPU mag esetén is magasabb értékről indul, és a processzormag számmal jobban skálázódik a MAP-T technológia teljesítménye. A kapcsolatszám növekedésével a 464XLAT teljesítménye romlik, a MAP-T teljesítménye pedig (a négy kilences okozta anomáliát leszámítva) nem változik. Így egyértelműen megállapítható, hogy a MAP-T teljesítménye jobb. Amiben a 464XLAT lehet jobb, az a címmegosztás képessége, és ezzel együtt a portszámok jobb kihasználása. Megvalósíthatóság tekintetében is a 464XLAT-ot volt egyszerűbb felépíteni a rendelkezésre álló implementációból.

## Irodalomjegyzék

- [1] Number Resource Organization, „Free Pool of IPv4 Address Space Depleted” <https://www.nro.net/ipv4-free-pool-depleted/>
- [2] G. Lencse, J. Palet Martinez, L. Howard, R. Patterson, I. Farrer, “Pros and Cons of IPv6 Transition Technologies for IPv4aaS”, Internet Draft,  
URL: <https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-transition-comparison>
- [3] M. Mawatari, M. Kawashima, C. Byrne, „464XLAT: Combination of Stateful and Stateless Translation”, 2013. április, IETF RFC 6877, DOI: 10.17487/RFC6877
- [4] O. Troan, W. Dec, X. Li, C. Bao, S. Matsushima, T. Murakami, T. Taylor „Mapping of Address and Port with Encapsulation (MAP-E)”, 2015. július, IETF RFC 7597, DOI: 10.17487/RFC7597
- [5] A. Durand, R. Droms, J. Woodyatt, Y. Lee „Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion” 2011. augusztus, IETF RFC 6333, DOI: 10.17487/RFC6333
- [6] Nathan Lutchansky „Tayga”, 2010. URL: <http://www.litech.org/tayga/>
- [7] Jool weboldala, URL: <https://www.jool.mx/en/intro-jool.html>
- [8] ClatD, URL: <https://github.com/toreanderson/clatd>
- [9] SNABB, URL: <https://github.com/Igalia/snabb>
- [10] Pyswmap, URL: <https://github.com/ejordangottlieb/pyswmap>
- [11] OpenWRT, URL: <https://openwrt.org/>
- [12] 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
- [13] M. Georgescu, L. Pislaru, G. Lencse „Benchmarking Methodology for IPv6 Transition Technologies”, IETF RFC 8219, 2017. augusztus, DOI: 10.17487/RFC8219
- [14] FortiClient, URL: <https://www.fortinet.com/products/endpoint-security/forticlient>
- [15] Benchmarking Methodology for Stateful NATxy Gateways using RFC 4814 Pseudorandom Port Numbers, URL: <https://datatracker.ietf.org/doc/html/draft-lencse-bmwg-benchmarking-stateful-03>
- [16] G. Lencse „Design and Implementation of a Software Tester for Benchmarking Stateful NATxy Gateways: Theory and Practice of Extending Siitperf for Stateful Tests” under second review in *Computer Communications*, vol. 172, no. 1, pp. 75-88, August 1, 2022, DOI: 10.1016/j.comcom.2022.05.028

# Függelék

## A függelék: Minta siitperf.conf

```
IP-L-Vers 6 # Left Sender's IP version for foreground traffic
IP-R-Vers 4 # Right Sender's IP version for foreground traffic

IPv6-L-Real 2001:2:0:0::2
IPv6-L-Virt :: # currently not used
IPv6-R-Real 2001:2:0:8000::2
IPv6-R-Virt 2001:2:0:1000::2
IPv4-L-Real 0.0.0.0 # currently unused
IPv4-L-Virt 198.18.0.2
IPv4-R-Real 198.19.0.2
IPv4-R-Virt 0.0.0.0 # currently unused

MAC-L-Tester a0:36:9f:c5:fa:1c # StarBED p094 enp5s0f0
MAC-R-Tester a0:36:9f:c5:fa:1e # StarBED p094 enp5s0f1
MAC-L-DUT a0:36:9f:c5:e6:58 # StarBED p095 enp5s0f0
MAC-R-DUT a0:36:9f:c5:e6:5a # StarBED p095 enp5s0f1

Forward 1 # left to right direction is active
Reverse 1 # right to left direction is active
Promisc 0 # no promiscuous mode for receiving

Num-L-Nets 1 # Use only a single src. and dst. address pair
Num-R-Nets 1 # Max. 256 destination networks are supported

CPU-L-Send 2 # Left Sender runs on this core
CPU-R-Recv 4 # Right Receiver runs on this core
CPU-R-Send 6 # Right Sender runs on this core
CPU-L-Recv 8 # Left Receiver runs on this core

MEM-Channels 2 # Number of Memory Channels

# parameters for RFC 4814 random port feature
Fwd-var-sport 3 # Does source port vary? 0: fix, 1: increase, 2: decrease, 3:
random
Fwd-var-dport 3 # Does destination port vary? 0: fix, 1: increase, 2: decrease,
3: random
Fwd-sport-min 1024
Fwd-sport-max 65535
Fwd-dport-min 1
Fwd-dport-max 49151
Rev-var-sport 3 # Does source port vary? 0: fix, 1: increase, 2: decrease, 3:
random
Rev-var-dport 3 # Does destination port vary? 0: fix, 1: increase, 2: decrease,
3: random
Rev-sport-min 1024
Rev-sport-max 65535
Rev-dport-min 1
Rev-dport-max 49151

# parameters for stateful tests
Stateful 1 # Initiator is on the left side, Responder is on the right side
Enumerate-ports 0 # no port enumeration
Responder-ports 3 # Responder chooses 4-tuples randomly
```

## B függelék: Mérőszkriptek:

### binary-rate-alg.sh-R

```
#!/bin/bash
#Parameters
impl="p094-p095-fl6c-hcp4-ipv4-Sf1-Ep4-40000x20000-30-810-R" # name of the tested implementation
(used for logging)
max=2000000 # maximum packet rate
N=800000000 # number of preliminary frames
M=800000000 # state table size
T=500 # global timeout for preliminary phase
D=0 # delay caused by the preliminary phase (will be computed)
DUT_IP=172.16.28.95
startcmd="ssh ${DUT_IP} /root/DUT-settings/set-iptables 30 810"
stopcmd="ssh ${DUT_IP} /root/DUT-settings/del-iptables"
countcmd="ssh ${DUT_IP} cat /proc/sys/net/netfilter/nf_conntrack_count"
dir="b" # valid values: b,f,r; b: bidirectional, f: forward (Left to Right, 6 --> 4), r: reverse
(Right to Left, 6 <-- 4)
fs=84 # IPv6 frame size; IPv4 frame size is always 20 bytes less
xpts=60 # duration (in seconds) of an experiment instance
to=2000 # timeout in milliseconds
n=2 # foreground traffic, if ( frame_counter % n < m )
m=2 # E.g. n=m=2 is all foreground traffic; n=2,m=0 is all background traffic; n=10,m=9 is 90% fg
and 10% bg
sleep=10 # sleeping time between the experiments
e=1000 # measurement error: the difference between the values of the higher and the lower bound of
the binary search, when finishing
no_exp=5 # number of experiments
res_dir="results" # base directory for the results (they will be copied there at the end)

#####

# Generate log/csv file headers
echo "#####" > ratetest.log
echo "Tested Implementation: $impl" >> ratetest.log
echo "N: $N" >> ratetest.log
echo "M: $M" >> ratetest.log
echo "T: $T" >> ratetest.log
echo "D: $D" >> ratetest.log
echo "Frame size: $fs:" >> ratetest.log
# echo "Direction $dir:" >> ratetest.log
echo "Value of n: $n" >> ratetest.log
echo "Value of m: $m" >> ratetest.log
# echo "Duration (sec): $xpts" >> ratetest.log
echo "Starting R rate (QPS): $((max/2))" >> ratetest.log
# echo "Timeout value (sec): $to" >> ratetest.log
echo "Measurement Error: $e" >> ratetest.log
echo "Sleep Time (sec): $sleep" >> ratetest.log
date +"Date&Time: %Y-%m-%d %H:%M:%S.%N" >> ratetest.log
echo "#####" >> ratetest.log
echo "No, Size, n, m, Initial R Rate, N, M, T, D, Error, Date, Iterations needed, rate" > rate.csv

# Execute $no_exp number of experiments
for (( No=1; No <= $no_exp; No++ ))
do
    # Execute a binary search in the [l, h] interval
    l=0 #Initialize lower bound to 0
    h=$max #Initialize higher bound with maximum query rate
    for (( i=1; $((h-l)) > $e; i++ ))
    do
        # Log some information about this step
        echo "-----" >> ratetest.log
        date +"%Y-%m-%d %H:%M:%S.%N Iteration no. '$(No)'"-'$i'" >> ratetest.log
        echo "-----" >> ratetest.log
        echo "Starting iptables on DUT with command: $startcmd"
        $startcmd
        r=$((h+l)/2) # initialize the test rate with (h+l)/2
        echo "Testing R rate: $r fps."
        echo "Testing R rate: $r fps." >> ratetest.log
        D=$((1000*N/r+2*T))
        echo "Command line is: ./build/siitperf-tp $fs 1 1 $to $n $m $N $M $r $T $D"
```

```

# Execute the test program
./build/siitperf-tp $fs 1 1 $to $n $m $N $M $r $T $D > temp.out 2>&1
# Log and print out info
cat temp.out >> ratetest.log
cat temp.out | tail
echo "Right after the test DUT's nf_conntrack_count is:" $($countcmd)
echo "Stopping iptables on DUT with command: $stopcmd"
$stopcmd
echo "DUT's nf_conntrack_count was cleared to:" $($countcmd)
# Check for any errors
error=$(grep 'Error:' temp.out)
if [ -n "$error" ]; then
    failedtofill=$(grep 'Failed to fill state table' temp.out)
    if [ -n "$failedtofill" ]; then
        echo "Failed to fill state table: therefore, this iteration is failed"
        echo "Failed to fill state table: therefore, this iteration is
failed" >> ratetest.log
    else
        echo "Error occurred, testing must stop."
        exit -1;
    fi
fi
# Collect and evaluate the results
fwd_rec=$(grep 'Preliminary frames received:' temp.out | awk '{print $4}')
echo Forward: $fwd_rec frames were received from the required $N frames
echo Forward: $fwd_rec frames were received from the required $N frames >>
ratetest.log
if [ $fwd_rec -eq $N ]; then
    l=$r
    echo TEST PASSED
    echo TEST PASSED >> ratetest.log
# Otherwise we choose the lower half interval
else
    h=$r
    echo TEST FAILED
    echo TEST FAILED >> ratetest.log
fi
# Some further logging (and preparing results)
echo "New diff: $((h-l))"
echo "New diff: $((h-l))" >> ratetest.log
date +%Y-%m-%d %H:%M:%S, "$i", "$r" > temprate.csv
echo "Sleeping for $sleep seconds..."
echo "Sleeping for $sleep seconds..." >> ratetest.log
# Sleep for $sleep time to give DUT a chance to relax
sleep $sleep
done # (end of the binary search)
# Collect results
summary=$(cat temprate.csv)
echo "$No, $fs, $n, $m, $((($max/2)), $N, $M, $T, $D, $e, $summary" >> rate.csv
rm temp*
done # (end of the $no_exp number of experiments)
# Save the results (may be commented out if not needed)
dirname="$res_dir/$(date +%simpl-'$fs'-'$n'-'$m'-'$N'-'$N'-'$T'-'$D'-'$e'-%F-%H%M)"
mkdir -p $dirname
mv ratetest.log $dirname/
mv rate.csv $dirname/
mv nohup.out $dirname/
cp -a siitperf.conf $dirname/

```

## binary-rate-alg.sh-sf

```

#!/bin/bash
#Parameters
impl="p094-p095-f16c-hcp4-ipv4-Sf1-Ep4-Rp3-40000x20000-30-740-tp" # name of the tested
implementation (used for logging)
max=2000000 # maximum packet rate
N=800000000 # number of preliminary frames
M=800000000 # state table size
R=1200000 # frame rate for preliminary phase
T=500 # global timeout for preliminary phase
D=$((1000*N/R+2*T)) # delay caused by the preliminary phase
DUT_IP=172.16.28.95
startcmd="ssh ${DUT_IP} /root/DUT-settings/set-iptables 30 740"
stopcmd="ssh ${DUT_IP} /root/DUT-settings/del-iptables"

```

```

countcmd="ssh ${DUT_IP} cat /proc/sys/net/netfilter/nf_conntrack_count"
dir="b" # valid values: b,f,r; b: bidirectional, f: forward (Left to Right, 6 --> 4), r: reverse
(Right to Left, 6 <-- 4)
fs=84 # IPv6 frame size; IPv4 frame size is always 20 bytes less
xpts=60 # duration (in seconds) of an experiment instance
to=2000 # timeout in milliseconds
n=2 # foreground traffic, if ( frame_counter % n < m )
m=2 # E.g. n=m=2 is all foreground traffic; n=2,m=0 is all background traffic; n=10,m=9 is 90% fg
and 10% bg
sleep=10 # sleeping time between the experiments
e=1000 # measurement error: the difference between the values of the higher and the lower bound of
the binary search, when finishing
no_exp=10 # number of experiments
res_dir="results" # base directory for the results (they will be copied there at the end)

#####

# Generate log/csv file headers
echo "#####" > ratetest.log
echo "Tested Implementation: $impl" >> ratetest.log
echo "N: $N" >> ratetest.log
echo "M: $M" >> ratetest.log
echo "R: $R" >> ratetest.log
echo "T: $T" >> ratetest.log
echo "D: $D" >> ratetest.log
echo "Frame size: $fs:" >> ratetest.log
echo "Direction $dir:" >> ratetest.log
echo "Value of n: $n" >> ratetest.log
echo "Value of m: $m" >> ratetest.log
echo "Duration (sec): $xpts" >> ratetest.log
echo "Starting rate (QPS): $(max/2)" >> ratetest.log
echo "Timeout value (sec): $to" >> ratetest.log
echo "Measurement Error: $e" >> ratetest.log
echo "Sleep Time (sec): $sleep" >> ratetest.log
date +%Date&Time: %Y-%m-%d %H:%M:%S.%N' >> ratetest.log
echo "#####" >> ratetest.log
echo "No, Size, Dir, n, m, Duration, Initial Rate, N, M, R, T, D, Error, Date, Iterations needed,
rate" > rate.csv

# Execute $no_exp number of experiments
for (( No=1; No <= $no_exp; No++ ))
do
    # Execute a binary search in the [l, h] interval
    l=0 #Initialize lower bound to 0
    h=$max #Initialize higher bound with maximum query rate
    for (( i=1; $(h-l) > $e; i++ ))
    do
        # Log some information about this step
        echo ----- >> ratetest.log
        date +%Y-%m-%d %H:%M:%S.%N Iteration no. "${No}"-"$i" >> ratetest.log
        echo ----- >> ratetest.log
        r=$((h+l)/2) # initialize the test rate with (h+l)/2
        echo "Testing rate: $r fps."
        echo "Testing rate: $r fps." >> ratetest.log
        FailedToFill="First trial" # Protection against 'Failed to fill state table' (test is
re-executed)
        while [ -n "$FailedToFill" ]
        do
            echo "Starting iptables on DUT with command: $startcmd"
            $startcmd
            echo "Command line is: ./build/siitperf-tp $fs $r $xpts $to $n $m $N $M $R $T
$D"
            echo "Command line is: ./build/siitperf-tp $fs $r $xpts $to $n $m $N $M $R $T
$D" >> ratetest.log
            # Execute the test program
            ./build/siitperf-tp $fs $r $xpts $to $n $m $N $M $R $T $D > temp.out 2>&1
            # Log and print out info
            cat temp.out >> ratetest.log
            cat temp.out | tail
            echo "Right after the test DUT's nf_conntrack_count is: " $(($countcmd))
            echo "Right after the test DUT's nf_conntrack_count is: " $(($countcmd)) >>
ratetest.log
            echo "Stopping iptables on DUT with command: $stopcmd"
            echo "Stopping iptables on DUT with command: $stopcmd" >> ratetest.log
            $stopcmd

```

```

echo "DUT's nf_conntrack_count was cleared to:" $($countcmd)
echo "DUT's nf_conntrack_count was cleared to:" $($countcmd) >> ratetest.log
FailedToFill=$(grep 'Failed to fill state table' temp.out)
if [ -n "$FailedToFill" ]
then
    echo "Test will be re-taken due to: $FailedToFill"
    echo "Test will be re-taken due to: $FailedToFill" >> ratetest.log
fi
done
# Check for any other errors
error=$(grep 'Error:' temp.out)
if [ -n "$error" ]; then
    echo "Error occurred, testing must stop."
    exit -1;
fi
# Collect and evaluate the results (depending on the direction of the test)
if [ "$dir" == "b" ]; then
    fwd_rec=$(grep 'Forward frames received:' temp.out | awk '{print $4}')
    rev_rec=$(grep 'Reverse frames received:' temp.out | awk '{print $4}')
    echo Forward: $fwd_rec frames were received from the required $((xpts*r))
frames
frames >> ratetest.log
    echo Reverse: $rev_rec frames were received from the required $((xpts*r))
frames
frames >> ratetest.log
    echo Reverse: $rev_rec frames were received from the required $((xpts*r))
    if [ $fwd_rec -eq $((xpts*r)) ] && [ $rev_rec -eq $((xpts*r)) ]; then
        l=$r
        echo TEST PASSED
        echo TEST PASSED >> ratetest.log
        # Otherwise we choose the lower half interval
        else
            h=$r
            echo TEST FAILED
            echo TEST FAILED >> ratetest.log
        fi
    fi
    if [ "$dir" == "f" ]; then
        fwd_rec=$(grep 'Forward frames received:' temp.out | awk '{print $4}')
        echo Forward: $fwd_rec frames were received from the required $((xpts*r))
frames
frames >> ratetest.log
        echo Forward: $fwd_rec frames were received from the required $((xpts*r))
        if [ $fwd_rec -eq $((xpts*r)) ]; then
            l=$r
            echo TEST PASSED
            echo TEST PASSED >> ratetest.log
            # Otherwise we choose the lower half interval
            else
                h=$r
                echo TEST FAILED
                echo TEST FAILED >> ratetest.log
            fi
        fi
        if [ "$dir" == "r" ]; then
            rev_rec=$(grep 'Reverse frames received:' temp.out | awk '{print $4}')
            echo Reverse: $rev_rec frames were received from the required $((xpts*r))
frames
frames >> ratetest.log
            echo Reverse: $rev_rec frames were received from the required $((xpts*r))
            if [ $rev_rec -eq $((xpts*r)) ]; then
                l=$r
                echo TEST PASSED
                echo TEST PASSED >> ratetest.log
                # Otherwise we choose the lower half interval
                else
                    h=$r
                    echo TEST FAILED
                    echo TEST FAILED >> ratetest.log
                fi
            fi
        fi
    # Some further logging (and preparing results)
    echo "New diff: $((h-l))"
    echo "New diff: $((h-l))" >> ratetest.log

```

```

        date +%Y-%m-%d %H:%M:%S, "$i", "$r" > temprate.csv
        echo "Sleeping for $sleep seconds..."
        echo "Sleeping for $sleep seconds..." >> ratetest.log
        # Sleep for $sleep time to give DUT a chance to relax
        sleep $sleep
    done # (end of the binary search)
    # Collect results
    summary=$(cat temprate.csv)
    echo "$No, $fs, $dir, $n, $m, $xpts, $((($max/2)), $N, $M, $R, $T, $D, $e, $summary" >>
rate.csv
    rm temp*
done # (end of the $no_exp number of experiments)
# Save the results (may be commented out if not needed)
dirname="$res_dir/$(date +%Y-%m-%d %H:%M:%S, "$i", "$r" > temprate.csv)
"$e"-%F-%H%M)"
mkdir -p $dirname
mv ratetest.log $dirname/
mv rate.csv $dirname/
mv nohup.out $dirname/
cp -a siitperf.conf $dirname/

```

## C függelék: Beállítószkriptek

### set-iptables (3 szerveres mérések)

```
if [ $1 -lt 22 -o $1 -gt 30 ]
then echo "Bad input"; exit -1;
fi
contracksz=$(2**$1)
hashsz=$(contracksz/1)
echo "Info: Number of conntrack table entries: $contracksz, hashsize:
$hashsz."
echo "$hashsz" > /sys/module/nf_conntrack/parameters/hashsize
#/sbin/ip netns exec napt /sbin/sysctl -w
net.netfilter.nf_conntrack_max=$contracksz
/sbin/ip netns exec napt echo $contracksz >
/proc/sys/net/netfilter/nf_conntrack_max
if [ -n "$2" ]
then
echo "Info: Setting UDP timeout to: $2"
/sbin/ip netns exec napt /sbin/sysctl -w
net.netfilter.nf_conntrack_udp_timeout=$2
fi
```

### set-iptables (4 szerveres mérések)

```
if [ $1 -lt 22 -o $1 -gt 30 ]
then echo "Bad input"; exit -1;
fi
contracksz=$(2**$1)
hashsz=$(contracksz/1)
echo "Info: Number of conntrack table entries: $contracksz, hashsize:
$hashsz."
echo "$hashsz" > /sys/module/nf_conntrack/parameters/hashsize
/sbin/sysctl -w net.netfilter.nf_conntrack_max=$contracksz
echo $contracksz > /proc/sys/net/netfilter/nf_conntrack_max
if [ -n "$2" ]
then
echo "Info: Setting UDP timeout to: $2"
/sbin/sysctl -w net.netfilter.nf_conntrack_udp_timeout=$2
fi
```