

TDK dolgozat

Horváth Viktor



SZÉCHENYI ISTVÁN EGYETEM
MŰSZAKI TUDOMÁNYI KAR
TÁVKÖZLÉSI TANSZÉK



TDK DOLGOZAT

6to4 relay implementációk teljesítőképességének és stabilitásának vizsgálata

Horváth Viktor

**Villamosmérnöki BSc szak
Infokommunikáció szakirány
4. évfolyam**

témavezetők: Dr. Lencse Gábor
egyetemi docens
Répás Sándor
PHD hallgató

1. Tartalomjegyzék

1. Tartalomjegyzék.....	1
2. Bevezető.....	3
3. 6to4 áttérési technika	4
3.1. 6to4.....	4
3.2. 6to4 működése.....	5
3.3. 6to4 relay	7
4. Implementációk.....	13
4.1. Implementációk	13
4.2. Relayként nem használható implementációk	14
4.2.1. gre interfész	14
4.2.2. gif interfész	16
5. 6to4 relay implementációk terheléstesztje	18
5.1. Mérési körülmények.....	18
5.1.1. Mérési elrendezés	18
5.1.2. Hardver és szoftver konfigurációk.....	19
5.1.3. Mérési módszer.....	20
5.1.4. Responder számítógép beállításai.....	22
5.1.5. Kliens számítógépek beállításai.....	23
5.2. Relay konfigurációk	23
5.2.1. Debian 7.1.0_x86 – sit	24
5.2.2. Debian 7.1.0_x86 – v4tunnel.....	24
5.2.3. OpenWrt (Attitude Adjustment) 12.09_x86 – sit	24
5.2.4. FreeBSD 9.1_x86 – stf	26
5.2.5. NetBSD 6.1.2_x86 – stf.....	26
5.2.6. OpenBSD 5.3_x86 – gif	27
5.3. Kiértékelés.....	28
5.4. Eredmények.....	30
5.4.1. Debian 7.1.0_x86 – sit	32
5.4.2. Debian 7.1.0_x86 – v4tunnel.....	33
5.4.3. OpenWrt (Attitude Adjustment) 12.09_x86 – sit	35

5.4.4. FreeBSD 9.1_x86 – stf	36
5.4.5. NetBSD 6.1.2_x86 – stf.....	38
5.4.6. OpenBSD 5.3_x86 – gif	40
5.5. Összehasonlítás	41
5.6. Teljesítőképesség átlagos PC-n.....	45
5.7. Végeredmény.....	49
6. Routerek.....	50
6.1.1. Ubiquiti EdgeRouter™ Lite – sit.....	50
6.1.2. Ubiquiti EdgeRouter™ Lite – v4tunnel.....	52
6.1.3. RouterStation Pro – sit.....	53
6.1.4. Konklúzió	56
7. Összefoglalás	58
8. Irodalomjegyzék.....	59
9. Függelék.....	1
9.1. Mérés	1
9.2. Eredmények kiértékelése.....	1
9.3. Terhelési adatok kiértékelése	2
9.4. Mellékletek.....	3

2. Bevezető

A mai modern társadalomban egyre nagyobb szerepet kap az információtovábbítás. Mivel egyre több olyan telekommunikációs eszközünk van, amely képes az internet segítségével kommunikálni, például: okostelefon, tv, táblagép, hűtő, mosógép stb., a mérnököknek új, bővebb címtérre kellett alkotniuk, mint az IPv4 címtérre.

Az IPv4 nyújtotta címtérre a 2010-es évek elején teljes mértékben kihasználtuk. A RIPE NCC 2012.09.14-én elkezdte az utolsó /8-as IPv4 blokk kiosztását [2] [3]. Az IP címek elfogyásának megakadályozása mellett szükség volt továbbá technológiai újításokra is, melyek nagyobb biztonságot garantálnak az internetes kommunikációk során [4]. Megoldásként érkezett az internet protokoll 6-os verziója. Az IPv6 szabványt már 1998-ban kidolgozták és közzétették az RFC 2460-ban [1].

A lehetőségek adta rengeteg cím kihasználása azonban egy hosszú távú folyamat lesz. Legfőbb probléma, hogy a régebbi eszközeink nagy része nem képes IPv6 címek kezelésére. Mind a szolgáltatók, mind a felhasználók nem kevés pénzt öltek egy jól működő informatikai szolgáltató rendszerbe, hálózatba, amelyben a régebbi eszközök újakra cserélése komoly költségekkel járna. A kényszeres modernizálást természetesen nem lehet sem a szolgáltatóktól, sem a felhasználóktól elvárni. Továbbá a legmodernebb hálózati kommunikációra képes eszközök némelyike már nem fogja támogatni az internet protokoll korábbi verzióját, így az új címezés egyik napról a másikra történő bevezetése lehetetlen.

A két protokollnak tehát együtt kell élnie még jó darabig. A két protokoll közti átjárhatóságot több, úgynevezett áttérési technika teszi lehetővé. Egyik ilyen technika a 6to4, mely dolgozatom témáját képezi. A 6to4 megvalósítására több, operációs rendszertípusonként különböző úgynevezett szoftverimplementáció készült. A célom az egyes 6to4 implementációk, pontosabban 6to4 relay implementációk képességeinek kiderítése volt. A 6to4 relayt megvalósító implementációk vizsgálatához konzulenseim segítségével ICMP üzenetek feldolgozásán alapuló módszert dolgoztam ki. A mérési módszer segítségével kiderítettem a leghatékonyabb operációs rendszer-implementáció kombinációt, amely alkalmazhatósága nem kizárt nagy forgalmú, megbízhatóságot követelő csomagkapcsolt hálózati környezetben sem.

Dolgozatommal továbbá útmutatást igyekszem nyújtani rendszergazdák, szolgáltatók, hálózatüzemeltetők számára, hogy az igényeknek megfelelően a leghatékonyabb konfigurációval biztosíthassák hálózatuk minőségét.

Szeretném ezúton is köszönetemet kifejezni mindazoknak, akik türelmükkel, szabadidejük feláldozásával, baráti tanácsaikkal segítettek dolgozatom elkészítésében. Külön köszönet illeti tanáraimat, akiknek a dolgozat elkészítése során kamatoztatott tudásomat köszönhetem.

3. 6to4 áttérési technika

3.1. 6to4

A 6to4 a 3056-os RFC-ben [5] van definiálva. Minden egyes állomás, mely rendelkezik publikus, azaz nem NAT-olt IPv4 címmel, generálhat magának egyedülálló 6to4 IP címet, amely cím segítségével és egy beágyazási technikával IPv6 csomagokat továbbíthat IPv4 csomagokba ágyazva. A 6to4 mechanizmus alkalmazásával IPv6 szigeteket köthetünk össze egymással IPv4 hálózaton keresztül. Továbbá úgynevezett relay útvonalválasztó alkalmazásával a natív IPv6 tartományokat is elérhetővé tehetjük a csak IPv4-es hálózati kapcsolattal rendelkező, de IPv6-os protokoll készletet támogató eszközeink számára. Habár a mechanizmus IPv6 tartományok összekötésére jött létre, egyedülálló, úgynevezett 6to4 hostok/kliensek is kommunikálhatnak ily módon, ha rendelkeznek publikus IPv4 címmel. A 6to4 mechanizmust legtöbbször határátjáró útvonalválasztókban (border router) használják. Ezen áttérési technika kapcsán a következő fogalmakat definiálták:

- 6to4 pszeudo-interfész:

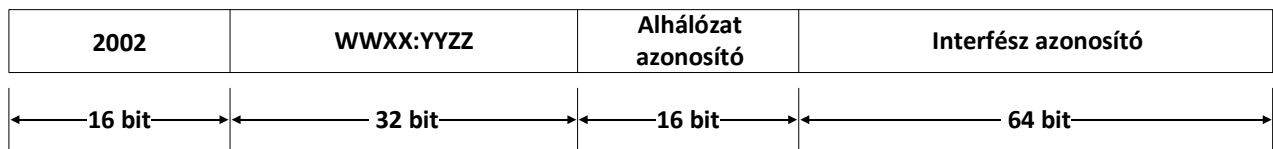
Egy logikailag IPv6-osnak tekinthető interfész, amely 6to4 beágyazás segítségével IPv6 csomagokat továbbít IPv4-es csomagokban. Ennek a logikai interfésznek a megvalósítására jöttek létre szoftverimplementációk az egyes operációs rendszerekhez. Dolgozatomban döntő részét ezeknek az implementációknak a tesztelése képezi.

- 6to4 prefix:

Mint minden kitüntetett IPv6-os címcsoport, a 6to4 is egy saját prefixet (előtagot) használ. Ez jelen esetben a 2002::/16. A 6to4 prefix alkalmazhatósága publikus IPv4 címet feltételez.

- 6to4 cím:

Egy érvényes, globálisan egyedi IPv6 cím, amely a következőképp áll elő: első 16 bit az egységes 6to4 prefix, a következő 32 bit az állomás egyedi publikus IPv4 címe hexadecimális formátumban, ezután lehetőség van 16 bitnyi alhálózat azonosító használatára, amely hasznos lehet, ha több fizikai hálózatunk van. Végül 64 bit marad a szabad felhasználású interfész azonosítóra (pl.: módosított EUI-64 algoritmust alkalmazva). Egy illusztráló példa az *1. ábrán*: 2002:WWXX:YYZZ::1/48. A 6to4 host IP cím /48-as prefixének köszönhetően minden érvényes 6to4 cím "mögött" több, akár 2^{16} darab IPv6 hálózat képezhető külön-külön 64 biten elhelyezhető különböző interfész címekkel.

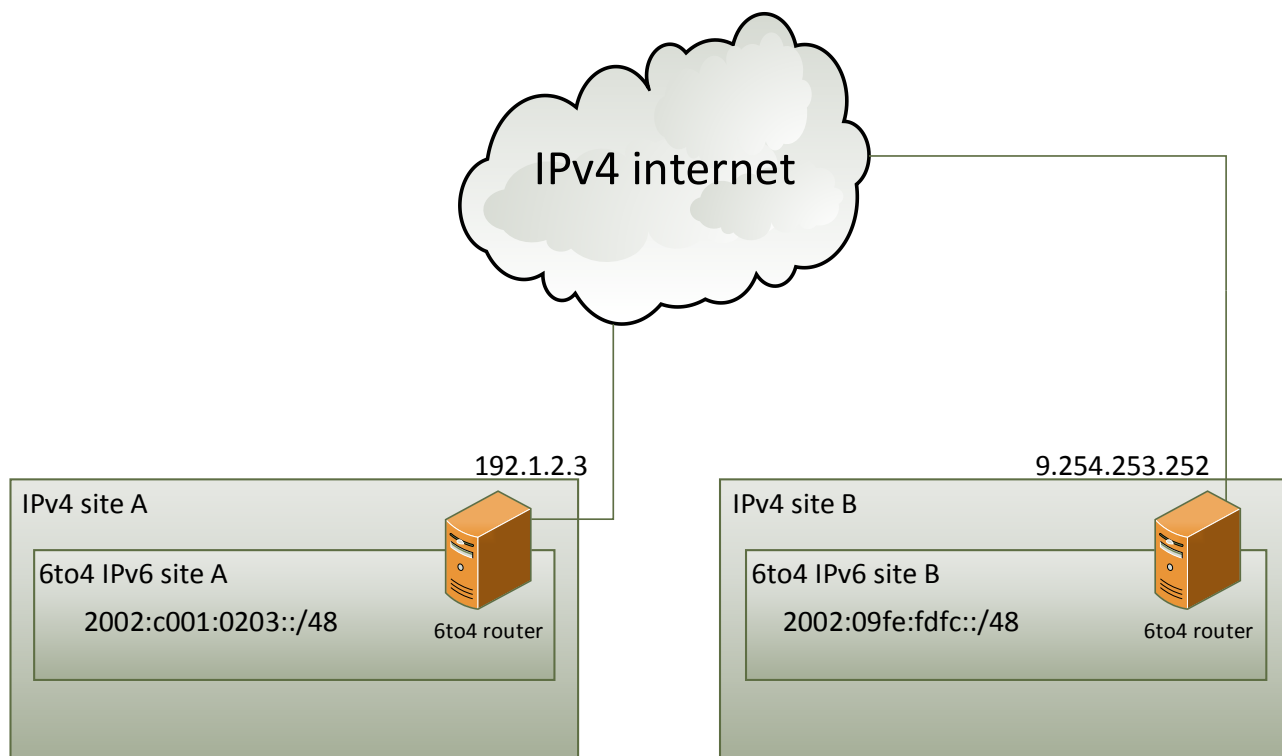


1. ábra A 6to4 IP cím szerkezete (forrás: [7]).

- natív IPv6 cím:
Egy IPv6 cím, mely nem 6to4 prefix használatával jött létre.
- 6to4 router:
Egy IPv6- és IPv4 címzést használó útvonalválasztó 6to4 pszeudo-interfész támogatással. Legtöbb esetben egy határátjáró útvonalválasztó egy IPv6-os sziget (vagy site) és a kiterjedt IPv4 (WAN) hálózat között.
- 6to4 host:
Egy IPv6 kliens, mely legalább egy 6to4 IP címmel rendelkezik.
- 6to4 site:
Egy tartomány, melyen belül 6to4 címeket használva IPv6 kommunikáció valósul meg. Legalább egy 6to4 hostot és egy 6to4 routert tartalmaz.
- 6to4 relay router:
Egy 6to4 tartomány és egy natív IPv6 tartomány közötti átjárhatóságot biztosító útvonalválasztó.
- külső 6to4 routing tartomány:
Egy routing tartomány, mely 6to4- és 6to4 relay routereket kapcsol össze. Különböző, mind a belső IPv6 routing tartományoktól, mind a natív külső (publikus) IPv6 tartományoktól.

3.2. 6to4 működése

A 6to4 mechanizmus legegyszerűbb alkalmazására egy példa látható a 2. ábrán. 6to4 hálózati kapcsolat kialakítható magánhálózaton belül is. Jelen esetben két különböző internetszolgáltatón keresztül épül fel alagút a két 6to4 site között. A csatorna kiépülésének egyetlen feltétele, hogy mindkét site képes legyen a 41-es protokollazonosítójú IPv4 csomagok fogadására és küldésére [20]. Ha egy IPv6 csomag érkezik bármelyik 6to4 routerhez, az beágyazódik IPv4 csomagba és a célállomásig vezető útvonalon végighalad, majd a fogadó router "kicsomagolja" és ismét IPv6 csomaggént továbbítja a 6to4 célhálózat felé. Látható, hogy egy 6to4 hostnak más 6to4 hostokkal való kommunikációjához elegendő egy publikus IPv4 cím és egy 6to4 pszeudo-interfész, azaz egy logikai hálózati csatoló, mely rendelkezik egy 6to4 IP címmel.



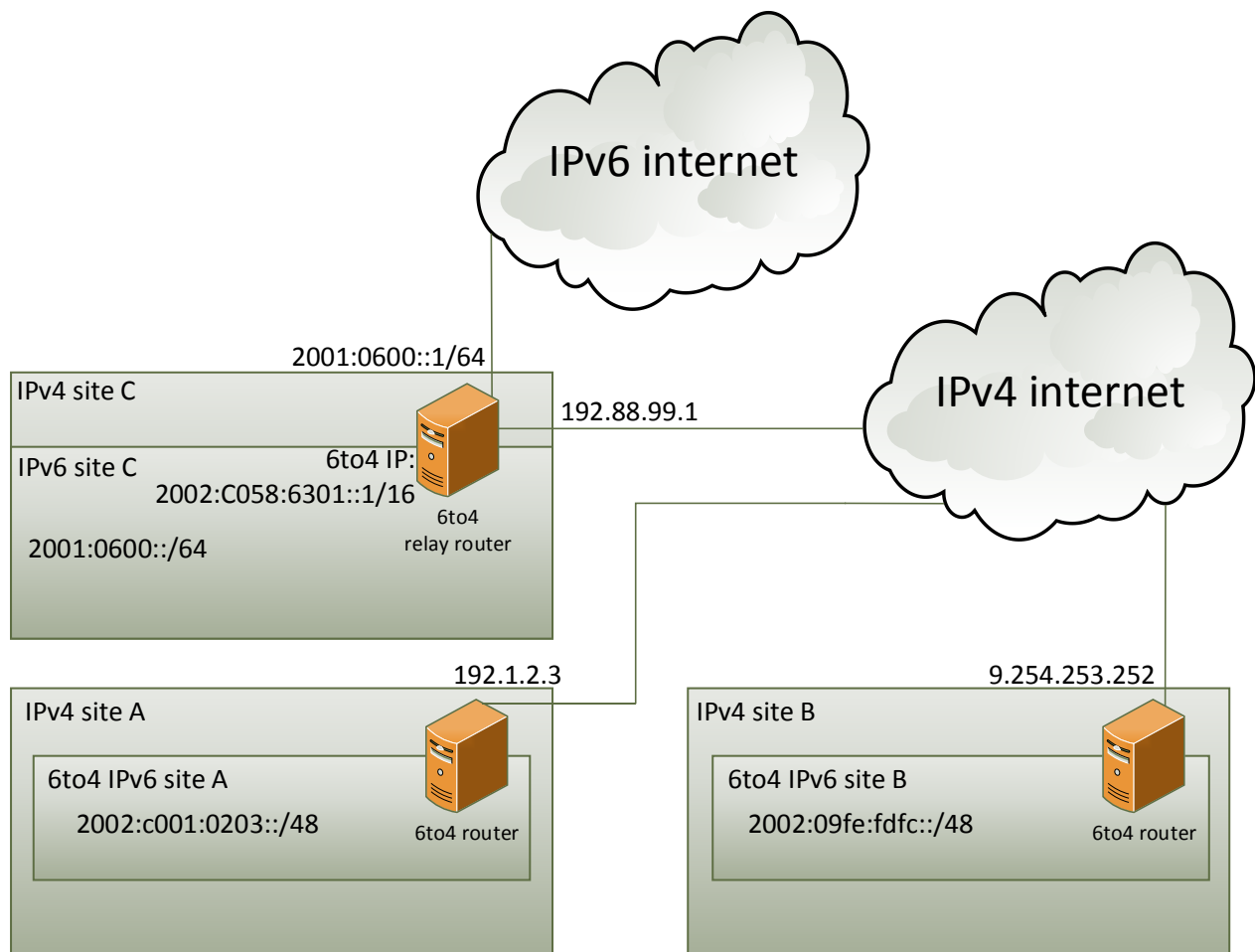
2. ábra 6to4 hálózat [5].

Ennél összetettebb eset, amikor valamelyik 6to4 site-on belüli host a natív IPv6 hálózatot is el szeretné érni: 3. ábra. Ekkor a site-on belül 6to4 relay routert kell alkalmazni, mely rendelkezik natív globális IPv6 címmel, publikus, azaz nem NAT-olt IPv4 címmel és 6to4 címmel is. Ahhoz, hogy natív IPv6 hostok is el tudják érni a 6to4 klienseinket, a relay routernek hirdetni kell egy útvonalat a natív, külső IPv6 tartományba a 2002::/16 hálózatról. Abban az esetben, ha privát 6to4 relayt üzemeltetünk, ezt az útvonalat csak a saját hálózatunkon hirdetjük. Ha a relay nyilvános, akkor a 2002::/16 címtartományról a teljes internet felé hirdetünk útvonalat. Ez a fajta kommunikáció anycast címzéssel valósul meg, amely során a kliensek a router metrikák szerinti legközelebbi 6to4 relayt keresik fel.

A B és az A site-ban lévő 6to4 hostok képesek elérni az internetet a 4-es verziójú internet protokoll segítségével a 9.254.253.252 (B) és a 192.1.2.3 (A) címen keresztül. A B site-ban lévő kliensek számára az A site 6to4 hostjai továbbra is az IPv4-es interneten keresztül, a 2002:09fe:fdfe::/48-as tartományból érhetők el. Az A site-ban lévő kliensek pedig a 2002:c001:0203::/48 hálózatból, az IPv4 interneten érik el a B site 6to4 hostjait. Ha esetleg IPv4 címtéren belül szeretne két host kommunikálni, abban az esetben nincs szükség semmilyen 6to4 útvonalválasztóra, a küldő fél automatikusan a célállomásnak küldi a csomagot IPv4-en keresztül. Ha a site A-, vagy B-ben lévő 6to4 hostok egy natív IPv6 címmel rendelkező számítógéppel (ez lehet például egy, az interneten keresztül elérhető IPv6-os szerver is) szeretnének kommunikálni, úgy azt a site C-ben elhelyezett 6to4 relay router segítségével oldhatják meg. Ebben az esetben egy 6to4 host IPv6 kérése (IPv6 csomagja) úgy jut el a célhálózatba, hogy az aktuális site 6to4 routere beágyazza

IPv4 csomagba a routeren alkalmazott 6to4 pszeudo-interfész segítségével, majd a csomag az IPv4 interneten keresztül eljut a 192.88.99.1 című 6to4 relay routerhez. A relay "kicsomagolja" a beágyazott IPv6 csomagot és gondoskodik annak továbbításáról a natív IPv6 célhálózat felé.

Ha nem áll módunkban 6to4 relayt üzemeltetni, akkor valaki más által üzemeltetett útvonalválasztó szolgáltatásait kell igénybe vennünk. A relayek felkeresésének többféle módja létezik. Egyes kliensekben beállítható a relay router IPv4 címe. Léteznek kliensek, melyekbe előre, gyárilag "beégették" a felkeresendő router IP címét [7]. Általános eset, mikor az RFC-ben [5] definiált 192.88.99.1-es címen anycast címzéssel keressük fel a relayt (nem feltétlenül a földrajzilag legközelebbit). Ez gyakran hosszú útvonalon, nagy válaszidővel valósul meg.



3. ábra Összetett 6to4 hálózat.

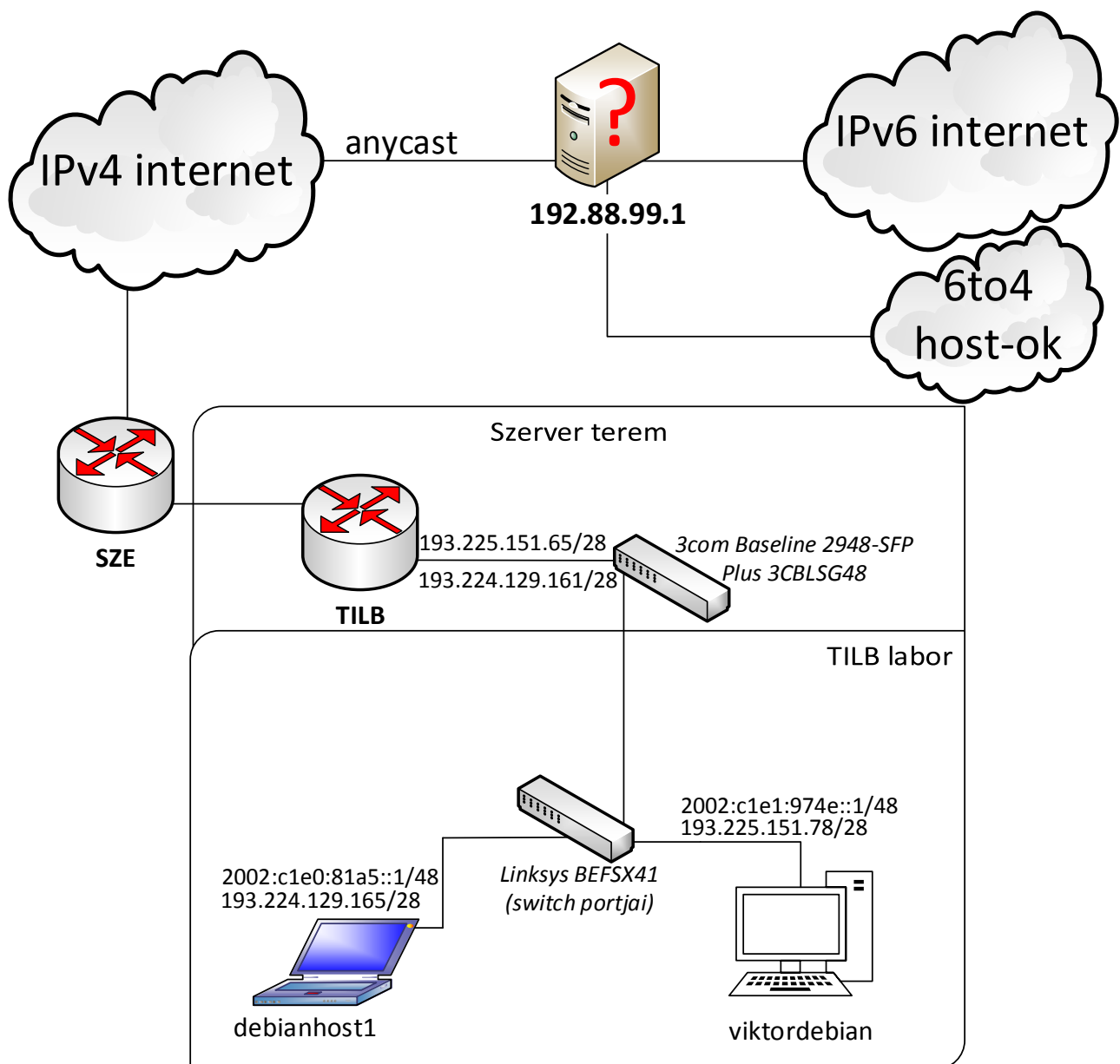
A 6to4 módszer legnagyobb hátránya, hogy hamis IPv6 címekről érkező csomagokkal elárasztható a hálózat (IP address spoofing) [8].

3.3. 6to4 relay

Ebben az alfejezetben a 6to4 relay szerepét, működését kívánom bővebben szemléltetni az olvasó számára. A laborban végzett teszt mérési összeállításomat a 4. ábra ismerteti. Az összes

implementáció tesztjét ebben az elrendezésben végeztem. A debianhost1 nevű számítógép és a viktordebian nevű gép IPv4 címe különböző tartományból kiosztott publikus IP címek, hogy az esetleges egy alhálózatba tartozás, és a közös Ethernet hálózatra csatlakozás ne befolyásolhassa a teszt eredményeit.

Az alábbi példával a 192.88.99.1 anycast címen elérhető 6to4 relay router feladatkörét igyekszem szemléltetni. A kérdőjel az útvonalválasztó földrajzi helyének és a pontos hálózati címének anonimitását szimbolizálja. Subnet ID-nek a csupa nulla értéket választottam az egyszerűség kedvéért. A kiosztott 6to4 cím mindkét számítógép esetén az első érvényes kiosztható IP cím, tehát az 1-es végződésű.



4. ábra Tesztkörnyezet.

Szemléltető konfigurációnak a Debian Linux sit- és v4tunnel interfész beállításait választottam. A debianhost1 kliens sittel, míg a viktordebian kliens v4tunnellel konfigurált. A két kliens 6to4 hostként funkcionál. Egy sittel, v4tunnellel konfigurált 6to4 környezet beállításához kiváló segítséget nyújt a [9, 10] dokumentáció. A konfiguráció beállításának leggyorsabb módja a /etc/network/interfaces file szerkesztése. A beállítások teljes értékű életbelépéséhez a számítógép újraindítása szükséges. A szükséges interfaces fájlbejegyzéseket az 5. és a 6. *ábra* mutatja.

```
auto lo
iface lo inet loopback

allow-hotplug eth0
auto eth0
iface eth0 inet dhcp

auto sit0
iface sit0 inet6 static
address 2002:c1e0:81a5::1
netmask 48
gateway ::192.88.99.1
```

5. ábra A debianhost1 /etc/network/interfaces fájl tartalma sit esetén.

```
auto lo
iface lo inet loopback

allow-hotplug eth0
auto eth0
iface eth0 inet static
address 193.225.151.78
netmask 255.255.255.240
gateway 193.225.151.65
broadcast 193.225.151.79

auto 6to4
iface 6to4 inet6 v4tunnel
address 2002:c1e1:974e::1
netmask 48
gateway ::192.88.99.1
endpoint any
local 193.225.151.78
```

6. ábra A viktordebian /etc/network/interfaces fájl tartalma v4tunnel esetén.

A viktordebian számítógép esetén az IPv4 cím-konfigurációt statikusan kellett beállítanom, hogy biztosan különböző címtartományba kerüljön a gép, míg a debianhost1 esetén a hálózati kártya MAC címe alapján DHCP címkiosztáskor a kliens mindig ugyanazt a publikus IP címet kapta. A számítógépek újraindítása után az ifconfig parancs segítségével ellenőrizhető a konfigurációk aktualitása. Érdekes megfigyelni az általam kiadott parancsok eredményeit a 7. és a 8. *ábrán*. Az

ábrák bizonyíték erejű magyarázattal bírnak, melyet a későbbiekben az implementációk helyes működésének ellenőrzésekor hívok segítségül (4.2.1-ben).

```
root@debianhost1:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:f1:1e:2a
          inet addr:193.224.129.165  Bcast:193.224.129.175  Mask:255.255.255.240
          inet6 addr: fe80::a00:27ff:fef1:1e2a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:11 errors:0 dropped:0 overruns:0 frame:0
          TX packets:199 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3762 (3.6 KiB)  TX bytes:8598 (8.3 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:430 errors:0 dropped:0 overruns:0 frame:0
          TX packets:430 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:38714 (37.8 KiB)  TX bytes:38714 (37.8 KiB)

sit0      Link encap:IPv6-in-IPv4
          inet6 addr: 2002:c1e0:81a5::1/48 Scope:Global
          inet6 addr: ::127.0.0.1/96 Scope:Unknown
          UP RUNNING NOARP  MTU:1480  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

7. ábra A debianhost1 aktuális beállításai.

```
root@viktordebian:~# ifconfig
6to4      Link encap:IPv6-in-IPv4
          inet6 addr: 2002:c1e1:974e::1/48 Scope:Global
          inet6 addr: ::193.225.151.78/128 Scope:Compat
          UP RUNNING NOARP  MTU:1480  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0      Link encap:Ethernet  HWaddr 00:60:52:09:b5:38
          inet addr:193.225.151.78  Bcast:193.225.151.79  Mask:255.255.255.240
          inet6 addr: fe80::a00:27ff:fef1:1e2a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:68 errors:0 dropped:0 overruns:0 frame:0
          TX packets:895 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:23256 (22.7 KiB)  TX bytes:37830 (36.9 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:430 errors:0 dropped:0 overruns:0 frame:0
          TX packets:430 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:38714 (37.8 KiB)  TX bytes:38714 (37.8 KiB)
```

8. ábra A viktordebian aktuális beállításai.

A 6to4 relay működésének teljes körű leellenőrzéséhez ping parancsokat használtam. A teszt során ellenőriztem a kliensek elérhetőségét IPv4- és IPv6 kapcsolaton keresztül, a publikus internet elérhetőségét (a Google egyik szerverét) szintén mindkét verziójú protokoll segítségével. A ping tesztek mindkét implementáció, tehát mindkét számítógép esetén külön-külön elvégeztem, bizonyítva ezzel az egyes implementációk 6to4 host mechanizmust megvalósító képességét. A kapcsolatok tesztelése és annak eredményei a 9., 10., 11., 12., 13., 14., 15. és a 16. ábrán láthatók. Az ábrák segítségével igyekszem bemutatni a kliensek számára elérhető kapcsolatok számosságát.

```
root@debianhost1:~# ping6 2002:c1e1:974e::1
PING 2002:c1e1:974e::1(2002:c1e1:974e::1) 56 data bytes
64 bytes from 2002:c1e1:974e::1: icmp_seq=1 ttl=64 time=1.88 ms
64 bytes from 2002:c1e1:974e::1: icmp_seq=2 ttl=64 time=1.46 ms
64 bytes from 2002:c1e1:974e::1: icmp_seq=3 ttl=64 time=1.52 ms
^C
--- 2002:c1e1:974e::1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.462/1.622/1.882/0.188 ms
```

9. ábra A viktordebian 6to4-es IPv6 elérhetőségének ellenőrzése.

```
root@debianhost1:~# ping6 google.com
PING google.com(fa-in-x8b.1e100.net) 56 data bytes
64 bytes from fa-in-x8b.1e100.net: icmp_seq=1 ttl=58 time=27.6 ms
64 bytes from fa-in-x8b.1e100.net: icmp_seq=2 ttl=58 time=26.7 ms
64 bytes from fa-in-x8b.1e100.net: icmp_seq=3 ttl=58 time=26.8 ms
^C
--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 26.790/27.071/27.610/0.381 ms
```

10. ábra A debianhost1 natív IPv6 kapcsolatának ellenőrzése.

```
root@debianhost1:~# ping google.com
PING google.com (173.194.70.113) 56(84) bytes of data.
64 bytes from fa-in-f113.1e100.net (173.194.70.113): icmp_req=1 ttl=43 time=20.5 ms
64 bytes from fa-in-f113.1e100.net (173.194.70.113): icmp_req=2 ttl=43 time=20.2 ms
64 bytes from fa-in-f113.1e100.net (173.194.70.113): icmp_req=3 ttl=43 time=20.1 ms
^C
--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 20.167/20.315/20.507/0.217 ms
```

11. ábra A debianhost1 IPv4 kapcsolatának ellenőrzése.

```
root@debianhost1:~# ping 193.225.151.78
PING 193.225.151.78 (193.225.151.78) 56(84) bytes of data.
64 bytes from 193.225.151.78: icmp_req=1 ttl=63 time=1.14 ms
64 bytes from 193.225.151.78: icmp_req=2 ttl=63 time=0.907 ms
64 bytes from 193.225.151.78: icmp_req=3 ttl=63 time=0.850 ms
^C
--- 193.225.151.78 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.850/0.965/1.140/0.130 ms
```

12. ábra A viktordebian IPv4 elérhetőségének ellenőrzése.

```
root@viktordebian:~# ping6 2002:c1e0:81a5::1
PING 2002:c1e0:81a5::1(2002:c1e0:81a5::1) 56 data bytes
64 bytes from 2002:c1e0:81a5::1: icmp_seq=1 ttl=64 time=1.36 ms
64 bytes from 2002:c1e0:81a5::1: icmp_seq=2 ttl=64 time=1.25 ms
64 bytes from 2002:c1e0:81a5::1: icmp_seq=3 ttl=64 time=1.23 ms
^C
--- 2002:c1e0:81a5::1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.234/1.286/1.365/0.056 ms
```

13. ábra A debianhost1 6to4-es IPv6 elérhetőségének ellenőrzése.

```
root@viktordebian:~# ping6 google.com
PING google.com(fa-in-x8b.1e100.net) 56 data bytes
64 bytes from fa-in-x8b.1e100.net: icmp_seq=1 ttl=58 time=27.1 ms
64 bytes from fa-in-x8b.1e100.net: icmp_seq=2 ttl=58 time=26.9 ms
64 bytes from fa-in-x8b.1e100.net: icmp_seq=3 ttl=58 time=27.2 ms
^C
--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 26.998/27.114/27.232/0.095 ms
```

14. ábra A viktordebian natív IPv6 kapcsolatának ellenőrzése.

```
root@viktordebian:~# ping google.com
PING google.com (173.194.70.100) 56(84) bytes of data.
64 bytes from fa-in-f100.1e100.net (173.194.70.100): icmp_req=1 ttl=41 time=20.9 ms
64 bytes from fa-in-f100.1e100.net (173.194.70.100): icmp_req=2 ttl=41 time=20.4 ms
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 20.455/20.706/20.957/0.251 ms
```

15. ábra A viktordebian IPv4 kapcsolatának ellenőrzése.

```
root@viktordebian:~# ping 193.224.129.165
PING 193.224.129.165 (193.224.129.165) 56(84) bytes of data.
64 bytes from 193.224.129.165: icmp_req=1 ttl=63 time=1.04 ms
64 bytes from 193.224.129.165: icmp_req=2 ttl=63 time=1.11 ms
^C
--- 193.224.129.165 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.045/1.082/1.119/0.037 ms
```

16. ábra A debianhost1 IPv4 elérhetőségének ellenőrzése.

A sikeres ICMP válaszok alapján egyértelműen kijelentem, hogy az anycast címen elérhető 6to4 relay router ellátja feladatát: A kliensek számára elérhetővé teszi a publikus IPv6 internetet és a publikus 6to4 routing tartományokat. Az én feladatom tehát egy hasonlóképpen működő router beüzemelése és egy szimulált környezet létrehozása, melyben a különböző operációs rendszerek implementációit terheléstesztnek vehetem alá.

4. Implementációk

4.1. Implementációk

A dolgozatomban Linux, OpenBSD, FreeBSD, NetBSD és OpenWRT platformokon futó 6to4 implementációk képességeinek bemutatása a cél, így első feladatom a működőképes, szabad implementációk felkutatása volt. Internetes keresőmotorok segítségével sokat megtudhat az ember a világról, ám speciális témákban, mint például az én konkrét helyzetem, feltétlenül szükséges célirányosan "jó helyen" keresgélni. Nem is beszélve arról, hogy az aktualitások felkutatása jóval nehezebb feladat, mint a régebbi, elavult technikák megtalálása, amelyekben már sok ember szerzett tapasztalatot, és ha szerencsénk van, esetleg közzé is tették azokat. Ezekből az okokból kifolyólag én az operációs rendszerek fejlesztői fórumait, fejlesztői levelezőlistáit kutattam fel, hiszen ki más ismerhetné jobban a rendszerek működését, mint maguk a fejlesztők. Így bukkantam a következő implementációkra:

- Debian:
 - sit interfész
 - A 2001-es Linux kernelektől fogva támogatott. Kezd elavulttá válni.
 - v4tunnel
 - Manapság ez a legelterjedtebb alagút képző implementáció.
- OpenBSD:
 - gif(4) interfész
- FreeBSD:
 - stf(4) interfész
- NetBSD:
 - stf(4) interfész
- OpenWRT:
 - 6to4 és kmod-sit csomag együttesen
- "Unix-alapú":
 - gre(4) kernel interfész

Az implementációk utáni zárójelben lévő szám a szoftveres interfész verziószámát jelöli. Az imént említett működőképes implementációk részletes ismertetése és konfigurációik az 5. fejezetben kerülnek bemutatásra. A 4.2. alfejezetben a 6to4 relayként nem alkalmazható implementációkra külön kitérek.

Léteznek továbbá automata alagútképző szoftvercsomagok, melyek előzetes regisztráció után ingyenesen elérhetők bárki számára: aiccu a Sixxs-től [11, 12], tspanc a Hexago-tól [13]. Gyakorlatilag

ez egy szolgáltatás, a két szervezet 6to4 relay routereket üzemeltet, a felhasználó ezekhez az útvonalválasztókhoz csatlakozhat. Ám az ő konfigurációik is szintén az általam felfedezett implementációkat használják, így nem volt érdemes külön ezekkel a csomagokkal is foglalkoznom.

4.2. Relayként nem használható implementációk

Feliratkoztam az OpenBSD fejlesztői levelezőlistájára. Ott kaptam két tanácsot, a gre(4) és gif(4) implementációk alkalmazhatóságával kapcsolatban. A levelezés archív változata elérhető a [14]-es hivatkozáson keresztül. A következő két alfejezetben az említett implementációkkal szerzett tapasztalataimat osztom meg az olvasóval.

4.2.1. gre interfész

A Cisco fejlesztése, a 2.4.x kernelaktól fogva támogatott. A gre, mint nem Unix fejlesztők által készített implementáció, alapértelmezetten nem biztos, hogy elérhető az adott Unix-típusú operációs rendszerben. Mindenek előtt győződjünk meg az operációs rendszerünk hivatalos honlapján közölt dokumentációkból, hogy az adott rendszer tényleg támogatja-e a gre interfész használatát. A gre interfész modul állapotának ellenőrzéséhez adjuk ki a `/sbin/lsmodule | grep ip_gre.o` parancsot, ha ez nem írja ki a konzolra az „ip_gre.o”-t az azt jelenti, hogy a modul nincs betöltve. A modul betöltéséhez adjuk ki root-ként a `/sbin/modprobe ip_gre.o` parancsot. A modult leíró fájl meglétét manuálisan is leellenőrizhetjük a `/lib/modules/*kernelszám*/kernel/net/ipv4` mappában. A gre interfész konfigurálásában segítséget nyújtott a [15]-ös leírás. Egy lehetséges gre konfiguráció Debian Linux alatt a 17. és a 18. ábrán látható.

```
auto lo
iface lo inet loopback

allow-hotplug eth0
auto eth0
iface eth0 inet dhcp

auto tun0
iface tun0 inet6 static
address 2002:c1e1:974d::1
netmask 16
pre-up iptunnel add tun0 mode gre local 193.225.151.77 remote 193.225.151.78 ttl
255
up ifconfig tun0 multicast
pointopoint 2002:c1e1:974e::1
```

17. ábra A debianhost1 `/etc/network/interfaces` fájl tartalma gre interfész konfiguráció esetén.

```

auto lo
iface lo inet loopback

allow-hotplug eth0
auto eth0
iface eth0 inet dhcp

auto tun0
iface tun0 inet6 static
address 2002:c1e1:974e::1
netmask 16
pre-up iptunnel add tun0 mode gre local 193.225.151.78 remote 193.225.151.77 ttl
255
up ifconfig tun0 multicast
pointopoint 2002:c1e1:974d::1

```

18. ábra A viktordebian /etc/network/interfaces fájl tartalma gre interfész konfiguráció esetén.

A létrehozandó interfészt mindkét kliens esetén tun0-nak neveztem el. A gépek beállításai újraindítás után leellenőrizhetők az `ifconfig` paranccsal, példaként a viktordebian aktuális interfész konfigurációi láthatók a *19. ábrán*.

```

root@viktordebian:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:60:52:09:b5:38
          inet addr:193.225.151.78  Bcast:193.225.151.79  Mask:255.255.255.240
          inet6 addr: fe80::a00:27ff:fef1:1e2a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:602 errors:0 dropped:0 overruns:0 frame:0
          TX packets:73 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:48028 (46.9 KiB)  TX bytes:10151 (9.9 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:560 (560.0 B)  TX bytes:560 (560.0 B)

tun000    Link encap:UNSPEC  HWaddr C1-E1-97-4E-00-00-65-74-00-00-00-00-00-00-00-00-
          inet6 addr: 2002:c1e1:974e::1/16 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1476  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1664 (1.6 KiB)  TX bytes:1664 (KiB)

```

19. ábra A viktordebian aktuális beállításai gre konfiguráció esetén.

A tun0 interfész "Link encap: UNSPEC" értéke már önmagában gyanúra ad okot, hiszen korábban sikeres beágyazó pseudo-interfész esetén ott "IPv6 – in – IPv4" érték szerepelt (például: *7. ábra*). A tesztelendő kapcsolat csupán a két gép közötti kommunikációra koncentrálódik. Eredményes ping tesztet csak /16-os 6to4 prefix használatakor sikerült végeznem. Továbbá a két kliens csak akkor volt képes a kommunikációra, ha mindkettejük gre konfigurációval működött, közös Ethernet hálózatra

csatlakoztak és egy IPv6 alhálózatba tartoztak. Mivel a 192.88.99.1 anycast IP cím segítségével nem sikerült működőképes konfigurációt készítenem (a gre interfész nem kapcsolódott fel, ha az említett címet használtam), ráadásul a sikeres kommunikáció feltétele mindkét oldalról (kliens és kiszolgáló) a gre interfész konfigurációk alkalmazása, a jelen összeállítást nem fogadom el működőképes 6to4-relay és host implementációnak sem. A gre implementáció nem képes dinamikusan tunnelt (alagutat) létrehozni, a kapcsolatot magunknak kell statikusan beállítani mindkét összekapcsolni kívánt számítógépen. Ez így pusztán 6in4 mechanizmus, amely ugyan IPv6 csomagokat továbbít IPv4 csomagokba ágyazva, de nem biztosítja számunkra eltérő 6to4 implementációt alkalmazó kliensek csatlakozását.

4.2.2. gif interfész

Az OpenBSD kapcsán bukkantam a gif interfész negyedik verziójára. Elméletileg ez a kernel interfész is bármely Unix-alapú operációs rendszerben elérhető. Bár Jun-ichiro "Itojun" Hagino kijelentése szerint [16]: „OpenBSD does not integrate 6to4 (RFC3056) automatic tunneling mechanism, which is stf(4) in other BSDs.” Azaz az OpenBSD-ben nincs implementálva a 6to4 mechanizmus biztonsági megfontolások alapján. Én a teljesség kedvéért mégis utánajártam, hogyan működhetne 6to4 relayként egy OpenBSD rendszer.

Jason Barbier [14] programozó tanácsára felkerestem a Hurricane Electric szolgáltató honlapját. Személyes adatok megadása után bárkit regisztrálnak és egy online menürendszeren keresztül szolgáltatást aktiválhatunk magunknak, teljesen ingyen. A Tunnelbroker.net fórumain keresztül jutottam el egy FreeBSD konfigurációhoz [17], mellyel elvileg 6to4 szolgáltatást nyújtanak a felhasználók számára. Mint kiderült, a fórumozók egy része tévesen azonosította a szolgáltatást 6to4 mechanizmusnak. Igaz ugyan, hogy IPv4 hálózaton keresztül elérhetővé teszik számunkra a publikus IPv6 hálózatot, viszont csak egy szimpla IPv6 in IPv4 alagutat kell konfigurálnunk a Hurricane Electric honlapján, az ott közölt minta konfiguráció alapján beállítani egy gif interfészt OpenBSD rendszerünkön és már kész is az "ál 6to4 host". Az általam tesztelt konfiguráció a 20. ábrán látható.

```
tunnel 193.224.129.165 216.66.86.122
!ifconfig gif1 inet6 alias 2001:470:6e:1d5::2 2001:470:6e:1d5::1 prefixlen 128
!route -n add -inet6 default 2001:470:6e:1d5::1
```

20. ábra A /etc/hostname.gif1 fájl tartalma gif interfész konfiguráció esetén.

Az OpenBSD-ben az interfészek konfigurációjának több módja is létezik. Az általam legegyszerűbbnek vélt a következő: létre kell hozni egy interfész fájlt az etc mappán belül. A konfigurációs fájl szintaktikája hostname.*, ahol a csillag az adott interfész nevét jelöli. Jelen esetben a létrehozandó interfészt gif1-nek nevezttem el.

A Hurricane Electric biztosított számomra egy szerveret, mely a 216.66.86.122 IPv4 címen érhető el, és a szerver a 2001:470:6e:1d5::1/64 IPv6 címen fogadja a beágyazott IPv6 csomagjaimat. Nekem, mint felhasználónak, a kliens oldalról a saját publikus IPv4 címemet és a HE szolgáltató által kiosztott 2001:470:6e:1d5::2/64 globális IPv6 címet felhasználva kellett konfigurálnom IPv6 in IPv4 alagutat. A mechanizmus működik, csak hogy ez ismét nem 6to4, hanem 6in4. Már az IPv6 prefix sem stimmel a feltételekhez. Mivel a 192.88.99.1 anycast IP cím segítségével nem sikerült működőképes konfigurációt készítenem (a gif interfész nem kapcsolódott fel, ha az említett címet használtam), és furcsa módon az alagút létrehozásához 128-as prefix hosszra kell alkalmazni, kijelentem, hogy a gif nem használható 6to4 host implementációként.

Mivel azonban létre lehet hozni alagutat OpenBSD-n, abban az esetben, ha 6to4 prefix alkalmazásával is sikerülne működő gif konfigurációt készíteni, az esély megvan rá, hogy 6to4 relay routerként még üzemelhet az operációs rendszer.

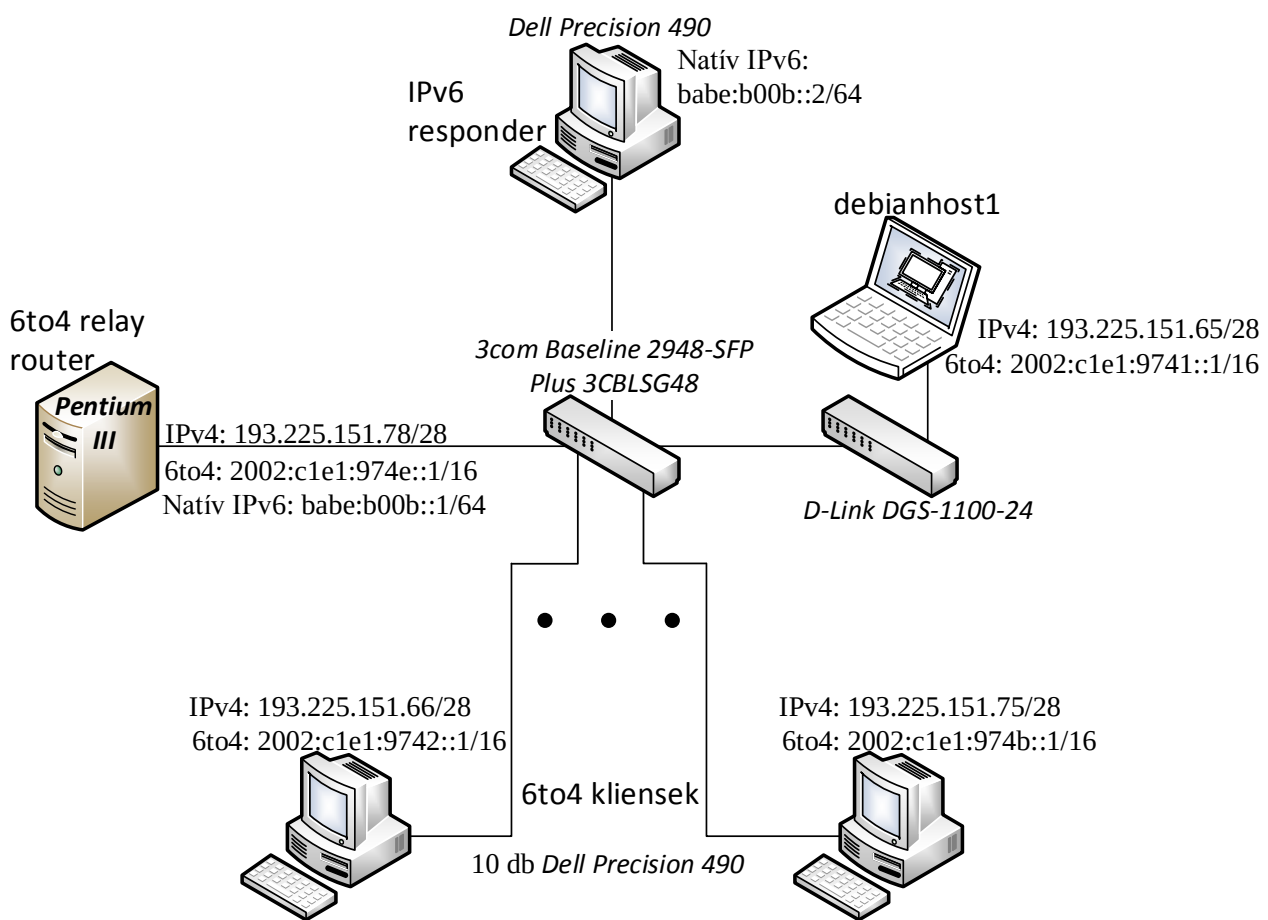
Nos, sikerült is-, meg nem is a feltevésnek eleget tennem. Az OpenBSD-t, mint 6to4 relay routert ugyan alávettem terheléstesztnek, de a gif és ezáltal az OpenBSD nem használható teljes értékű relay implementációként. A részleteket és a mérés eredményét az **5.2.6.** alfejezetben ismertetem részletesen.

5. 6to4 relay implementációk terheléstesztje

5.1. Mérési körülmények

5.1.1. Mérési elrendezés

Az implementációk terheléstesztjét a Széchenyi István Egyetem Távközlés-informatikai Laboratóriumában végeztem el. A teljesítmőképesség-vizsgálat elszeparált helyi hálózaton folyt, hogy az esetleges más hálózat és a közbelső hálózati eszközök által okozott időbeli késleltetés ne befolyásolhassa a végeredményt. A mérési összeállítás a 21. ábrán látható.



21. ábra Mérési összeállítás.

A mérés izolált környezetben zajlott, ezért önkényesen, bármilyen publikus, globális IP címtartományt felhasználhattam volna. A valóságot szimulálva, nyilvános IPv4 tartománynak az egyetem laboratóriumának kiosztott 193.225.151.64/28 hálózatot választottam. A kliensek által elérni kívánt célhálózatnak pedig a 2001:738:2c01:8000::/64 globális IPv6 tartományt.

Az imént említett két hálózat egyébként az útvonalválasztók és a hálózat megfelelő konfigurálása után a valóságban is ténylegesen igénybe vehető címeket tartalmaz.

A relay és a responder között elhelyezkedő globális IPv6 hálózathoz egy fiktív tartományt, a babe:b00b::/64-et választottam. A két hálózat mérete a kliensek és az elérni kívánt célállomások maximálisan becsült számának tudatában került meghatározásra. A 193.225.151.64/28 hálózatban 14, míg a 2001:738:2c01:8000::/64 hálózatban 18,447 trillió érvényes IP cím kerülhetne kiosztásra. A 14 IPv4 cím fedezi a 10 db kliens, plusz a 6to4 relaynek szánt 1 db, plusz a vezérlő számítógépem 1 db címfoglalási igényét. A 18,447 trillió IPv6 címről pedig azt gondoltam, biztosan elegendő lesz az általam generált terhelések során.

A 21. ábra tetején található IPv6 responder szimulálta az elérni kívánt server sokaságot, tehát az összes 6to4 tartományból érkező kérésre ez válaszolt. A mérni kívánt eszköz a 6to4 relay router, melyen az összes forgalom áthaladt. Ahogy a nevében is szerepel, ez valósítja meg a relayt. A 6to4 kliensektől IPv4 alagútban érkező IPv6 csomagokat továbbítja a célhálózat, azaz a responder felé. A válaszként visszafele érkező IPv6 csomagokat pedig elirányítja a 6to4 célhálózat felé.

A 10 db Dell számítógép hozta létre a terhelést, szimulálva ezzel egy nagyszámú klienspark kéréseit.

Az ábra jobb szélén található D-Link hálózati kapcsoló és a debianhost1 nevű számítógép csupán az én munkámat könnyítette meg, a mérési eredményeket nem befolyásolták. A debianhost1 géppel vezéreltem a mérést, a switch segítségével pedig rácsatlakoztam a laboratórium hálózatára, ugyanakkor többlet csatlakozási pontot biztosítottam magamnak.

5.1.2. Hardver és szoftver konfigurációk

Az implementációk vizsgálatához egy régi, kis teljesítményű számítógépet választottam, hogy biztosan 100%-os processzorterhelést érjek el a relayen, ebből fakadóan, maximális terhelés esetén is megvizsgálhassam az implementációk teljesítőképességét. A teszteket Gigabit Ethernet hálózaton végeztem, hogy a hálózat sebessége ne befolyásolja a mérések eredményét.

Relay tulajdonságai:

- Intel D815EE2U alaplap
- Intel Pentium(R) III (800 MHz) processzor
- 128 MB (100 MHz) SD memória
- 2 db TP-LINK TG-3269 REV 3.0 Gigabit PCI hálózati kártya

A 10 db kliens azonos kiépítésű Dell Precision Workstation 490 típusú számítógép.

Kliensek tulajdonságai:

- Dell 0GU083 alaplap Intel 5000X chipsettel
- 2 db Intel Xeon 5140 (2.33 GHz) dupla magos processzor
- Broadcom 5752 NetXtreme Gigabit PCIe hálózati kártya

- 4 db 1 GB (533 MHz) DDR2 FB-DIMM memória (quad channel módban)

A responder számítógép szintén egy Dell Precision Workstation 490 típusú munkaállomás. A különbség a 10 db kliens számítógép konfigurációjához képest a processzorban rejlik: 2 db Intel Xeon 5160 3GHz dupla magos processzor. A klienseken a Debian 6.0.7, Linux 2.6.32-5-amd64 kernelverziójú operációs rendszer futott. A responder számítógépen az OpenBSD 5.3 64 bites verziója végezte a kiszolgálást.

5.1.3. Mérési módszer

A terhelés kivitelezéséhez Répás Sándor munkájában [7](52. oldal 48. ábra) közölt mérési script-hez hasonló bash-shell script-et használtam, melyet a 22. *ábra* tartalmaz.

```
#!/bin/bash
echo -e "\n\033[4;35mFut a script!\033[0m"

i=`cat /etc/hostname | grep -o '[0-9]'`
for b in {0..255}
do
    rm -rf $b
    mkdir $b
    for c in {0..252..4}
    do
        ping6 -c8 -i0 2001:738:2c01:8000::193.$i.$b.$c >> \
            $b/6to4-193-$i-$b-$c &
        ping6 -c8 -i0 2001:738:2c01:8000::193.$i.$b.$c >> \
            $b/6to4-193-$i-$b-$c &
        ping6 -c8 -i0 2001:738:2c01:8000::193.$i.$b.$((c+1)) >> \
            $b/6to4-193-$i-$b-$((c+1)) &
        ping6 -c8 -i0 2001:738:2c01:8000::193.$i.$b.$((c+1)) >> \
            $b/6to4-193-$i-$b-$((c+1)) &
        ping6 -c8 -i0 2001:738:2c01:8000::193.$i.$b.$((c+2)) >> \
            $b/6to4-193-$i-$b-$((c+2)) &
        ping6 -c8 -i0 2001:738:2c01:8000::193.$i.$b.$((c+2)) >> \
            $b/6to4-193-$i-$b-$((c+2)) &
        ping6 -c8 -i0 2001:738:2c01:8000::193.$i.$b.$((c+3)) >> \
            $b/6to4-193-$i-$b-$((c+3)) &
        ping6 -c8 -i0 2001:738:2c01:8000::193.$i.$b.$((c+3)) >> \
            $b/6to4-193-$i-$b-$((c+3)) &
    done
done
echo -e "\n\033[4;32mVégzett a script.\033[0m"
```

22. ábra 6to4ping.sh

A terhelést létrehozó script kísérletezés útján jött létre. Az összes tesztelni kívánt operációs rendszert és implementációt előzetesen végig teszteltem. A célom egy olyan terhelést kiváltó script megírása volt, amely nyolc kliens egyidejű terhelése esetén már leterheli annyira a relayt, hogy legyenek időpillanatok, amikor a processzorkihasználtság 100 %. Ebből következően a tíz klienses terhelés során a relay processzorterhelése szinte végig 100 %-os értékre várható, így az egyes implementációk képessége a rendszer túlterhelése esetén is megfigyelhető lesz.

A script második és utolsó sora csupán kiírja a konzol képernyőre színes betűkkel a script aktuális állapotát, ez a két sor engem segít a mérés állapotának nyomon követésében. Az *i* változó értéke az aktuális kliens hostnevéből a szám értékét veszi fel. Tehát például a fekete1 nevű számítógép esetén *i*=1 értéket vesz fel. A *b* változó nullától kétszázötvenötig egyesével, összesen 256 értéket helyettesít. A *b* változó aktuális értékét könyvtárak létrehozására használok fel, melyekbe az eredményfájlok kerülnek. A script az előző mérésekből származó eredmény mappákat törli, ha voltak ilyenek, majd újra létrehozza azokat. A belső for ciklusban felhasznált *c* változó nullától kétszázötvenkettőtől négyesével, összesen 64 értéket vesz fel. A for ciklus do/done operandusai között található maga a terhelést létrehozó ping parancssorozat. A ping6 program segítségével IPv6 címmel rendelkező állomások elérhetőségét ellenőrizhetjük. A -c8 kapcsolóval megadom a programnak, hogy 8 db ICMP Echo-Request üzenetet, azaz nyolc IP csomagot küldjön. A -i0 kapcsolóval a csomagok küldése közti időtartamot nulla másodpercre állítom. Az elküldött kérésekre érkező válaszokat és a csomagstatisztikákat a 256 létrehozott könyvtáron belül több, IP címenként különálló „6to4-193-*-*.*” fájlnev szerkezetű fájlokba mentem. A ciklusban egyszerre 8 ping parancs futtatását hajtja végre a script. A script lefutása után a kliensek egyenként $256 \cdot 64 \cdot 8 \cdot 8 = 1\,048\,576$ db IP csomagot küldtek, kliensenként összesen $256 \cdot 64 \cdot 4 = 65\,536$ IP címre. Maximális terhelés esetén $10 \cdot 65\,536 = 655\,360$ különböző IP címre küldtek kérést a kliensek, ez viszonylag nagy szám, de belátható, hogy a korábban (túl)becsült 18,447 trillió címter bőven fedezi a méréseim során felmerülő IP cím szükségletet.

A relayen a mérések alatt a vmstat nevű elemzőprogrammal vizsgáltam a számítógép terheltségét. Ahhoz, hogy elegendő terheltség értéket fájlba tudjon menteni a program, a vmstat indítása után megváltoztattam a program „nice value” értékét a renice -10 PID paranccsal, ahol a PID (Process Identifier) a folyamatot azonosító számot jelöli, melyet a ps aux parancs kimenetéből nyertem ki. Minderre azért volt szükség, mert nagy terhelés esetén a relay az összes erőforrást a csomagok célba juttatására használja fel, és az eredmények kiírására, tehát a vmstat program részére már nem marad szabad processzorkapacitás. Utólagos ellenőrzéseim alapján ez a rendszerbeavatkozás az eredmények alakulása szempontjából csupán elhanyagolható mértékben befolyásolta némelyik implementáció teljesítőképességét, ha a prioritást a maximális -20 értékkel szabályoztam volna, az a valóságtól nagymértékben eltérő teljesítőképesség eredményeket produkált volna, így kísérletezés útján, kompromisszum kötelezettségében maradtam a -10-es értéknél.

A relayen a vmstat 1 >> vmstat.txt paranccsal létrehoztam egy vmstat.txt nevű állományt, majd 1 másodpercenként ebbe a fájlba irányítottam a rendszert jellemző terheltségi adatokat, többek között a memória-, és a processzorkihasználtság aktuális értékeit. A relay terhelését minden implementáció esetén először 10 klienssel végeztem el. A 10 kliens egyidejűleg kezdte el futtatni az automeres.sh scriptet, mely kliensenként mérte a mérés idejét és meghívta a 22. ábrán bemutatott mérőscriptemet.

Az automeres.sh program véleményem szerint, feladatát tekintve, nem túl lényeges momentum a mérés során, ezért a dolgozatomban nem részletezem működését. A függelékben az 56. ábrán tájékozódhat mélyebben az érdeklődő.

A script lefutását követően átmásoltam a relayről a vmstat.txt fájlt a vezérlő számítógépre, majd a relay újraindítására került sor, hogy az útvonal bejegyzések eltávolításával töröljem a relay átmeneti routing tábláját, s így kiszűrjem a mérést befolyásoló tényezőket. A klienseken az automeres.sh elmentette az eredményeket egy erre a célra létrehozott külön könyvtárba. A relay "felállása" után eggyel kevesebb klienssel kezdődött előlről a mérési folyamat. A mérés sorozat addig folytatódott, míg a végén már csak egy kliens terhelte a relayt. A tíz mérés végeztével a kliensek az eredményeiket tartalmazó könyvtárat gzip-pel betömörítették, majd a tömörített állományokat a debianhost1 számítógépemre másoltam egy NFS (Network File System) megosztáson keresztül. Az eredmények mentése után a mérésben résztvevő összes számítógépet, tehát az egész mérőrendszert újraindítottam, mielőtt a következő implementációt teszteltem volna.

5.1.4. Responder számítógép beállításai

Ahogy az 5.1.1. alfejezetben már említettem, a responder számítógép feladata az Echo-Request üzenetek megválaszolása volt. Répás Sándor [7](7.2.3. alfejezet, második bekezdés) és jómagam kutatásai szerint is, a NAT66 [6] mechanizmus az OpenBSD és a pf tűzfalprogram együttes alkalmazásával valósítható meg a legegyszerűbben, továbbá az említett kombináció képes a legnagyobb teljesítményre a létező NAT66 implementációk közül. A szükséges beállításokat a 23. ábra tartalmazza.

```
set timeout interval 2
set limit states 400000
pass in on bge0 inet6 from any to 2001:738:2c01:8000::/64 rdr-to babe:b00b::2
```

23. ábra A /etc/pf.conf fájl tartalma a responder számítógépen.

Az ábrán látható fájlbejegyzések nagyszámú kapcsolatot engedélyeznek azonos időben a responder számítógép irányába. A bge0 a responder számítógép interfésze, melyen keresztül a mérőhálózatra csatlakozott. A harmadik bejegyzés a 2001:738:2c01:8000::/64 tartomány irányába érkező kéréseket a responder globális IPv6 címére, a babe:b00b::2-re irányítja, így a gép minden kérésre ugyanúgy válaszol, mintha az a saját publikus IP címére érkezett volna. A vmstat programmal a válaszoló számítógép teljesítményét is folyamatosan monitoroztam a mérések alatt. A terhelések miatt bekövetkezett többlet memória igény maximális értéke 3,831 MB, a processzorkihasználtság maximális értéke 12 % volt. Az alacsony erőforrásigényből következtethető, hogy a responder gép teljesítménye nem befolyásolta a mérés kimenetelét.

A responder számítógép irányába vezető útvonal beállításáért felelős fájlbejegyzéseket az egyes 6to4 relay implementációk konfigurációjának ismertetése során külön-külön mutatom be az 5.2. alfejezetben.

5.1.5. Kliens számítógépek beállításai

A terhelést létrehozó Linux kliensek hálózati interfész beállítása a 24. ábrán látható.

```
auto lo
iface lo inet loopback

#publikus ipv4
auto eth0
iface eth0 inet static
address 193.225.151.66-75
netmask 255.255.255.240
gateway 193.225.151.78
broadcast 193.225.151.79

#6to4
auto sit0
iface sit0 inet6 static
address 2002:c1e1:9742::1- ...974b::1
netmask 16
gateway ::193.225.151.78
```

24. ábra A /etc/network/interfaces fájl tartalma a kliens számítógépeken.

A klienseken sit konfigurációt választottam a 6to4 host mechanizmushoz, de a v4tunnel is megfelelő lett volna. A teszt terheléseim során ugyanis mindkét implementáció egyformán teljesített, mint terhelést generáló implementáció.

5.2. Relay konfigurációk

Ebben az alfejezetben a különböző relay implementációk működéséhez és méréséhez szükséges beállításokat mutatom be a relay router szemszögéből. Az egyes opciók magyarázatai a bejegyzések fölötti # jel után olvashatók.

5.2.1. Debian 7.1.0_x86 – sit

```
auto lo
iface lo inet loopback

#publikus IPv4:
auto eth0
iface eth0 inet static
address 193.225.151.78
netmask 255.255.255.240
broadcast 193.225.151.79

#globális IPv6:
auto eth1
iface eth1 inet6 static
address babe:b00b::1
netmask 64
#IPv6-os csomagok továbbításának engedélyezése:
pre-up echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
#IPv4-es csomagok továbbításának engedélyezése:
pre-up echo 1 > /proc/sys/net/ipv4/ip_forward
#Responder-hez vezető útvonal statikus bejegyzése:
post-up ip -f inet6 route add 2001:738:2c01:8000::/64 via babe:b00b::2

#6to4:
auto sit0
iface sit0 inet6 static
address 2002:c1e1:974e::1
netmask 16
```

25. ábra A /etc/network/interfaces fájl tartalma Debian Linux-sit 6to4 relay router esetén.

5.2.2. Debian 7.1.0_x86 – v4tunnel

```
*
A 6to4 bejegyzéseken kívül minden megegyezik a Debian 7.1.0_x86 – sit beállításával (előző ábra).
*

#6to4:
auto 6to4
iface 6to4 inet6 v4tunnel
address 2002:c1e1:974e::1
netmask 16
endpoint any
local 193.225.151.78
```

26. ábra A /etc/network/interfaces fájl tartalma Debian Linux-v4tunnel 6to4 relay router esetén.

5.2.3. OpenWrt (Attitude Adjustment) 12.09_x86 – sit

A 6to4 relay implementáció betöltéséhez és a mérés kivitelezéséhez az alábbi csomagokat telepítettem: 6to4, kmod-sit, ip, procps. A procps csomag tartalmazza a vmstat elemzőprogramot.

Az OpenWrt, mint Linux disztribúció, elvileg hasonlóképpen kellene, hogy működjön, mint a Debian, ám tapasztalataim azt mutatják, hogy a Backfire 10.03.1 óta nem elegendő az interfész konfigurációkat egyetlen fájlban elhelyezni. A beállítások teljes értékű érvénybelépéséhez

csatlakozni kell az operációs rendszer által elsődlegesnek (routerek esetén WAN portnak) beállított interfészre, hogy a fizikai szintű kommunikáció megléte után a többi interfészt is felkapcsolja a rendszer. További érdekességként megemlítem, hogy a `/etc/config/network` fájlban csak egyetlen fizikai interfész beállítása lehetséges, és ha az itt beállított interfész nem az elsődleges hálózati csatoló, akkor semmilyen kártyát nem tölt be a rendszer automatikusan az indulásakor.

Én nem kívántam használni az elsődleges interfészt, ugyanis az a Pentium III számítógép alaplajára integrált 100 Mbit/s sebességű Ethernet kártya volt. A megfelelő interfészek használatához a 27. ábrán bemutatott módon statikus IP cím kiosztást választottam az alaplapi kártyához és rácsatlakoztam rajta keresztül is egy UTP kábellel a 3Com switch-re. Ennek következtében az összes interfész felkapcsolt és az operációs rendszer végrehajtotta a `/etc/rc.local` fájlban tárolt parancsokat, mely a 28. ábrán látható. A `/etc/rc.local` fájlra legalább olvasási és futtatási jogosultságot kell kiosztani a tulajdonos számára a `chmod 500` paranccsal, különben a rendszer betöltődésekor figyelmen kívül hagyja a fájlt, tehát a benne tárolt parancsok végrehajtására sem kerül sor. A relay konfigurációja így már teljes.

További érdekességként megjegyzem, hogy a Backfire 10.03.1-es verzió óta ebben az operációs rendszerben is sit pszeudo-interfészként jelenik meg az implementáció csakúgy, mint a Debian-ban, míg a régebbi verziókban tetszőlegesen elnevezhettük a 6to4 mechanizmust megvalósító logikai interfészt.

```
config interface loopback
    option ifname    lo
    option proto      static
    option ipaddr     127.0.0.1
    option netmask    255.0.0.0

config interface lan
    option ifname     eth0
    option proto       static
#6to4:
config interface sit0
    option proto       6to4
    option ipaddr      193.225.151.78
    option netmask     255.255.255.240
```

27. ábra A `/etc/config/network` fájl tartalma OpenWrt 12.09 6to4 relay router esetén.

```
#először az alapértelmezett hálózati csatolót kapcsolom fel:
ifconfig eth0 up

#publikus IPv4:
ifconfig eth1 193.225.151.78 netmask 255.255.255.240 up

#globális IPv6:
ifconfig eth2 babe:b00b::1 up

#6to4:
ifconfig sit0 2002:c1e1:974e::1/16 up

#Responder-hez vezető útvonal statikus bejegyzése:
ip -f inet6 route add 2001:738:2c01:8000::/64 via babe:b00b::2
```

28. ábra A /etc/rc.local fájl tartalma OpenWrt 12.09 6to4 relay router esetén.

```
#IPv4-es csomagok továbbításának engedélyezése:
net.ipv4.ip_forward=1

#IPv6-os csomagok továbbításának engedélyezése:
net.ipv6.conf.all.forwarding=1
```

29. ábra A /etc/sysctl.conf szükséges fájl tartalma OpenWrt 12.09 6to4 relay router esetén.

5.2.4. FreeBSD 9.1_x86 – stf

```
#ipv6 hálózati interfész konfiguráció engedélyezése:
ipv6_network_interfaces="auto"

#publikus IPv4:
ifconfig_re0="inet 193.225.151.78 netmask 255.255.255.240"

#globális IPv6:
ifconfig_rel_ipv6="inet6 babe:b00b::1/64"

#Responder-hez vezető útvonal statikus bejegyzése:
ipv6_static_routes="responderhez"
ipv6_route_responderhez="2001:738:2c01:8000::/64 babe:b00b::2"
#IPv6-os csomagok továbbításának engedélyezése:
ipv6_gateway_enable="YES"
#IPv4-es csomagok továbbításának engedélyezése:
gateway_enable="YES"

#6to4:
ipv6_prefix_stf0="2002:c1e1:974e::1/16"
stf_interface_ipv4addr="193.225.151.78"
```

30. ábra A /etc/rc.conf fájl tartalma FreeBSD 6to4 relay router esetén.

5.2.5. NetBSD 6.1.2_x86 – stf

A NetBSD operációs rendszer általam tesztelt verziójában sikeres 6to4 relay router "üzemmódhoz" a NetBSD interfész konfigurációit és a Unix rendszerek rc.local scriptjét együttesen kellett alkalmaznom. A /etc/rc.local fájlra NetBSD esetén is érvényes, hogy olvasási és futtatási

jogosultságot kell kiosztani a tulajdonos számára a `chmod` paranccsal. A szükséges beállítások a 31., 32., 33. ábrán láthatók.

```
up
media autoselect
inet6 babe:b00b::1 prefixlen 64 alias
```

31. ábra Globális IPv6 cím beállítása a `/etc/ifconfig.re0` fájlban NetBSD esetén.

```
up
media autoselect
193.225.151.78 netmask 0xffffffff media autoselect
```

32. ábra Publikus IPv4 cím beállítása a `/etc/ifconfig.re1` fájlban NetBSD esetén.

```
#Responder-hez vezető útvonal statikus bejegyzése:
route add -inet6 2001:738:2c01:8000:: -prefixlen 64 babe:b00b::2

#6to4:
ifconfig stf0 create
ifconfig stf0 inet6 2002:c1e1:974e::1 prefixlen 16 alias

#IPv4-es csomagok továbbításának engedélyezése:
sysctl -w net.inet.ip.forwarding=1
#IPv6-os csomagok továbbításának engedélyezése:
sysctl -w net.inet6.ip6.forwarding=1
```

33. ábra A `/etc/rc.local` fájl tartalma NetBSD 6to4 relay router esetén.

5.2.6. OpenBSD 5.3_x86 – gif

A 4.2.2. alfejezetben már említettem, hogy az OpenBSD operációs rendszerben a legkézenfekvőbb az interfész beállításokat a `/etc` mappában létrehozott `hostname` kezdetű fájlokba bejegyezni. Az "ál 6to4 host" konfiguráció sikerén felbuzdulva, hosszas kísérletezés árán sikerült egy, a 34. ábrán látható működőképes 6to4 relay konfigurációt létrehoznom.

```
tunnel 193.225.151.78 193.225.151.66
!ifconfig gif0 inet6 2002:c1e1:974e::1 prefixlen 16
```

34. ábra 6to4 relayt megvalósító implementáció OpenBSD-n, `/etc/hostname.gif0` fájl tartalma.

Az IPv6 csomagok beágyazásához egy alagutat kell létrehozni a kliens és a router között, ezt valósítja meg az első sor. A második sor a 6to4 IP címet rendeli a pszeudo-interfészhez, azaz a gif0-hoz. Az elképzelésem az volt, hogy minden egyes klienshez egy új gif interfészt hozok létre egyedi beállításokkal. Az ötlet azonban a gyakorlatban nem kivitelezhető. Az első, azaz jelen esetben a gif0 virtuális hálózati csatoló és a 193.225.151.66-os IP című gép között működik a kommunikáció, ám a többi gif interfész és a hozzájuk rendelt kliensek közti kapcsolat nem jön létre. Próbáltam a pf tűzfalat kikapcsolni, majd újraindítani a rendszert, de ez nem segített a mechanizmus kiterjesztésében. Mivel a jelenlegi 6to4 relay konfiguráció csak egy darab klienst képes kiszolgálni, az OpenBSD-t és vele

együtt a gif implementációt elvetem, mint 6to4 relay implementációt. Úgy tűnik Hagino-nak igaza volt, az OpenBSD-ben nincs lehetőségünk a mechanizmus megvalósítására egyszerű fájlbejegyzésekkel. A rendszert mindenesetre kíváncsiságból leteszteltem, az eredményeket az 5.4.6. alfejezetben tekintheti meg az olvasó. A relay router működéséhez a további szükséges beállításokat a 35., 36., 37. ábra tartalmazza.

```
!ifconfig re0 inet 193.225.151.78 netmask 255.255.255.240
```

35. ábra Publikus IPv4 cím beállítása a /etc/hostname.re0 fájlban OpenBSD esetén.

```
#globális IPv6:
!ifconfig re1 inet6 babe:b00b::1 prefixlen 64

#Responder-hez vezető útvonal statikus bejegyzése:
!route add -net -inet6 2001:738:2c01:8000::/64 babe:b00b::2
```

36. ábra Globális IPv6 cím + útvonal beállítása a /etc/hostname.re1 fájlban OpenBSD esetén.

```
#IPv4-es csomagok továbbításának engedélyezése:
net.inet.ip.forwarding=1

#IPv6-os csomagok továbbításának engedélyezése:
net.inet6.ip6.forwarding=1
```

37. ábra A /etc/sysctl.conf szükséges fájl tartalma OpenBSD 6to4 relay router esetén.

5.3. Kiértékelés

Az eredmények feldolgozása, a feldolgozás sebessége kulcsfontosságú kérdéskör volt számomra, ugyanis ha bármi hiba keletkezett a rendszerben, (például: egy számítógép nem vett részt a mérésben, később indult újra egy gép és ezzel késleltetést okozott a terhelésben, stb.) akkor az adott mérést, rosszabb esetben az egész méréssorozatot meg kellett ismételni. Belátható tehát, hogy az eredmény fájlok egyesével történő kiértékelése nem járható út.

A Microsoft Excel program ugyan segítséget nyújt, ám az adatbeillesztéssel és a számításokkal rengeteg idő elmegy, még akkor is, ha csak részeredményeket átlagolok, vagy összegzek. Ennek oka, hogy "kézzel", manuálisan kell bemásolnom minden egyes adathalmazt, ráadásul nagyméretű (több, mint 65536) elemszámú sor, vagy oszlop feldolgozása már az Excel számára is időigényes feladat, hiába van gyors számítógépünk. Más megoldás után kellett tehát nézmem.

Az eredmények feldolgozásához és kiértékeléséhez általam készített Bash Shell scripteket használtam. A programok elkészítésében nagy segítségemre voltak [7] és [18]. A scriptek megírása ugyan több napomba telt, de később az eredmények kinyerése során óriási könnyebbség volt, hogy automata programok gyorsan eredmény fájlokat készítenek nekem, amely fájlok már tartalmazzak minden szükséges adatot. Excelt csak az adatok megjelenítéséhez és a scriptjeim helyes működésének

az ellenőrzésére használtam. Az eredményfájlokból az összes adatot bemásoltam egy Excel munkafüzetbe implementációnként külön területre, a táblázatszerkesztő funkciók segítségével táblázatokba összesítettem az eredményeket, és diagramokat készítettem az ábrázolni kívánt értékekből. A kész táblázatokat és diagramokat egyszerűen csak átmásoltam a dolgozatom megfelelő sorába. Ha új értékekkel írtam felül egy meglévő eredmény táblázat celláit, az Excel munkafüzetben automatikusan frissült a diagram és az új eredményeket ismét átmásoltam a dolgozatomba.

A 22. ábrán bemutatott 6to4ping.sh script által készített eredményfájlok feldolgozása és kiértékelése volt az elsődleges feladatomban. Ahogy azt az 5.1.3. pontban említettem, az eredményeket kliensenként egy tömörített fájlba helyeztem, ezeket a fájlokat pedig összegyűjtöttem implementációnként külön könyvtárba. A kiértékelő scriptjeimet a Debian Linux 7.1.0 64 bites verziója alatt egy Asus N56VJ (Intel Core i7-3630QM processzorral) típusú hordozható számítógépen futtattam. Az összes eredmény kiértékelése nem tartott tovább 6 óránál.

A feldolgozás során az első lépés ezen tömörített állományok kicsomagolása és rendszerezése. Ezt a folyamatot végezte a kiertekeles.sh programom. Terjedelmét tekintve nem kívántam a dolgozatomban részletezni a script működését, de a mellékletek között a „scriptek” mappában megtalálható a program, a program tartalmazza a működésének ismertetését is. A script futása során egy külső script segítségével végzi el az érdemi munkát. Ez a meghívott program a felki.sh, melyet a 38. ábra részletez. A felki.sh végzi a 6to4ping.sh és az automeres.sh futása során gyártott nyers eredményfájlok feldolgozását és részben kiértékelését is. Az egyes műveletek fölött, a # jel után olvasható a lépések magyarázata.

```
#!/bin/bash
h=/home/v/eredmeny
k=/home/v/EREDMENYEK

#A script felveszi változóként, hogy melyik mérés eredményeit dolgozza fel éppen:
l=$(pwd | tr "/" "\t" | awk '{print $8}')
mkdir $k/$l
rm -rf $h/
mkdir $h

#A script az összes (256 db) eredményfájlt megvizsgálja a ciklusban:
for b in {0..255}
do
#Kigyűjtöm az RTT-ket:
cat $b/* | grep '64 bytes from' | awk '{print $7}' | sed 's/time=//g' >>
$k/$l/rtt.txt
#Kigyűjtöm a küldött, a fogadott csomagokat:
cat $b/* | grep loss | awk '{print $1 "\t" $4}' >> $h/cs.txt
done

#Kigyűjtöm a küldött és a fogadott csomagok számát:
awk '{ SENT+=$1; RECEIVED+=$2 } END {print "Kuldott: " SENT "\tFogadott: "
RECEIVED}' $h/cs.txt > $h/csomagszam.txt

#Kigyűjtöm a mérési időt (csak azt):
echo $(cat meresideje.txt | grep :) > meresideje.txt

#Mérés ideje másodpercben: (másodpercek) + (percek másodpercben) = [sec]:
echo $(cat meresideje.txt | sed 's/[0-9]*://g') + $(echo $(( $(cat meresideje.txt |
sed 's/:[0-9]*.[0-9]*//g')*60))) | bc -l > $h/meresideje.txt

cd $h
#Csomagvesztés [%]:
awk '{print 100-($4/$2)*100}' csomagszam.txt > csomagvesztes.txt

#Összesen forgalmazott csomag [db szám]:
awk '{print $2+$4}' csomagszam.txt > forgcsomag.txt

#Forgalom sebessége [db csomag/sec]:
echo $(cat forgcsomag.txt) / $(cat meresideje.txt) | bc -l > csomagsebes.txt
```

38. ábra Eredményeket feldolgozó és részben kiértékelő felki.sh script.

A vmstat-tal készített terhelés kimutatásokat is shell scriptek dolgozták fel. Érdekesképp megemlítem, hogy a vmstat program kimenete különböző Linux, BSD és NetBSD rendszerekben. Ezért a terhelési adatok feldolgozásához három feldolgozó programot kellett készítenem. A kiertekeles.sh script lefutása után a megfelelő feldolgozó scriptet kézzel indítottam. A script a terheltségi értékeket hozzáfűzte a korábban előállított eredményfájlokhoz. A programok ismertetése a függelék 9.3. alfejezetében található.

5.4. Eredmények

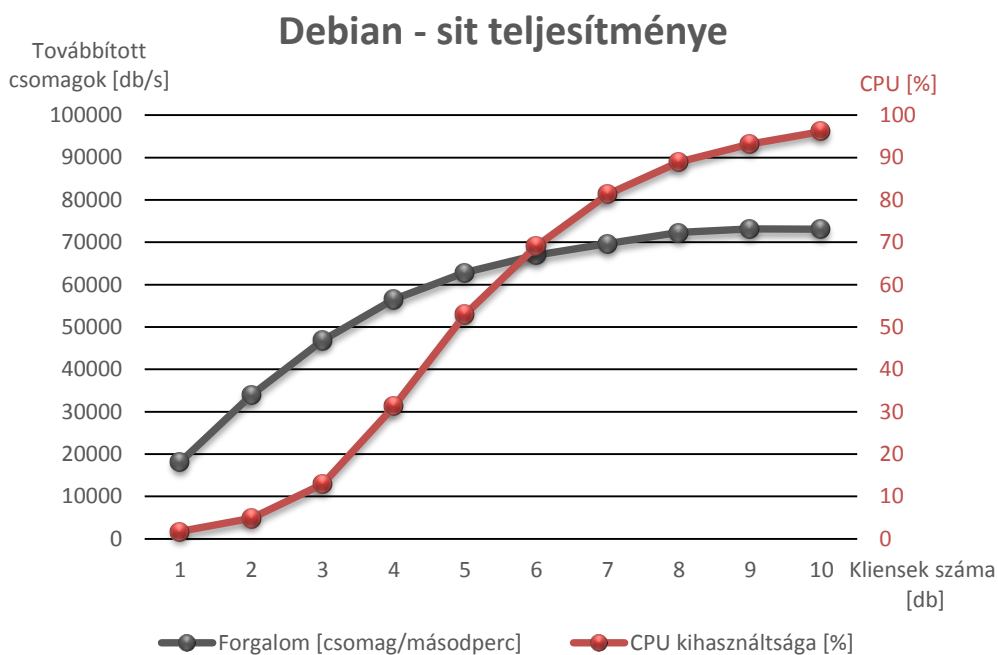
A mérések eredményét az 1. táblázathoz hasonló módon ábrázoltam minden egyes implementáció esetén. A táblázat felépítése a következő:

1. sor Az első sorban a terhelést létrehozó kliensek száma olvasható. A számok tehát azt is jelentik, hogy egyidejűleg hányszor került futtatásra a 22. ábrán bemutatott mérőscriptem.
2. sor A második sorban az elküldött kérések és a válaszul visszaérkezett csomagok viszonyát, azaz a csomagvesztés értékét láthatjuk százalékos arányban. A számértékek tulajdonképpen azt jelölik, hogy az elküldött kérések hány százalékára nem érkezett válasz. A mérések során kliensenként (ideális esetben, ha nincs veszteség) $256 \cdot 64 \cdot 8 \cdot 8 = 1\,048\,576$ db eredmény keletkezik, a végeredményhez ezeket az értékeket átlagoltam.
3. sor A táblázat harmadik sora az átlagos válaszidőt tartalmazza kliensenként, amely megmutatja, hogy mennyi idő alatt érkezik válasz egyetlen kérésre. Megfigyelhető, hogy a terhelés növekedtével a válaszidő is nő, ami logikus, hiszen a relaynek azonos idő alatt egyre több kérést kell kiszolgálnia.
4. sor A negyedik sorban a válaszidők szórása található, amely az egyes válaszidők közti különbözőség nagyságát mutatja meg.
5. sor Az ötödik sorban a mérések során adódó legnagyobb válaszidő olvasható.
6. sor A hatodik sorban a processzorkihasználtság pillanatnyi értékeinek átlaga található.
7. sor A táblázat hetedik sorában a processzorterhelés értékeinek szórása olvasható. Minél kisebb ez az érték, annál egyenletesebb a relay terhelése az adott mérés során.
8. sor A nyolcadik sorban a felhasznált memória értékeket láthatjuk. A foglalt memória méretét a mérés megkezdése előtt rögzített legnagyobb szabad memória érték és a mérés során rögzített legkisebb rendelkezésre álló memória érték különbségéből számoltam. A relay számítógépen más rendszerfolyamatok is futottak, melyek memóriát fogyasztottak, mivel a bekapcsoláskor különböző időközönként indulnak ezek a folyamatok, a ténylegesen felhasználható memória-méret folyamatosan változik, ezért 100%-os pontossággal nem adható meg a mérések során lefoglalt memória nagysága.
9. sor A kilencedik sorban a relayen áthaladó csomagforgalom átlagos sebessége látható, ami egyúttal megegyezik az egy másodperc alatt kiszolgált kérések számának kétszeresével.
10. sor Az utolsó, azaz a táblázat tizedik sora tartalmazza, hogy az adott mérés végeztével hány millió csomag haladt át a relayen. (Az összesen forgalmazott csomag és a forgalom hányadosával kiszámítható az aktuális mérés időtartama másodpercben.)

5.4.1. Debian 7.1.0_x86 – sit

1. táblázat Debian Linux – sit 6to4 relay mérési eredmények.

Kliensek száma		1	2	3	4	5	6	7	8	9	10
Csomagvesztés [%]		0,002	0,006	0,008	0,013	0,020	0,035	0,035	0,037	0,048	0,061
Válaszidő [ms]	Átlag	0,287	0,353	0,445	0,566	0,710	0,868	1,043	1,209	1,411	1,626
	Szórás	0,174	0,248	0,335	0,423	0,509	0,588	0,685	0,772	0,832	0,864
	Maximum	27,900	28,400	28,500	28,900	29,400	30,700	31,100	34,100	32,800	39,600
CPU kihasználtsága [%]	Átlag	1,756	4,821	12,933	31,243	52,964	69,049	81,319	88,941	93,206	96,132
	Szórás	1,944	2,811	5,619	12,215	16,379	16,493	12,690	9,817	5,289	7,388
Memória foglalása [KB]		10,855	10,418	10,363	10,594	10,824	10,996	10,855	10,994	10,828	11,137
Forgalom [csomag/másodperc]		18051	33953	46856	56534	62853	66947	69663	72304	73129	73050
Összesen forgalmazott csomag [$x \cdot 10^6$ db]		2,097	4,194	6,291	8,388	10,485	12,581	14,678	16,774	18,870	20,965



39. ábra A Debian Linux – sit 6to4 relay teljesítménye.

A mérések alapján kijelenthető, hogy:

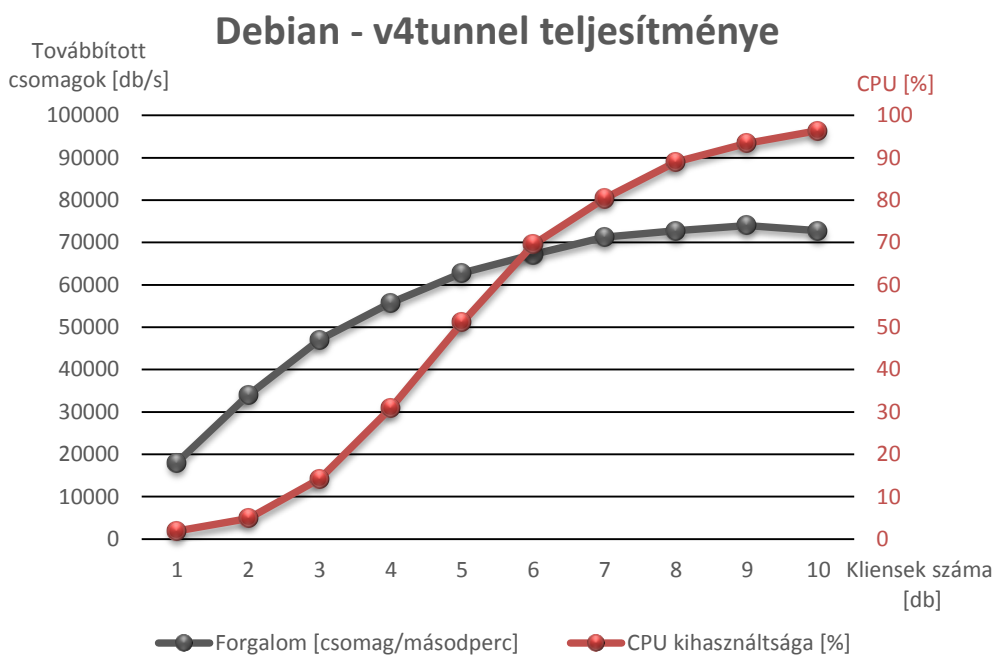
- A csomagvesztés nagysága a hat- és a hét klienses mérést kivéve növekvő, értéke tíz kliens egyidejű terhelése esetén is viszonylag alacsony: 0,061 %.
- A válaszidők átlaga, szórása, maximuma a terheléssel növekszik, az átlagos válaszidő tíz kliens esetén sem haladja meg az 1,63 milliszekundumot.

- A processzorterhelés folyamatosan növekvő, de nem lineáris jellegű. A 4, 5, 6, 7-es mérési pontban a szórás magasabb, mint 12 %, ami ingadozó processzor-igénybevételre utal.
- A mérések során kis ingadozással körülbelül 10,5 és 11,1 KB közötti értéken maradt a memóriaszükséglet, a memóriakihasználtság tíz kliens terhelése esetén volt a maximális, ám még ekkor is nagyon alacsonynak tekinthető.
- A forgalom nagysága növekvő. Az implementáció teljesítőképessége kilenc kliens esetén a legnagyobb, a processzorterhelés ekkor 93,206 % volt. További kliens bevonásával már nem növekszik a teljesítőképesség. Ez köszönhető részben az **5.1.3.** pontban említett erőforrás kiosztásnak, részben annak, hogy a terhelést létrehozó script egymást követő ping6 parancsai csak akkor kerülnek végrehajtásra, ha az előző parancs lefutott, tehát csak akkor generálódik újabb kérés, ha az előzőleg elindított kérésekre megérkeznek a válaszok.

5.4.2. Debian 7.1.0_x86 – v4tunnel

2. táblázat Debian Linux – v4tunnel 6to4 relay mérési eredmények.

Kliensek száma		1	2	3	4	5	6	7	8	9	10
Csomagvesztés [%]		0,003	0,006	0,008	0,011	0,018	0,033	0,036	0,039	0,047	0,060
Válaszidő [ms]	Átlag	0,287	0,351	0,444	0,579	0,709	0,865	1,007	1,198	1,389	1,632
	Szórás	0,174	0,251	0,334	0,428	0,508	0,588	0,690	0,776	0,842	0,887
	Maximum	27,800	27,700	28,700	29,200	24,000	30,100	31,300	35,100	33,900	32,800
CPU kihasználtsága [%]	Átlag	1,915	4,886	14,202	30,927	51,121	69,555	80,392	89,042	93,441	96,444
	Szórás	1,727	3,037	6,871	12,412	16,664	14,790	13,807	10,084	7,934	5,461
Memória foglalása [KB]		10,664	10,559	10,910	10,555	10,855	10,728	10,730	10,602	11,102	11,438
Forgalom [csomag/másodperc]		18083	34062	47079	55828	62788	67181	71315	72759	74025	72792
Összesen forgalmazott csomag [$x \cdot 10^6$ db]		2,097	4,194	6,291	8,388	10,485	12,581	14,677	16,774	18,870	20,965



40. ábra A Debian Linux – v4tunnel 6to4 relay teljesítménye.

A mérések alapján kijelenthető, hogy:

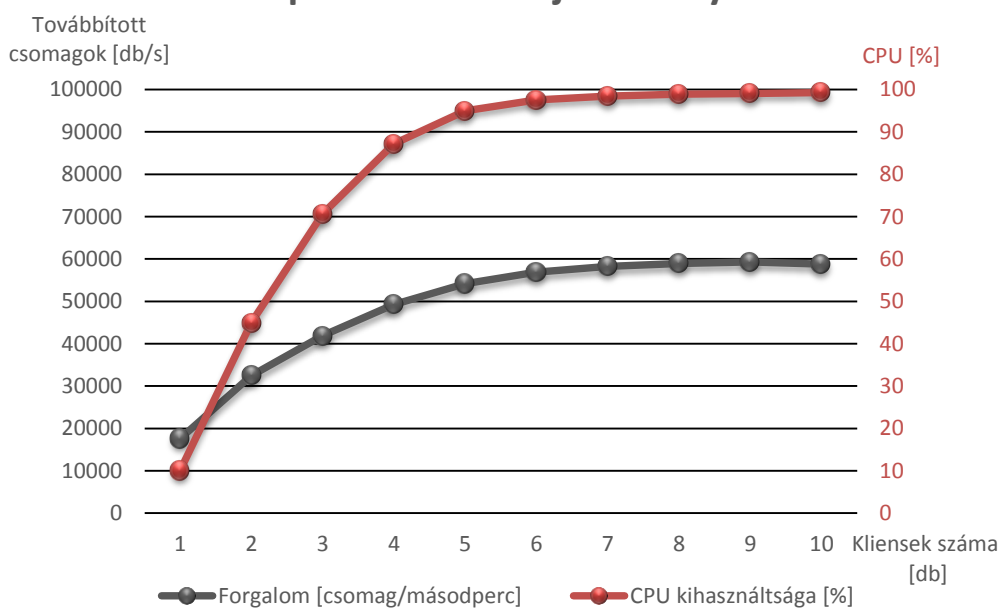
- A csomagvesztés nagysága folyamatosan növekvő, de értéke tíz kliens egyidejű terhelése esetén is viszonylag alacsony marad.
- A válaszidők átlaga és szórása a terheléssel növekszik, a válaszidő tíz kliens esetén a maximális.
- A processzorterhelés itt is folyamatosan növekvő, de nem lineáris jellegű. A sit és a v4tunnel erőforrásigénye közt nincs szignifikáns különbség. A 4, 5, 6, 7-es mérési pontban a 12 %-nál magasabb szórás érték jelzi, hogy a processzorkihasználtság nem volt igazán egyenletes.
- A memóriaszükséglet kis mértékben ingadozik, a memóriakihasználtság tíz kliens terhelése esetén volt a legnagyobb: 11,438 KB. A rendelkezésre álló memóriamérethez viszonyítva azonban ez elhanyagolható mennyiség.
- A forgalom nagysága növekvő kilenc kliensig. Az implementáció teljesítőképessége ebben a pontban a maximális. A teljesítőképesség a sitnél említett okok miatt nem növelhető. A legnagyobb processzorterhelés 96,444 % 10 kliens alkalmazása esetén, ekkor a forgalom sebessége az előző ponthoz képest már érzékelhetően lecsökken.

5.4.3. OpenWrt (Attitude Adjustment) 12.09_x86 – sit

3. táblázat OpenWrt – sit 6to4 relay mérési eredmények.

Kliensek száma		1	2	3	4	5	6	7	8	9	10
Csomagvesztés [%]		0,004	0,006	0,007	0,013	0,018	0,026	0,036	0,064	0,079	0,089
Válaszidő [ms]	Átlag	0,314	0,402	0,568	0,733	0,909	1,118	1,358	1,616	1,873	2,160
	Szórás	0,161	0,239	0,330	0,420	0,508	0,583	0,652	0,705	0,773	0,829
	Maximum	25,000	25,300	25,500	25,500	26,500	27,100	27,000	27,100	27,300	28,100
CPU kihasználtsága [%]	Átlag	10,067	45,015	70,713	87,188	94,979	97,540	98,467	98,916	99,066	99,288
	Szórás	3,188	5,593	5,828	9,376	7,954	7,462	4,991	4,567	4,824	4,410
Memória foglalása [KB]		10,316	10,414	10,359	10,727	10,469	10,324	10,746	10,492	10,066	10,469
Forgalom [csomag/másodperc]		17595	32488	41906	49270	54196	56920	58272	58928	59332	58763
Összesen forgalmazott csomag [$x \cdot 10^6$ db]		2,097	4,194	6,291	8,388	10,485	12,581	14,677	16,772	18,867	20,962

OpenWrt - sit teljesítménye



41. ábra Az OpenWrt – sit 6to4 relay teljesítménye.

A mérések alapján kijelenthető, hogy:

- A csomagvesztés nagysága növekvő, maximális értéke tíz kliens egyidejű terhelése esetén 0,089 %, amely az eddigi Linux implementációkéhoz képest a legmagasabb.
- A válaszidők átlaga, szórása a terheléssel növekszik, az átlagos válaszidők nagyobb mértékben változnak a korábbiakhoz képest. A maximális válaszidők viszont alacsonyabbak

a Debian értékeinél és a terhelés függvényében – az előzőekkel ellentétben – alig növekednek. Tíz kliens esetén a legnagyobb a válaszidő, amely jelen esetben 28,1 ms, ami nem sokkal nagyobb, mint az egy kliens esetén mért 25 ms.

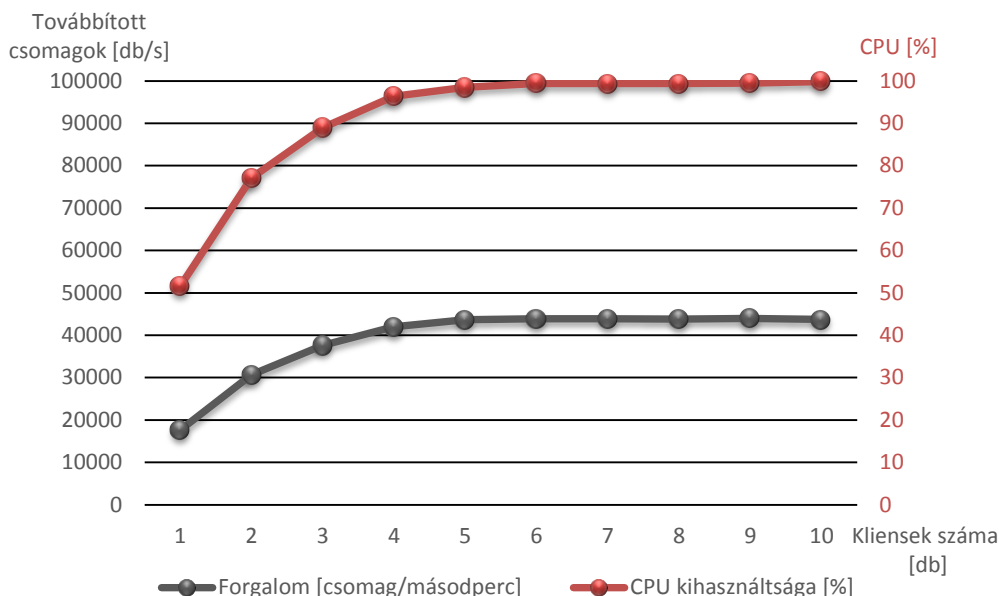
- A processzorterhelés két kliens esetén az egy kliens terhelésének 4,5-szerese, majd a görbe meredeksége csökken és a szabad kapacitás elfogyásával telítésbe megy. A szórásértékek 10 % alatt maradnak, a processzor igénybevétele tehát egyenletesebb.
- A memóriakihasználtság durván 10,5 KB körüli. Jelen esetben is ingadozás figyelhető meg az értékek között. Érdekes módon a memóriaszükséglet nem tíz kliens terhelése esetén volt a maximális, ez valószínűleg a magasabb processzorterhelésnek is köszönhető, ami miatt a vmstat program az aktuális legkisebb vagy legnagyobb szabad memóriaértéket nem volt képes fájlba menteni és ebből adódóan pontatlan érték került kiszámításra.
- A kilenc klienses mérési pontig a forgalom nagysága növekvő. Az implementáció teljesítőképessége kilenc kliens esetén a legnagyobb, a processzorterhelés ekkor 99,066 %, ami körülbelül 3 %-kal magasabb az eddigieknél. A teljesítőképességet jól szemlélteti, hogy az egy másodperc alatt maximálisan továbbított csomagok száma körülbelül 7 %-kal kevesebb a korábbi két Linux implementációénál. További kliens bevonásával már nem növekszik a teljesítőképesség az 5.4.1-ben tisztázott okok miatt. A tíz klienses mérési pontban a forgalom sebessége a v4tunneléhez hasonló mértékben lecsökken.

5.4.4. FreeBSD 9.1_x86 – stf

4. táblázat FreeBSD – stf 6to4 relay mérési eredmények.

Kliensek száma		1	2	3	4	5	6	7	8	9	10
Csomagvesztés [%]		0,013	0,008	0,010	0,012	0,013	0,015	0,017	0,018	0,019	0,019
Válaszidő [ms]	Átlag	0,315	0,456	0,681	0,941	1,268	1,637	2,011	2,385	2,740	3,126
	Szórás	0,111	0,171	0,314	0,404	0,450	0,457	0,463	0,466	0,480	0,490
	Maximum	22,200	9,220	12,800	15,400	17,600	18,100	18,800	18,500	19,600	19,400
CPU kihasználtsága [%]	Átlag	51,525	77,110	88,994	96,380	98,482	99,435	99,395	99,371	99,462	99,859
	Szórás	6,899	5,140	6,465	7,398	7,593	3,447	5,336	6,445	5,971	0,475
Memória foglalása [KB]		0,008	0,012	0,012	0,273	0,395	0,398	0,445	0,406	0,500	0,492
Forgalom [csomag/másodperc]		17594	30656	37613	41982	43681	43892	43875	43819	43970	43737
Összesen forgalmazott csomag [$x \cdot 10^6$ db]		2,097	4,194	6,291	8,388	10,485	12,582	14,679	16,776	18,873	20,970

FreeBSD - stf teljesítménye



42. ábra A FreeBSD – stf 6to4 relay teljesítménye.

A mérések alapján kijelenthető, hogy:

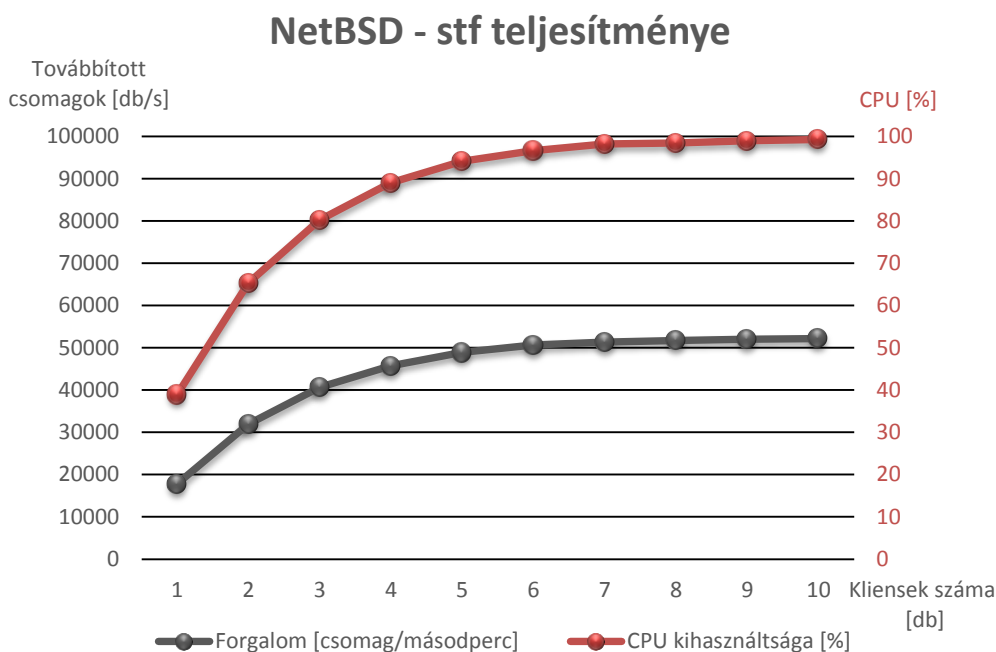
- A csomagveszteség nagysága ingadozó. A legnagyobb csomagveszteség értéke 0,019 %, amely az eddigi Linux implementációkéhoz képest jóval alacsonyabb, viszont, viszonylag alacsony terhelésnél (1-3 kliens) a csomagvesztési arány itt a legmagasabb.
- A válaszidők átlaga, szórása a terheléssel növekszik, az átlagos válaszidők az eddigiek közül itt nőnek a legnagyobb mértékben (1 kliens: 0,315 ms; 10 kliens: 3,216 ms). A maximális válaszidők viszont a legalacsonyabbak (minden esetben 20 ms alatt vannak), bár erősen ingadoznak. Az átlagos válaszidőt tekintve viszont megállapíthatjuk, hogy nagy terhelés mellett az stf implementáció jóval lassabb működésre képes a korábban bemutatottakhoz képest (10 kliensnél 3,216 ms az átlagos válaszidő, míg mindkét Debian implementáció esetén 1,6 ms körül volt, és még OpenWrt-nél is 2,2 ms alatt maradt).
- A processzorterhelés nagysága a 6 klienses mérési pontig növekvő, a későbbi szakaszon lecsökken a görbe meredeksége. Az stf processzor-igénybevétele jóval magasabb az eddigieknél, egyetlen kliens már körülbelül 51,5 %-os kihasználtságot eredményez, tíz kliens egyidejű terhelése pedig közel 100 %-os igénybevételnek felel meg. A processzor-kihasználtságok szórása 8 % alatti, amiből egyenletesen eloszló terhelésre lehet következtetni.
- A memóriakihasználtság majdnem végig növekvő, de rendkívül alacsony. A legnagyobb értéke 0,5 KB. A memóriaszükséglet nem tíz kliens terhelése esetén volt a maximális, ez valószínűleg ismét a magasabb processzorterhelésnek is köszönhető, ami miatt a vmstat program az aktuális legkisebb vagy legnagyobb szabad memóriaértéket nem volt képes fájlba menteni és ebből adódóan pontatlan érték került kiszámításra.

- Az öt klienses mérési pontig a forgalom nagysága növekvő. Ebben a pontban a relay láthatóan elérte a maximális másodpercenkénti csomagsebességét, érdemben már nem képes tovább nőni. A későbbi szakaszon a relay teljesítőképessége a túlterhelés miatt minimálisan ingadozik. Az előző implementációkhoz képesti alacsony teljesítőképességet jól szemlélteti, hogy az egy másodperc alatt maximálisan továbbított csomagok száma nem éri el a negyvennégyezret.

5.4.5. NetBSD 6.1.2_x86 – stf

5. táblázat NetBSD – stf 6to4 relay mérési eredmények.

Kliensek száma		1	2	3	4	5	6	7	8	9	10
Csomagvesztés [%]		0,011	0,016	0,028	0,047	0,056	0,051	0,044	0,038	0,031	0,031
Válaszidő [ms]	Átlag	0,301	0,418	0,603	0,823	1,061	1,326	1,620	1,908	2,210	2,519
	Szórás	0,186	0,236	0,319	0,403	0,499	0,571	0,631	0,681	0,707	0,712
	Maximum	5,760	11,500	13,600	16,900	18,900	21,400	21,100	21,700	22,200	24,300
CPU kihasználtsága [%]	Átlag	38,957	65,382	80,290	89,055	94,130	96,671	98,259	98,435	99,020	99,306
	Szórás	4,519	6,229	9,771	3,769	5,878	6,664	3,759	5,751	6,243	4,642
Memória foglalása [KB]		0,016	0,027	0,055	0,148	0,191	0,203	0,695	0,336	0,480	0,180
Forgalom [csomag/másodperc]		17797	31937	40639	45745	48913	50686	51345	51750	52062	52202
Összesen forgalmazott csomag [$x \cdot 10^6$ db]		2,097	4,194	6,291	8,387	10,483	12,580	14,677	16,774	18,871	20,968



43. ábra A NetBSD – stf 6to4 relay teljesítménye.

A mérések alapján kijelenthető, hogy:

- A csomagveszteség nagysága az öt klienses mérési pontig emelkedik, majd innentől az utolsó előtti mérési pontig csökken, kiszámíthatatlan a rendszer viselkedése, irréalisan, növekvő terhelés mellett egyre jobban teljesít a relay. A legnagyobb csomagveszteség értéke 0,056 %, ami az eddigi implementációkéhoz képest közepes, elfogadható érték, ám megbízhatóság szempontjából eddig ez a leginstabilabb implementáció-operációs rendszer páros.
- A válaszidők átlaga, szórása, maximális értéke a terheléssel növekszik, az átlagos válaszidők az OpenWrt-jéhez hasonló mértékben változnak. Az átlagos válaszidő tíz kliens együttes terhelése esetén a legnagyobb, amely jelen esetben 2,519 milliszekundum. Az stf implementáció és a NetBSD a Linux implementációkhoz viszonyítva lassabb működésre képes.
- A processzorterhelés a 7 klienses mérési pontig növekvő, a 7-es és a 8-as pontban a terhelés közel megegyező nagyságú. A rendszer további többlet terhelése mellett a processzorkihasználtság még körülbelül 1 %-kal emelkedik. Az stf processzor-igénybevétele viszonylag egyenletesnek mondható, mert a szórásértékek nem haladják meg a 10 %-ot. A processzor-erőforrásigény egy kliens esetén már meghaladja a 38 %-ot, a rendszer maximális terhelése esetén pedig 99,306 % a processzor terhelése, ezek alapján a NetBSD-n minimalisan, de kevesebb az stf implementáció erőforrásigénye, mint a FreeBSD operációs rendszeren.
- A memóriakihasználtság a 7-es mérési pontig növekvő, de nagyon alacsony. A rendszer túlterhelése esetén, érdekes módon csökkenni kezd a memóriaszükséglet. A legnagyobb

értéke 0,695 KB, de ez még mindig elenyészően kicsi a felhasználható memóriamérethez viszonyítva.

- A relay teljesítőképessége a túlterhelés miatt nem ingadozik. Az implementáció teljesítőképessége tíz kliens terhelése, tehát a maximális processzorkihasználtság esetén a legnagyobb, bár a NetBSD áteresztőképessége szabályszerűen változik a terhelések függvényében, a teljesítménye mégis gyengének mondható, mert az egy másodperc alatt maximálisan továbbított csomagok száma nem éri el az ötvenháromezret.

5.4.6. OpenBSD 5.3_x86 – gif

Ahogy azt az 5.2.6. alfejezetben tisztáztam, az OpenBSD és a gif implementáció nem használható 6to4 relay mechanizmus megvalósítására. Kezdeti sikereim miatt és a végleges következtetések leszűrése érdekében, kíváncsiságból teszteltem az általam készített konfigurációt egyetlen klienssel (technikai okok miatt csak egyel, lásd: 5.2.6.), a mérés eredményét a 6. táblázat tartalmazza.

6. táblázat OpenBSD – gif 6to4 relay mérési eredmények.

Kliensek száma		1
Csomagvesztés [%]		0,156
Válaszidő [ms]	Átlag	0,651
	Szórás	0,179
	Maximum	6,000
CPU kihasználtsága [%]	Átlag	16,491
	Szórás	20,779
Memória foglalása [KB]		78,074
Forgalom [csomag/másodperc]		4807,2
Összesen forgalmazott csomag [$x \cdot 10^6$ db]		2,096

A mérés alapján kijelenthető, hogy:

- A csomagvesztés nagysága számottevő, majdnem duplája az OpenWrt legnagyobb értékének. A rendszer csomagvesztése csupán egyetlen kliens terhelése esetén nagyobb az összes eddig tesztelt implementáció maximális, azaz a legrosszabb értékénél.
- Az átlagos válaszidő önmagában nem lenne túlságosan magas, de tekintve, hogy jelen esetben csupán egyetlen számítógép terhelte a relayt, a 0,651 ms a rendszer lassú működését

szemlélteti. A válaszidők szórása és maximális értéke viszonylag alacsony.

- A processzorterhelés az eddigi egy klienses mérésekhez képest közepesnek mondható, ám a 20 % feletti szórás egyenetlen, nagymértékben ingadozó processzorkihasználtságra utal.
- A memóriakihasználtság az eddig mért legmagasabb érték (Debian – v4tunnel, 11,438 KB) több mint 6,5-szerese. Mivel azonban több, mint egy nagyságrenddel több szabad memória áll rendelkezésre, az implementáció teljes értékű működése során, még ez a memória-szükséglet sem jelentene számottevő problémát.
- A rendszer áteresztőképessége nagyon alacsony, a több mint 2 millió csomag célba juttatásához az eddigiekben tapasztaltnál sokkal több időre volt szükség.

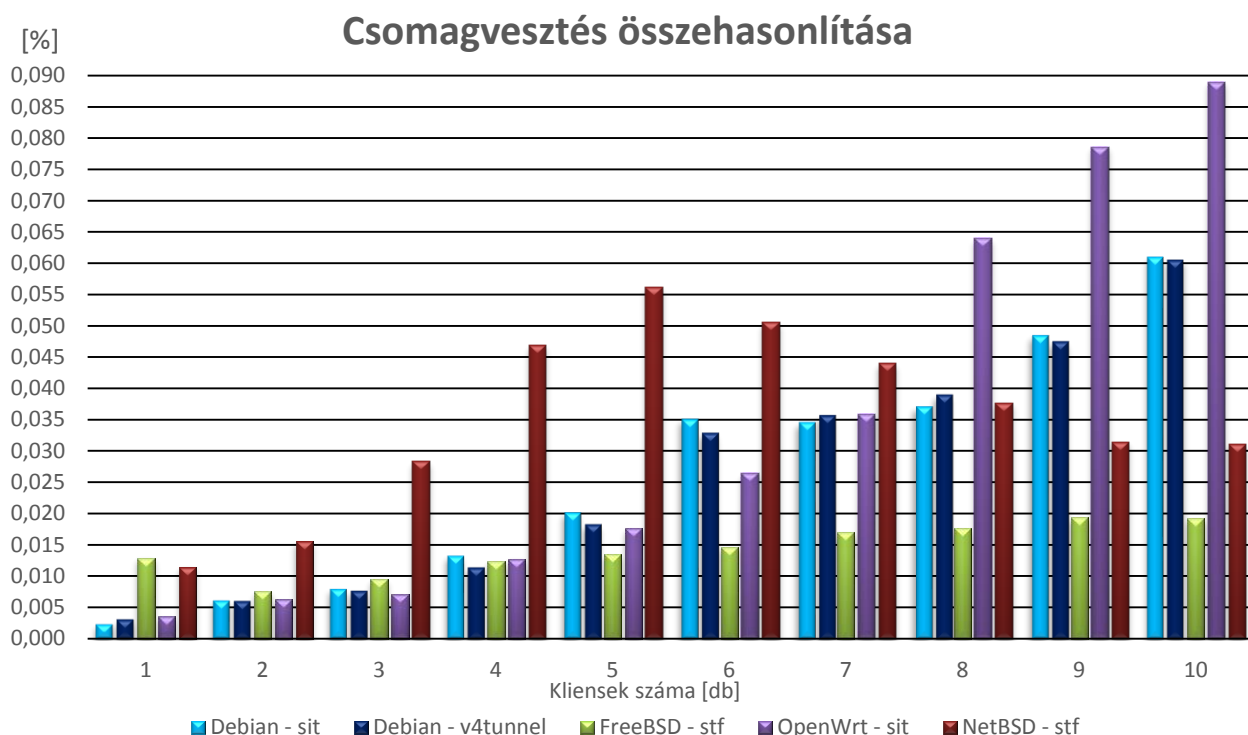
Az eredmények alapján az OpenBSD és a gif implementáció kombinációja teljes mértékben alkalmatlan 6to4 relay megvalósítására. Minden tekintetben gyenge teljesítménye miatt, még ha a rendszert át is lehetne alakítani nagyszámú klienspark kéréseinek kiszolgálására, az OpenBSD-t és a gif implementációt senkinek, semmilyen körülmények között nem ajánlom 6to4 relay router üzemeltetése céljából.

Az OpenBSD-vel a dolgozat hátralévő részében érdemben tovább nem foglalkozom.

5.5. Összehasonlítás

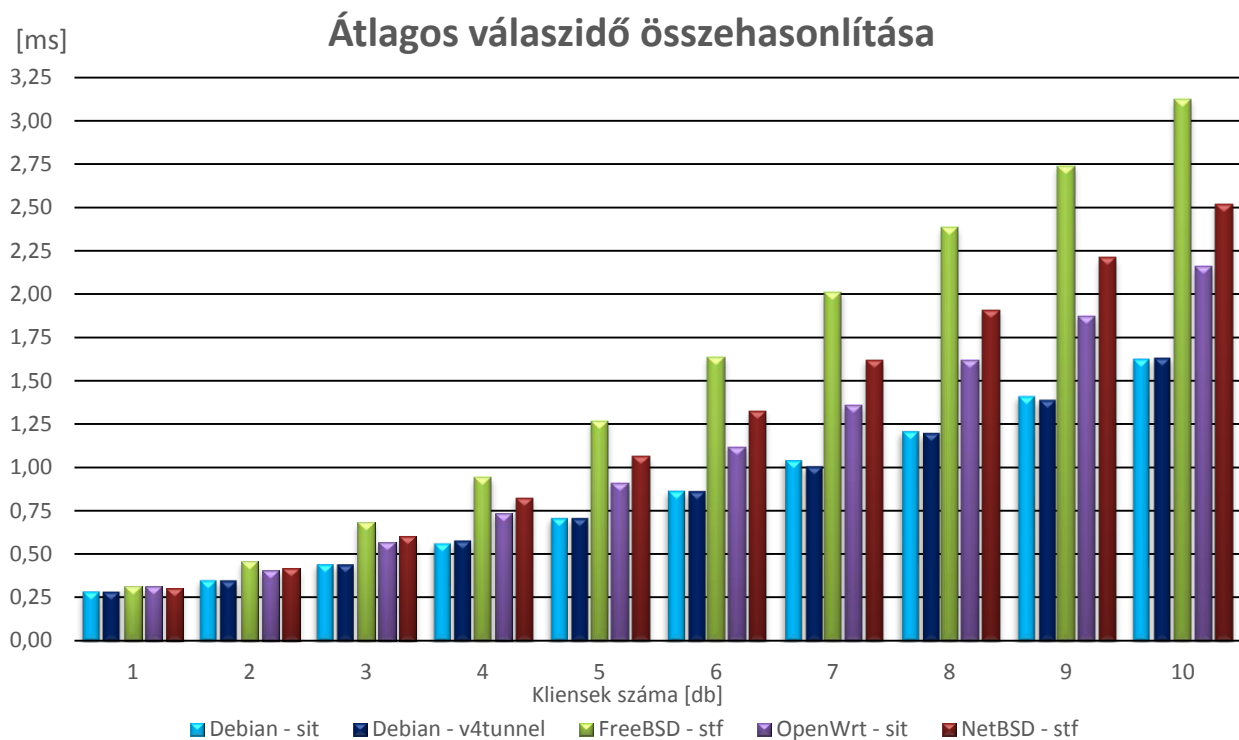
Véleményem szerint ahhoz, hogy végleges eredményt hirdethessek, az egyes implementációk teljesítőképességét különböző szempontok alapján össze kell hasonlítanom. Az egyes tulajdonságok szemléltetéséhez összefoglaló diagramokat készítettem, melyek a 44., 45., 46., 47. ábrán láthatók.

Az egyes implementációk megbízhatóságát leginkább a csomagveszteségi rátájuk jellemzi, az erről készített diagramot a 44. ábra tartalmazza. Ahogy az ábrán látható, a Linux implementációk csomagvesztesége a terhelés arányában növekvő jellegű. A Linux implementációk közül a v4tunnel teljesít a legjobban. A BSD implementáció, az stf, kezdetben magasabb értéket produkál. A NetBSD esetén a csomagveszteség nagysága irreálisan változik. A FreeBSD rendelkezik egyetlen kliens esetén a legnagyobb csomagvesztéssel, ám a terhelés növekedtével csupán minimálisan változik csomagvesztesége, tíz kliens egyidejű terhelése esetén ez a rendszer teljesít a legmegbízhatóbban. Maximális terhelés esetén az OpenWrt teljesít a legrosszabbul, a FreeBSD-hez képest több, mint 4,5-szer annyi csomag vesz el áthaladásuk során a relayen.



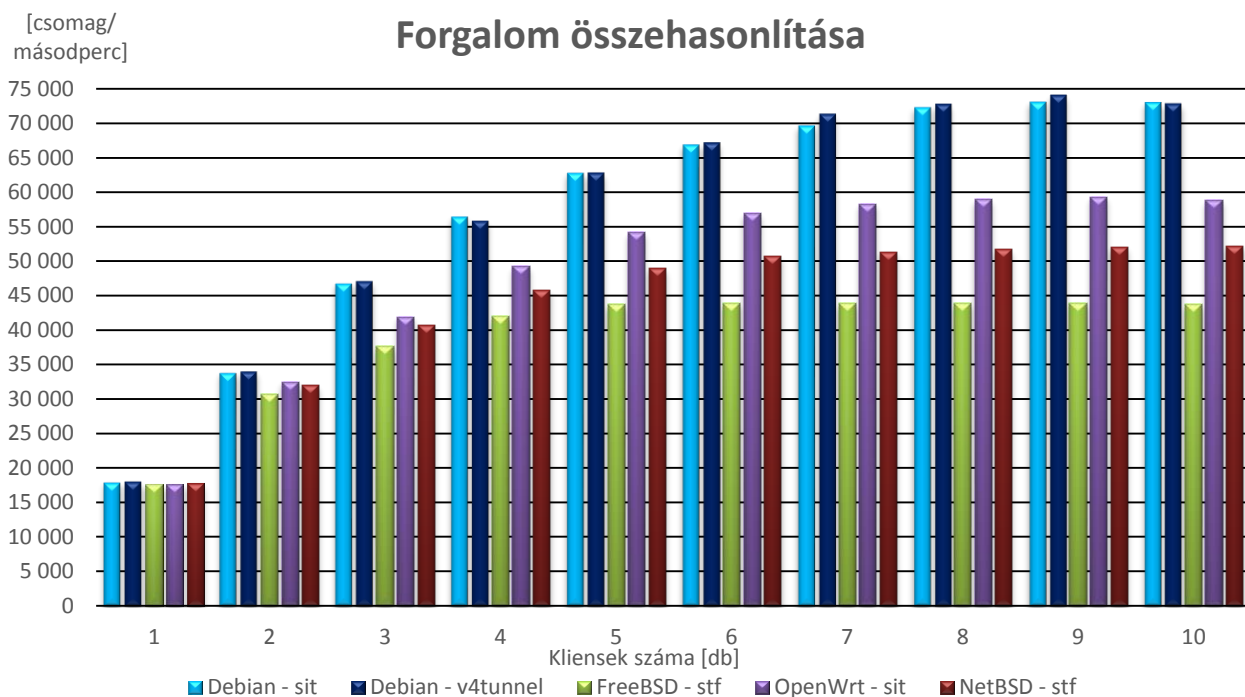
44. ábra 6to4 relay implementációk megbízhatóságának összehasonlítása.

Az implementációk és az operációs rendszerek sebessége a válaszidők átlagos értékével jellemezhető. A sebességek összehasonlítása a 45. ábrán szerepel. Mindegyik implementációra igaz, hogy a terhelés változásának függvényében változik a csomagtovábbítás sebessége is, minél nagyobb a terhelés, annál nagyobb időt vesz igénybe a válasz csomagok visszajuttatása a kliensekhez. Általánosságban elmondható, hogy a Linux implementációk gyorsabbak a BSD implementációknál. A leggyorsabb relay implementáció a sit és a v4tunnel, sebességük csupán minimálisan tér el egymástól. A leglassabb implementáció az stf a FreeBSD operációs rendszeren, tíz kliens terhelése esetén a sit és a v4tunnel körülbelül kétszer gyorsabb.



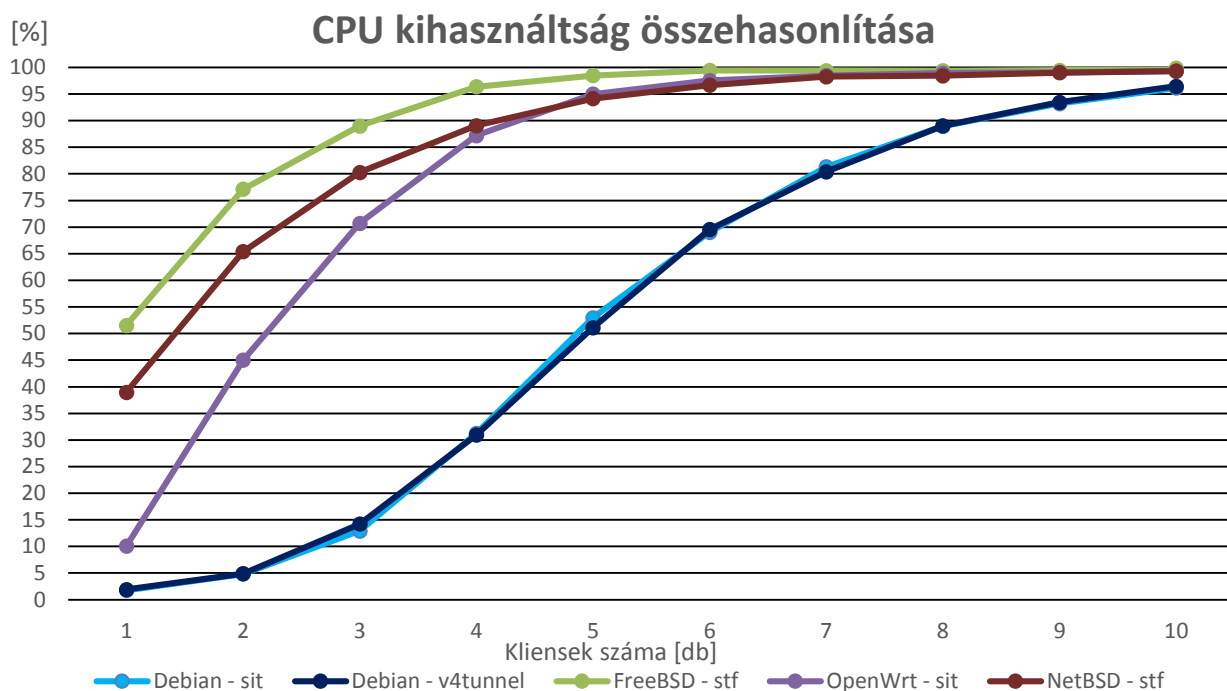
45. ábra 6to4 relay implementációk sebességének összehasonlítása.

Az implementációk teljesítőképességét jól tükrözi a relayeken áthaladó forgalom, azaz, hogy másodpercenként hány darab csomag továbbítására képes a rendszer adott terhelés mellett. Az erről készített diagramom a 46. ábrán látható. A terhelés nagyságának függvényében a forgalom sebessége is változik. Egyetlen kliens terhelése esetén közel azonosan teljesítenek az implementációk. Megfigyelhető, hogy a legnagyobb forgalom általában kilenc kliens együttes terhelése esetén adódott. A rendszer túlterhelésekor egyetlen esetben sem következik be drasztikus változás. Ismét igaz, hogy a Linux implementációk jobban teljesítenek. A két Debian implementáció, a sit és a v4tunnel teljesítőképessége vetekszik egymással, az eltérés csupán minimális, ezek teljesítenek a legjobban, maximálisan, körülbelül 74000 csomagot képesek továbbítani másodpercenként. A leggyengébb az stf és a FreeBSD együttes teljesítménye, kilenc kliens esetén a Linux implementációkhoz képest 41 %-kal kevesebb csomagot képes útba irányítani a relay.



46. ábra 6to4 relay implementációk teljesítőképességének összehasonlítása.

Utolsó szempontként az implementációk processzor-erőforrásigényét hasonlítottam össze, amely a 47. ábrán látható. Az OpenWrt, a FreeBSD és a NetBSD operációs rendszerek esetén a processzorkihasználtság a kliensek számának növelésével ugrásszerűen változik, a 7 klienses mérési pontig növekvő. A Debian Linux esetén maximális terhelést tíz kliens produkál, míg a többi rendszer esetén, általában elegendő 7 kliens együttes alkalmazása, hogy elérjem a 99 % körüli processzor-terheltség értéket. A sit és a v4tunnel szinte egyformán használja a processzort. A legnagyobb erőforrásigénye a FreeBSD-stf implementációnak van, egy darab kliens terhelése már több, mint 51 %-os processzor-terhelést jelent. A 99 % feletti processzorkihasználtságot pedig már hat kliens együttes alkalmazásakor eléri az implementáció, azonban a relay túlterhelése esetén sem omlasztja össze az operációs rendszert. A leghatékonyabbnak a Debian Linux tekinthető, mivel egyetlen kliens esetén kevesebb, mint 2 % processzor-terhelést okoz a 6to4 mechanizmus megvalósítása, továbbá a maximálási terhelés esetén is marad 3 % szabad processzorkapacitás a rendszer számára. A többi implementáció a legnagyobb terhelés esetén közel 100 %-os processzor-igénybevételt produkál.



47. ábra 6to4 relay implementációk erőforrásigényének összehasonlítása.

5.6. Teljesítőképesség átlagos PC-n

A végeredmény magabiztos kijelentéséhez, véleményem szerint az implementációkat még további teszteknek kell alávetnem olyan körülmények között is, ahol a teljesítőképességeket nem befolyásolja a hardver. A valóságban, egy "éles" rendszerben nem egy "ösrégi" Pentium III számítógép fog előfizetőket kiszolgálni, így meg kellett győződnöm róla, hogy az implementációk akkor is hasonlóképpen teljesítenek-e, ha a célfeladatra szánt számítógép sokkal erősebb hardver-konfigurációval rendelkezik.

Az általam választott számítógép, mely ugyan lényegesen gyorsabb a teszt számítógépnél, a mai felfogás szerint már egy lassúnak tekinthető, korosodó asztali számítógép, amelyet viszont nagyobb valószínűséggel használnék kiszolgálóként az ár/érték aránya miatt.

A számítógép tulajdonságai a következők:

- ASUS M2N-E SLI Revision 1201 alaplap
- AMD Athlon 64 X2 4200+ (2,2 GHz) duplamagos processzor
- 2 db 1GB DDR2 (667 MHz) memória
- 2 db TP-LINK TG-3269 REV 3.0 Gigabit PCI hálózati kártya

Ahogy látható, a gigabites hálózati csatlók jelen esetben is a korábban használt TP-Link kártyák, hogy a hálózat teljesítőképessége azonos legyen a korábbiéval. A mérési elrendezés megegyezik a 21. ábrán bemutatottal. Az operációs rendszereken semmiféle módosítást nem hajtottam végre, az összes konfiguráció megegyezik az 5.2. alfejezetben közöltekkel. A FreeBSD kivételével csupán

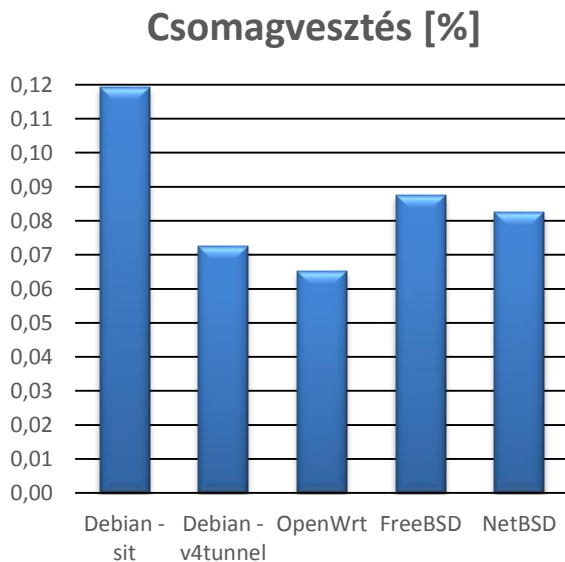
kicseréltem a háttértárat a számítógépben a Pentium számítógépnél használt merevlemezre. A FreeBSD 9.1_x86 verzióját újra kellett telepítenem, mert az új számítógép hardverváltozásai miatt az operációs rendszer nem volt képes elindulni.

Az új hardveres relay konfigurációkat természetesen nem tudtam nagymértékben leterhelni, de a vizsgálat szempontjából elegendő, tíz klienses terhelés okozta a legnagyobb igénybevételt. A responder számítógépen a terhelések miatt bekövetkezett többlet memória igény maximális értéke 3,83 MB volt, a processzorkihasználtság pedig nem haladta meg a 17 %-ot, ezért a mérések eredményei hitelesnek tekinthetők.

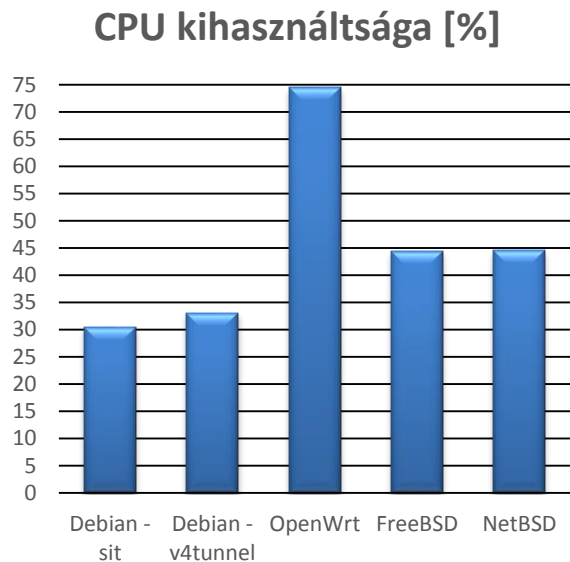
A mérési eredményeket a 7. táblázat foglalja össze, az eredmények grafikus formában a 48., 49., 50., 51. ábrán láthatók.

7. táblázat Átlagos PC - 6to4 relay mérési eredmények.

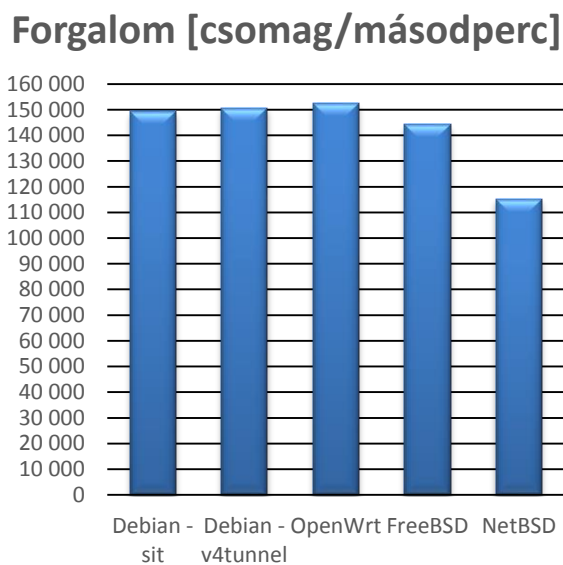
Implementáció		Debian - sit	Debian - v4tunnel	OpenWrt - sit	FreeBSD - stf	NetBSD - stf
Csomagvesztés [%]		0,119	0,072	0,065	0,087	0,082
Válaszidő [ms]	Átlag	0,442	0,472	0,460	0,519	0,799
	Szórás	0,754	1,134	1,138	0,936	0,903
	Maximum	73,700	71,000	64,800	66,500	53,200
CPU kihasználtsága [%]	Átlag	30,479	33,106	74,504	44,469	44,517
	Szórás	13,608	11,045	10,091	4,251	3,747
Memória foglalása [KB]		12,230	13,684	12,020	0,012	0,770
Forgalom [csomag/másodperc]		149269	150562	152360	144195	115162
Összesen forgalmazott csomag [$x \cdot 10^6$ db]		20,959	20,964	20,965	20,962	20,963



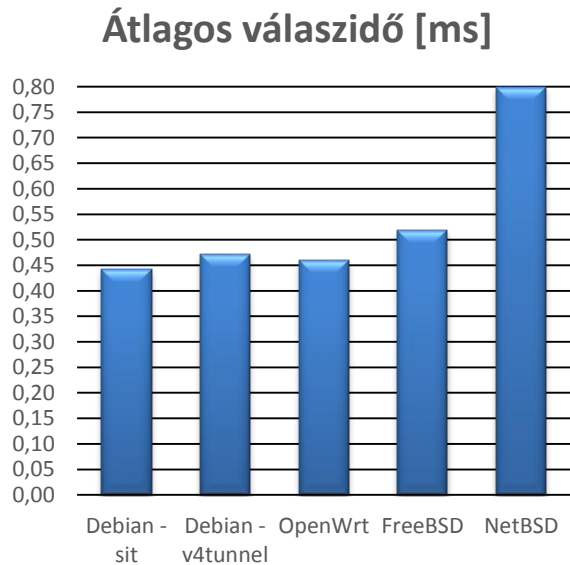
48. ábra 6to4 relay implementációk megbízhatóságának összehasonlítása.



49. ábra 6to4 relay implementációk erőforrásigényének összehasonlítása.



50. ábra 6to4 relay implementációk teljesítőképességének összehasonlítása.



51. ábra 6to4 relay implementációk sebességének összehasonlítása.

A mérések alapján kijelenthető, hogy:

- A csomagvesztés az OpenWrt-t kivéve minden rendszer esetén megnőtt. A FreeBSD teljesítményét tekintve, közel beérte a Linux operációs rendszereket. Az stf csomagvesztése átlagosnak mondható. A Debian-sit konfiguráció csomagvesztése a legnagyobb, ez a korábbi képességének közel duplája. A legkisebb csomagvesztéssel az OpenWrt tűnik a legmegbízhatóbbnak, a mérések alapján a Debian-sit konfiguráció fele ennyire megbízható. Az OpenWrt a korábbi utolsó helyéről az elsőre ugrott.
- A válaszidők átlaga mindegyik implementáció esetén közel negyedére csökkent, ami az

erősebb hardverkonfigurációnak köszönhető. A lassabb működést most is a BSD-alapú, az stf implementáció produkálta, de a leglassabb most a NetBSD-stf párosítás volt 0,799 ms-mal, ami egyébként óriási javulás az eddigi sebességéhez képest, viszont a Debian-sit még nála is kétszer gyorsabb. A leggyorsabb implementáció tehát a sit. Minimálisan, de a Debian-sit kombináció jobban teljesít az OpenWrt-hez képest. Összességében mindegyik rendszer tűrhetően teljesített.

- A processzorterhelés az OpenWrt-t kivéve, 50 % alá csökkent. Az OpenWrt esetén 74,5 %-os volt a processzor kihasználtsága, ezzel ez a legkevésbé hatékony konfiguráció. Az stf implementáció mindkét operációs rendszeren hasonlóan, közepesen teljesített, processzor-erőforrásigénye 45 % körüli. A Debian-sit kombináció végzi a 6to4 relay mechanizmus megvalósítását a legkisebb energia igényvel, amely alig több mint 30 %. A processzor-kihasználtság szórása Linux esetén nagyobb, mint 10 %, ebből kifolyólag a processzor igénybevétele kevésbé stabil, mint a BSD rendszerek esetében.
- A memóriakihasználtság a Pentium számítógépes eredményekhez hasonló arányban változik. A legkevesebb felhasznált memória a FreeBSD operációs rendszer esetén adódott, amely rendkívül alacsony: 0,012 KB volt. A rendelkezésre álló több, mint 1 GB szabad memóriához viszonyítva a legnagyobb érték, a 13,68 KB is nevetségesen kevés, ezért kijelenthető, hogy a 6to4 mechanizmus megvalósítása nagyon alacsony memóriakapacitást igényel.
- A másodpercenkénti forgalom az összes implementáció esetén több, mint duplájára nőtt. Kijelenthető, hogy a BSD implementációk most is gyengébben teljesítettek a Linux implementációkhoz képest. A forgalom sebessége a FreeBSD-n volt a legalacsonyabb: 115162 csomag másodpercenként. A legjobban az OpenWrt-sit párosítás teljesített, amely 32 %-kal több csomagot képes áteresztetni egy másodperc alatt, mint a FreeBSD. Azonban érdemes figyelembe venni, hogy ez csak 10 kliens terhelése esetén igaz, valószínű, hogy többlet terhelés esetén a többi Linux jobban teljesítene, mert az OpenWrt-nek sokkal nagyobb a processzor erőforrásigénye.

Minden jelenlegi eredményt és a tesztelt operációs rendszereket összevetve leszűrhető, hogy bár az OpenWrt-nek a legnagyobb az erőforrásigénye, az összes többi tulajdonsága alapján, a mérés során generált forgalom méretét figyelembe véve, ez a legkézenfekvőbb választás. Alkalmazhatósága nem kizárt nagy forgalmat lebonyolító, megbízható összeköttetést igénylő szolgáltatói környezetben sem.

Legjobb tudomásom szerint nem létezik 64 bites processzorra írt OpenWrt operációs rendszer, ezért teszteltem végig ugyanazokat az operációs rendszer kiadásokat, mint korábban. Azonban, ha a többi operációs rendszer 64 bites verziójával vetném össze az általam tesztelt OpenWrt-t, lehetséges, hogy alulmaradna, mivel a 64 bites architektúrákhoz tervezett utasításkészlettel rendelkező rendszerek esetleg sokkal jobban kihasználnák az AMD processzorban rejlő lehetőségeket.

5.7. Végeredmény

A kétfázisú mérésorozatomnak köszönhetően – véleményem szerint – mélyrehatóan sikerült tanulmányoznom a sit, v4tunnel, stf implementációk teljesítőképességét OpenWrt, Debian Linux, FreeBSD és NetBSD platformokon. Tesztjeim végeredménye alapján kijelentem, hogy a gif és a gre implementációk, továbbá az OpenBSD operációs rendszer nem alkalmas 6to4 relay mechanizmus megvalósítására. Általánosságban véve igaz, hogy a Linux implementációk működésüket tekintve gyorsabbak a BSD implementációknál, továbbá, hogy a 6to4 mechanizmus megvalósításához néhányszor tíz kilobájtnyi memóriakapacitás elegendő.

Abban az esetben, ha a relay router üzembe helyezéséhez gyenge hardverkonfigurációjú számítógép áll rendelkezésre, olyan környezetbe, ahol a gyors rendelkezésre állás az elsődleges szempont, a Debian Linux operációs rendszert ajánlom sit-, vagy v4tunnel konfigurációval. A két implementáció nagy hatásfok mellett egyenletes, megbízható teljesítményt nyújt nagy terhelés esetén is. Ha az üzemeltetni kívánt hálózatban a csomagok célba érése a kulcsfontosságú tényező, akkor a kisebb teljesítményű, ám annál inkább megbízható FreeBSD-stf párosítást célszerű választani, mert a relay konfiguráció a terhelés növekedése mellett folyamatosan képes tartani a csomagveszteségi rátáját, továbbá a rendszer túlterhelése esetén is minimális csomagveszteséggel dolgozik.

Amennyiben gyorsabb (minimum 2,2 GHz-en működő egymagos AMD processzorral felszerelt) számítógép áll a rendelkezésünkre, már többféle szempont szerint helyezhetünk üzembe 6to4 relay router-t. Ha az operációs rendszerünk és a rajta futtatott programok számára minimális processzorkapacitás is elegendő, a minden tekintetben kiváló tulajdonságokkal rendelkező OpenWrt-sit kombinációt ajánlom. Abban az esetben, hogyha például web, levelezés, tárhely kiszolgálást is végzünk az adott kiszolgálón és mindemellett gyors reakcióidejű 6to4 relay mechanizmust szeretnénk, a Debian Linux és a sit implementáció alkalmazása a legelőnyösebb, mert alacsony processzorkihasználtság mellett a rendszer még nagy forgalom esetén is gyors válaszidővel képes végrehajtani az útvonal-irányítást. Harmadik lehetséges opció, hogy az operációs rendszerünkön nem tervezünk tartósan nagy processzorterhelést kiváltó szolgáltatásokat futtatni és a hálózat minősége, milyensége szempontjából nincsen semmiféle extra elvárás. Ebben az esetben nyugodt szívvel ajánlom a FreeBSD-stf és a Debian-v4tunnel konfigurációkat.

6. Routerek

Konzulensem, Répás Sándor kérésére megvizsgáltam két útvonalválasztót. Kisebb szolgáltatók előszeretettel alkalmaznak olcsó, hálózati üzemeltetés céljából készített eszközöket, így az én véleményem szerint is érdemes kideríteni mire képesek a 6to4 mechanizmus megvalósítása terén ezek a "számítógépek". Az eszközök tesztelését is a 21. ábrán látható elrendezés szerint végeztem. A mérések során a responder számítógépen a terhelések miatt bekövetkezett többlet memória igény maximális értéke 3,83 MB, a maximális processzorkihasználtság 12 % volt. A számítógép teljesítőképessége ennél fogva nem befolyásolta a mérés kimenetelét.

6.1.1. Ubiquiti EdgeRouter™ Lite – sit

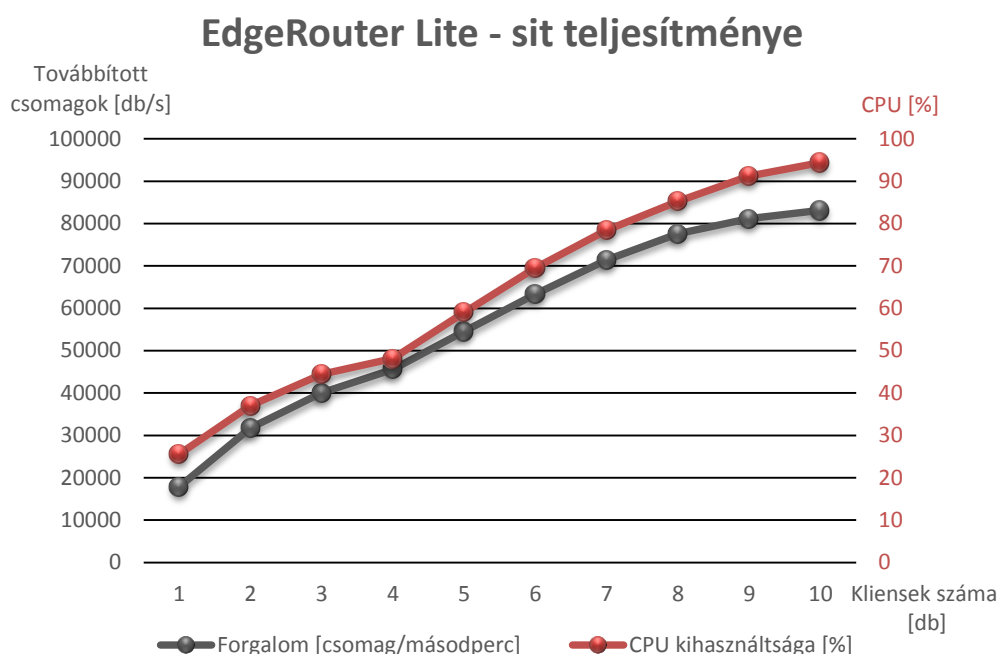
A router tulajdonságai:

- MIPS64 (500 MHz) dupla magos processzor
- 512 MB DDR2 memória
- 3 db gigabites Ethernet port
- EdgeRouter.ER-e100.v1.2.0.4574253.130626.1248 firmware verzió
- Gyártó által módosított Debian GNU/Linux operációs rendszer
- Linux ubnt 2.6.32.13-UBNT kernelverzió

A relay konfigurációja megegyezett az 5.2.1. pontban közölt Debian-sit konfigurációval. A mérési eredmények a 8. táblázatban, a router teljesítőképessége grafikusán az 52. ábrán látható.

8. táblázat Ubiquiti EdgeRouter™ Lite – sit 6to4 relay mérési eredmények.

Kliensek száma		1	2	3	4	5	6	7	8	9	10
Csomagvesztés [%]		0,003	0,004	0,007	0,011	0,012	0,016	0,020	0,025	0,029	0,036
Válaszidő [ms]	Átlag	0,304	0,428	0,624	0,837	0,914	0,970	1,027	1,109	1,235	1,373
	Szórás	0,075	0,166	0,278	0,363	0,401	0,463	0,534	0,641	0,711	0,752
	Maximum	4,620	9,370	13,600	17,500	21,700	25,800	28,900	36,800	38,700	38,500
CPU kihasználtsága [%]	Átlag	25,517	37,022	44,445	48,111	59,053	69,532	78,425	85,281	91,227	94,429
	Szórás	3,052	6,102	5,095	4,122	5,527	3,914	7,223	7,441	5,934	9,049
Memória foglalása [KB]		0,188	0,211	0,246	0,363	0,512	1,703	1,605	1,678	1,906	2,012
Forgalom [csomag/másodperc]		17857	31758	40037	45583	54551	63301	71426	77599	81135	83126
Összesen forgalmazott csomag [x*10 ⁶ db]		2,097	4,194	6,291	8,388	10,485	12,582	14,679	16,775	18,872	20,968



52. ábra Az Ubiquiti EdgeRouter™ Lite – sit 6to4 relay teljesítménye.

A mérések alapján kijelenthető, hogy:

- A csomagvesztés nagysága a terhelés függvényében növekvő. A relay tíz klienses terhelése esetén is minimális, 0,036 % a csomagvesztés.
- A válaszidők átlaga, szórása a terheléssel növekszik, az átlagos válaszidő tíz kliens esetén sem haladja meg az 1,373 milliszekundumot. A válaszidők maximuma a kilenc klienses mérési pontig növekvő, tíz kliens hatására sem vesz fel számottevően nagy értéket.
- A processzorterhelés folyamatosan növekvő. A négy klienses mérési pontban törése van a grafikonnak. Úgy tűnik négy számítógép egyidejű terhelése csupán körülbelül 4 % többlet terhelést jelent a három klienses terheléshez képest, míg a többi mérési pont esetében legalább 7 % az eltérés. (A mérés sorozatot többször megismétltem, de mindig ezt tapasztaltam.) A relay processzorkihasználtsága egyetlen kliens esetén viszonylag magas, körülbelül 25,5 %, viszont maximális terhelés mellett sem éri el a 95 %-ot. A tíz klienses terhelésnél a processzorterhelés szórása meghaladta a 9 %-ot, ezért itt némi terhelés ingadozásra lehet következtetni, a többi mérés esetén egyenletesnek mondható a relay terhelése.
- A mérések során kis ingadozással körülbelül 0,19 és 2 KB közötti értéken maradt a memóriaszükséglet, a memóriakihasználtság tíz kliens terhelése esetén volt a maximális, ám még ekkor is csekélynek tekinthető a rendelkezésre álló több mint 300 MB-hoz képest.
- Az implementáció teljesítőképessége tíz kliens terhelése esetén a legnagyobb, bár a változás az előző pontokban mért teljesítménynövekedéshez képest kisebb, véleményem szerint a router még ennél is kicsivel többre lenne képes. Sajnos nem volt elég eszközöm hozzá, hogy kiderítsem a készülék képességeinek határát.

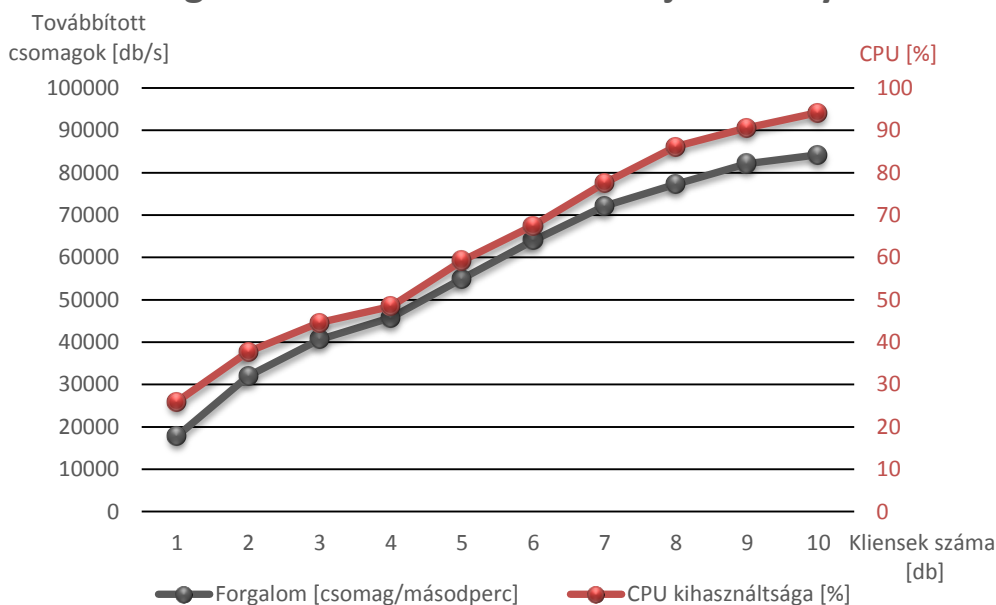
6.1.2. Ubiquiti EdgeRouter™ Lite – v4tunnel

A relay konfigurációja megegyezett az 5.2.2. pontban közölt Debian-v4tunnel konfigurációval. A mérési eredmények a 9. táblázatban, a router teljesítőképessége grafikusan az 53. ábrán látható.

9. táblázat Ubiquiti EdgeRouter™ Lite – v4tunnel 6to4 relay mérési eredmények.

Kliensek száma		1	2	3	4	5	6	7	8	9	10
Csomagvesztés [%]		0,003	0,004	0,007	0,011	0,013	0,019	0,019	0,024	0,030	0,034
Válaszidő [ms]	Átlag	0,304	0,417	0,606	0,831	0,903	0,953	1,012	1,116	1,211	1,372
	Szórás	0,077	0,160	0,271	0,357	0,402	0,466	0,545	0,642	0,698	0,774
	Maximum	4,520	9,900	13,500	17,100	21,000	26,300	31,400	35,300	39,600	40,400
CPU kihasználtsága [%]	Átlag	25,975	37,750	44,641	48,447	59,234	67,516	77,566	86,194	90,607	94,183
	Szórás	5,717	4,425	5,172	4,046	4,510	9,049	6,588	5,918	6,997	5,837
Memória foglalása [KB]		0,184	0,953	1,066	1,031	1,102	1,180	1,898	1,824	2,066	2,043
Forgalom [csomag/másodperc]		17830	31999	40640	45771	54960	64024	72137	77290	82160	84236
Összesen forgalmazott csomag [$x \cdot 10^6$ db]		2,097	4,194	6,291	8,388	10,485	12,582	14,679	16,775	18,872	20,968

EdgeRouter Lite - v4tunnel teljesítménye



53. ábra Az Ubiquiti EdgeRouter™ Lite – v4tunnel 6to4 relay teljesítménye.

A mérések alapján kijelenthető, hogy:

- A csomagvesztés nagysága a hatos és a hetes mérési pontot kivéve növekvő, értéke tíz kliens egyidejű terhelése esetén a maximális.

- A válaszidők átlaga, szórása és maximuma a terhelés függvényében növekszik, a válaszidő tíz kliens esetén a maximális, amely szinte megegyezik a sit implementáció legnagyobb válaszüzvejével, a gyakorlatban a két implementációt egyforma sebességűnek ítélem meg.
- A processzor terhelés itt is folyamatosan növekvő, de nem lineáris jellegű. A v4tunnel esetén is a négyes mérési pontban törése van a grafikonnak. Ebben a pontban mindössze 0,3 % körüli az eltérés a sit processzor terheléséhez képest. A relay processzor kihasználtsága egyetlen kliens esetén hasonlóan magas. Maximális igénybevétele mellett 94,183 % a processzor terhelése. A v4tunnel implementáció a sittel közel egyenlő nagyságú processzor-kapacitást igényel. A hat klienses terhelésnél a processzor terhelés szórása meghaladta a 9 %-ot, ezért itt némi terhelés ingadozásra lehet következtetni, a többi mérés esetén egyenletesnek mondható a relay terhelése.
- A memóriaszükséglet növekvő jellegű, kis mértékben ingadozik. A memóriakihasználtság tíz kliens terhelése esetén volt a legnagyobb: 2,066 KB. A rendelkezésre álló memóriaméretéhez viszonyítva azonban ez elhanyagolható mennyiség.
- A forgalom változása egybefüggő a terheltség változásával. Az eltérő processzor terheléseknek köszönhetően az 1-es, 4-es, 5-ös, 6-os, 7-es mérési pontban a két grafikon jobban eltávolodik egymástól, mint a sit implementáció esetén. Az implementáció teljesítőképessége jelen esetben is a tízes mérési pontban a maximális, ekkor a legnagyobb a processzor kihasználtsága. A teljesítőképesség a sitnél említettek szerint, talán még növelhető lenne. A legnagyobb csomagsebesség értéke 84236 másodpercenként, ez 1,3 %-kal több, mint a sit maximális forgalma.

6.1.3. RouterStation Pro – sit

A router tulajdonságai:

- Atheros AR7161 (rev 2) rendszer architektúra
- MIPS 24Kc V7.4 processzor
- 128 MB rendszer memória
- 4 db Gigabit Ethernet port
- openwrt-ar71xx-generic-ubnt-rspro-squashfs-factory firmware verzió
- OpenWrt Attitude Adjustment 12.09, r36088 kiadású operációs rendszer
- Linux 3.3.8 kernelverzió

A relay konfigurációja az 54., 55., 56. ábrán látható. A routeren a következő csomagokat telepítettem: kmod-ipv6, 6to4, kmod-sit, procps. A sikeres 6to4 relay mechanizmus működéséhez az 5.2.3. alfejezetben említett feltételek most is érvényesek, mivel a két operációs rendszer verziószáma

megegyezik, gyakorlatilag ugyanarról a rendszerről van szó, viszont routerek esetén kicsit eltérő a konfigurációs állományok tartalma, például a beállítani kívánt értékeket aposztrófok közé kell tenni.

```
config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
    option ipaddr '127.0.0.1'
    option netmask '255.0.0.0'

#publikus IPv4:
config interface 'lan'
    option ifname 'eth1'
    option proto 'static'
    option 'ipaddr' '193.225.151.78'
    option 'netmask' '255.255.255.240'

#globális IPv6:
config interface 'wan'
    option ifname 'eth0'
    option proto 'static'
    option 'ip6addr' 'babe:b00b::1/64'

#6to4:
config 'interface' 'sit0'
    option 'proto' '6to4'
    option 'ipaddr' '193.225.151.78'
    option 'netmask' '255.255.255.240'

config switch
    option name 'switch0'
    option reset '1'
    option enable_vlan '1'

config switch_vlan
    option device 'switch0'
    option vlan '1'
    option ports '0 1 2 3 4'
```

54. ábra A /etc/config/network fájl tartalma RouterStation Pro 6to4 relay router esetén.

```
#Responder-hez vezető útvonal statikus bejegyzése:
ip -f inet6 route add 2001:738:2c01:8000::/64 via babe:b00b::2

#6to4 IPv6 cím hozzárendelése a pszeudo-interfészhez:
ifconfig sit0 2002:c1e1:974e::1/16 up
```

55. ábra A /etc/rc.local fájl tartalma RouterStation Pro 6to4 relay router esetén.

```
#IPv4-es csomagok továbbításának engedélyezése:
net.ipv4.ip_forward=1

#IPv6-os csomagok továbbításának engedélyezése:
net.ipv6.conf.all.forwarding=1
```

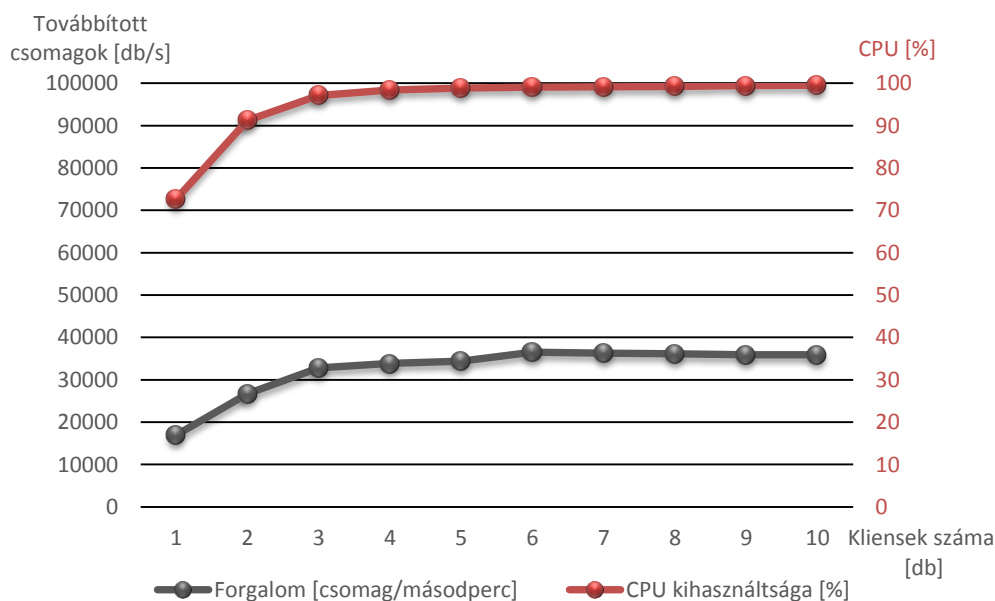
56. ábra A /etc/sysctl.conf szükséges fájl tartalma RouterStation Pro 6to4 relay router esetén.

A mérések eredményét a 10. táblázat tartalmazza, a router teljesítőképességét grafikusán az 57. ábrán tüntettem fel.

10. táblázat RouterStation Pro – sit 6to4 relay mérési eredmények.

Kliensek száma		1	2	3	4	5	6	7	8	9	10
Csomagvesztés [%]		0,003	0,006	0,010	0,026	0,075	0,134	0,165	0,158	0,169	0,164
Válaszidő [ms]	Átlag	0,385	0,639	0,901	1,326	1,770	2,076	2,537	2,992	3,472	3,926
	Szórás	0,181	0,284	0,386	0,514	0,675	0,829	0,920	0,962	0,924	1,005
	Maximum	28,300	28,900	29,700	30,300	30,800	31,000	31,600	32,100	32,700	20,500
CPU kihasználtsága [%]	Átlag	72,626	91,256	97,127	98,432	98,897	99,115	99,159	99,254	99,441	99,554
	Szórás	14,607	5,729	7,820	9,240	4,211	6,044	2,444	4,385	3,323	4,251
Memória foglalása [KB]		10,645	10,586	10,531	10,688	10,246	10,309	10,688	10,135	10,559	10,570
Forgalom [csomag/másodperc]		16927	26697	32800	33852	34428	36508	36312	36178	35907	35932
Összesen forgalmazott csomag [$\times 10^6$ db]		2,097	4,194	6,291	8,388	10,482	12,574	14,668	16,764	18,858	20,954

RouterStation Pro - sit teljesítménye



57. ábra A RouterStation Pro – sit 6to4 relay teljesítménye.

A mérések alapján kijelenthető, hogy:

- A csomagvesztés nagysága a hét klienses terhelési pontig növekvő, maximális értéke kilenc kliens egyidejű terhelése esetén 0,169 %, amely a korábbi Linux implementáció tesztekhez képest jóval magasabb.
- A válaszidők átlaga, szórása a terheléssel együtt növekszik, az átlagos válaszidők nagyobb mértékben változnak a korábbiakban tapasztaltakhoz képest. A maximális válaszidő a 9 klienses mérési pontig egyre növekvő, az utolsó pontban, valószínűleg a rendszer túlterhelése

miatt visszaesik, mert a valóságban legnagyobb válaszidővel visszaérkező csomag minden bizonnyal elveszett és nem került rögzítésre. Tíz kliens esetén a legnagyobb a válaszidő: 3,926 milliszekundum, ez minden eddigi eredménynél magasabb érték. A router működési sebessége az összes rendszer közül a leglassabb.

- A relayen már egy kliens is több, mint 14 %-os szórás mellett körülbelül 72 % processzor-terhelést okoz. A relay processzorkihasználtsága ekkor eléggé instabil. A processzor-terhelés nagysága a három kliens terhelési pontjáig ugrásszerűen nő, a későbbi szakaszon kisebb mértékben emelkedik tovább. A szórásértékek az egy klienses terhelést kivéve 10 % alatt maradnak, a processzor igénybevétele ekkor egyenletesebb. A processzorkihasználtság már 6 kliens terhelése esetén meghaladja a 99 %-ot. A RouterStation Pro minden eddiginél nagyobb terhelést szenvedett el, ez minden bizonnyal a routerben található integrált processzor gyenge teljesítményének köszönhető.
- A memóriakihasználtság durván 10,5 KB körüli. Ingadozás figyelhető meg az értékek között. Érdekes módon a memóriaszükséglet nem tíz kliens terhelése esetén volt a maximális, ez valószínűleg a magasabb processzor-terhelésnek is köszönhető, ami miatt a vmstat program az aktuális legkisebb vagy legnagyobb szabad memóriaértéket nem volt képes fájlba menteni és ebből adódóan pontatlan érték került kiszámításra.
- A hat klienses mérési pontig a forgalom nagysága szabálytalanul nő. Az implementáció teljesítőképessége hat kliens esetén a legnagyobb, a processzor-terhelés ekkor 99,115 %. A teljesítőképességet jól szemlélteti, hogy az egy másodperc alatt maximálisan továbbított csomagok száma körülbelül 56 %-kal kevesebb a korábbi routerénél. További kliens bevonásával már nem növekszik a teljesítőképesség az 5.4.1-ben tisztázott okok miatt, a többlet terhelés a teljesítőképesség rovására ment. A tíz klienses mérési pontban a forgalom sebessége körülbelül 1,6 %-ot csökken a maximálishoz képest.

6.1.4. Konklúzió

Az EdgeRouter Lite kiváló beruházásnak ígérkezik kis forgalmú szolgáltatók számára. Gyakorlatilag jobban teljesített az összes Pentiumon tesztelt implementáció-operációs rendszer kombinációnál. Amennyiben nincs szükségünk hatalmas forgalom kiszolgálására, az EdgeRouter Lite bizonyul a legjobb választásnak. Jó ár/érték aránya mellett energiafogyasztása, helyigénye, működési zaja, hűtési igénye eltörpül egy öreg asztali számítógépéhez képest. Elméletileg bármilyen operációs rendszert fel lehetne rá telepíteni, így specifikus hatékonysága szoftveresen tovább javítható. A router használatát csak ajánlani tudom.

A RouterStation Pro processzorának teljesítménye elvileg szoftveresen javítható, de a router ebben az esetben extra hűtést igényel. Az eszközt már nem gyártják, ha valaki mégis olcsón hozzájut

interneten keresztül, gyenge teljesítménye ellenére otthoni használatra még kiváló lehet. Különböző verziójú operációs rendszerek és ezeknek függvényében különböző csomagok telepíthetők rá. Cross-compile módszerrel lehet készíteni saját telepítő csomagokat és programokat is, melyeket más Unix rendszerhez írtak, de mi szeretnénk routerünkön használni. Elvileg készíthetünk egyedi operációs rendszert is. Nem csak a Linuxoknál ismert kernel fordítással, hanem egyéb fejlesztés alatt álló modulok saját célra átírásával és betöltésével igényeinknek megfelelően tovább formálható a rendszer. Ezen tulajdonságai miatt a router jó választás lehet vállalkozó kedvű szakemberek számára, akik nem feltétlenül 6to4 relay routerként szeretnék üzembe helyezni.

7. Összefoglalás

Munkám során felkutattam a szabad, széles körben alkalmazott 6to4 relay implementációkat Linux, FreeBSD, NetBSD és OpenWRT platformokra. Kutatásaim szerint az említett kategóriában három különböző 6to4 relay implementáció biztosan létezik: sit, v4tunnel, stf. Dolgozatomban bemutattam mire képesek az egyes implementációk és az operációs rendszerek. A mérések kivitelezése során bő ismeretre tettem szert a különböző operációs rendszerek és a 6to4-hez kapcsolódó szakirodalomnak köszönhetően. Megismertem a 6to4 mechanizmus sokoldalú felhasználhatóságát a különböző protokoll verziójú hálózatokban. A két terheléssorozatnak köszönhetően kiderítettem az egyes konfigurációkban rejlő képességeket, nagyon gyenge-, és erősebb hardver környezetben is. Az implementációk megbízhatóságára túlterhelés esetén is fényt derítettem.

A méréseim alapján a legjobban a Linux operációs rendszer teljesít. Igények és lehetőségek függvényében mind a sit, mind a v4tunnel implementáció kiváló választás. Kisméretű hálózati forgalom (körülbelül 85.000 64 bájt méretű csomag/másodperc) esetén az EdgeRouter Lite a legjobb választás, míg robosztusabb kiszolgálást az OpenWrt vagy a Debian Linux-sit és egy nagyobb teljesítményű (2,2 GHz-en működő egymagos AMD processzornál erősebb processzorral felszerelt) asztali számítógép segítségével nyújthatunk.

A folyamatosan növekvő adatátviteli sebességek miatt érdemes nagyobb hangsúlyt fektetni a 6to4 relayt megvalósító rendszerek fejlesztésére is. A jövőben ezért szükség lehet 64 bites rendszerek teljesítőképességének a tanulmányozására is, hiszen a legtöbb operációs rendszer, amely akár képes lehet 6to4 relayként is üzemelni, rendelkezik a mai modern, AMD64 architektúrájú processzorokhoz készült utasításkészlettel. Terveim közé tartozik a 6to4 mechanizmus biztonsággal kapcsolatos problémakörének felkutatása, tanulmányozása, ugyanis a sebesség mellett a hálózatok biztonságával szemben is nagyok az elvárások.

Úgy gondolom dolgozatom témája aktuális és jövőbeni problémák egy lehetséges megoldását szemlélteti, így a megírásakor elsajátított tapasztalatokat az elkövetkező szakmai életben sikerrel fogom kamatoztatni. Egyes vélemények szerint a 6to4 technika már elavultnak számít [19]. Véleményem szerint az áttérési technikák alkalmazására az elkövetkező pár évben lesz a legnagyobb szükség. Amíg nem tud minden készülék és szolgáltató natív IPv6 címmel kommunikálni, addig a 6to4 mechanizmus használatára is lesz igény.

8. Irodalomjegyzék

- [1] S. Deering, Cisco, R. Hinden, Nokia - Internet Protocol, Version 6 (IPv6) Specification, Request for Comments: 2460, 1998 december
- [2] The Number Resource Organization, „Free pool of IPv4 address space depleted”
Online. Elérhető: <http://www.nro.net/news/ipv4-free-pool-depleted>
[Utolsó hozzáférés: 2013.11.04.]
- [3] RIPE NCC, „RIPE NCC Begins to Allocate IPv4 Address Space From the Last /8”
Online. Elérhető: <http://www.ripe.net/internet-coordination/news/announcements/ripe-ncc-begins-to-allocate-ipv4-address-space-from-the-last-8>
[Utolsó hozzáférés: 2013.11.04.]
- [4] Dr. Lencse Gábor - Számítógép-hálózatok, 2. kiadás, UNIVERSITAS-GYŐR Nonprofit Kft., 2008, Győr
- [5] B. Carpenter, K. Moore - Connection of IPv6 Domains via IPv4 Clouds, Request for Comments: 3056, 2001 február
- [6] M. Wasserman, Painless Security, F. Baker, Cisco Systems - IPv6-to-IPv6 Network Prefix Translation, Request for Comments: 6296, 2011 június
- [7] Répás Sándor - "TMDK Dolgozat - A DNS64 és NAT64 IPv6 áttérési technikák egyes implementációinak teljesítőképesség- és stabilitás-vizsgálata", 2012, Győr
- [8] P. Savola, C. Patel - Security Considerations for 6to4, Request for Comments: 3964, 2004 december
- [9] ARIN IPv6 Wiki team - "Linux or BSD 6to4 Relays", utoljára módosítva: 2012. július 30., 17:04-kor
Online. Elérhető: http://www.getipv6.info/index.php/Linux_or_BSD_6to4_Relays
[Utolsó hozzáférés: 2013.03.29.]
- [10] Debian Wiki team - "Debian IPv6 Project", Utoljára Natanael Arndt módosította 2012.10.22., 09:28:33-kor.
Online. Elérhető:
http://wiki.debian.org/DebianIPv6#Setting_up_a_6to4_relay_router_using_Debian
[Utolsó hozzáférés: 2013.03.29.]
- [11] Debian Support - SixXS Automatic IPv6 Connectivity Client Utility, 2007.01.15.
Online. Elérhető: <http://packages.debian.org/unstable/net/aiccu>
[Utolsó hozzáférés: 2013.04.28.]
- [12] SixXS - AICCU „Automatic IPv6 Connectivity Client Utility”, 2007.01.15.
Online. Elérhető: <http://www.sixxs.net/tools/aiccu/>
[Utolsó hozzáférés: 2013.04.28.]
- [13] Stephane Bortzmeyer - "Setting up an IPv6 tunnel with Hexago/Freenet", 2008 február
Online. Elérhető: <http://forum.slicehost.com/index.php?p=/discussion/1421/setting-up-an-ipv6-tunnel-with-hexagofreenet/pl>
[Utolsó hozzáférés: 2013.03.29.]

[14] openbsd dev - ipv6 (programfejlesztők fóruma) - Discussion of IPv6 and IPsec in OpenBSD > 6to4 relay implementation question, Utoljára módosította: Stuart Henderson, 2013.március 27.; 1:11pm

Online. Elérhető: <http://openbsd.7691.n7.nabble.com/6to4-relay-implementation-question-td225788.html>

[Utolsó hozzáférés: 2013.10.07.]

[15] Bert Hubert - Creating an IP Tunnel using GRE on Linux/Linux Advanced Routing & Traffic Control HOWTO, 2002

Online. Elérhető: <http://tier.cs.berkeley.edu/drupal/howto/ip-tunnel-using-gre-on-linux>

[Utolsó hozzáférés: 2013.03.30.]

[16] Federico Biancuzzi - The Man in the Machine, 2007.12.04., (SecurityFocus online cikk.)

Online. Elérhető: <http://www.securityfocus.com/columnists/459/1>

[Utolsó hozzáférés: 2013.10.07.]

[17] Hurricane Electric's IPv6 Tunnel Broker Forums > Tunnelbroker.net Specific Topics > Questions & Answers > How to create a local tunel from one to other mashine, Utolsó hozzászólás „arkadietz” által 2010.november.2-án, délután 4:10:41-kor.

Online. Elérhető:

<http://www.tunnelbroker.net/forums/index.php?PHPSESSID=a09e0be5ca14a0e78c827578803c7cf7&topic=1254.msg6949#msg6949>

[Utolsó hozzáférés: 2013.10.29.]

[18] Kallai Péter, Kovács Ákos, Lencse Gábor, Molnár Zoltán Vilmos, Sinkó Gergely - Hálózati operációs rendszerek (Linux), Széchenyi István Egyetem, Távközlési Tanszék, 2011. 11. 20., Győr

[19] Nick Sayer - My 6to4 page > „6to4 is obsolete” bekezdés.

Online. Elérhető: <http://www.kfu.com/~nsayer/6to4/>

[Utolsó hozzáférés: 2013.10.29.]

[20] Drago Žagar, Goran Martinović és Snježana Rimac-Drlje - Testing the Tools for IPv6 Traffic Tunneling, University of Osijek Faculty of Electrical Engineering Kneza Trpimira 2b , HR-31000 Osijek, CROATIA December 16-18, 2005.

9. Függelék

9.1. Mérés

A scriptek működésének magyarázatai minden esetben soronként, az adott művelet fölött a # jel után olvashatók.

```
#!/bin/bash

#Elérhető-e a responder szerver?:
true=$(ping6 -c1 -q 2001:738:2c01:8000::1 | grep loss | awk '{print $4}')
if [ "$true" == "0" ]
then
#Ha nem, a program írja ki pirossal a képernyőre, hogy „HIBA!” és állítsa le
#a scriptet:
echo -e "\n\033[4;31mHIBA!\033[0m"
kill $(ps aux | grep automeres.sh | awk '{print $2}')
else
#Ha elérhető, a script törölje az előző mérés mérési idejét tartalmazó fájlt,
#ha volt ilyen, majd hozza létre a mostani méréshez szükséges txt állományt:
rm -rf /root/meres/meresideje.txt
touch /root/meres/meresideje.txt
#Futtassa a 6to4ping.sh scriptemet és közben mérje a time paranccsal a script
#futása alatt eltelt időt, majd a végeredményt mentse a
#/root/meres/meresideje.txt fájlba:
/usr/bin/time -f "%E" -o /root/meres/meresideje.txt -a ./6to4ping.sh
#A script írja ki zölddel, aláhúzva a „minta:” karaktereket, majd a mérés
#utolsó eredményfájljából írjon ki a képernyőre egy sor eredményt mintaként,
#hogy meggyőződhessek róla, a mérés sikeres volt:
echo -e "\n\033[4;36mminta:\033[0m" $(cat 255/6to4-*63 | tail -2 | grep loss)
#A program mentse az eredményeket:
cd mentoscriptek
./mentes.sh
cd ..
fi
```

58. ábra automeres.sh

9.2. Eredmények kiértékelése

A mellékletek között található kiertekeles.sh program a felki.sh scriptem segítségével végezte az adatok kiértékelését. A kiertekeles.sh felépítését tekintve elég primitív, sokkal "szebben", kevesebb sorból is meg tudtam volna írni, de a script felépítése szándékosan lett ilyen. Abban az esetben, hogyha tesztek végzem csupán, vagy ha bármelyik mérést meg kellett ismételnem, az adott mérés kiértékelésében nagy segítségemre volt, hogy csak a script megfelelő sorait kimásoltam akár konzol képernyőre, akár egy új fájlba és a programrészlet segítségével csak a szükséges lépések szerint értékeltem ki a részeredményeket. További előny volt, hogy a program egyszerűsége az esetleges hibák kiszűrésében és nyomon követésében is segített, mert nem írta tele a konzol képernyőt a hiányzó állományok miatt generált felesleges hibaüzenetekkel.

9.3. Terhelési adatok kiértékelése

```
#!/bin/bash
h=/home/v
#A relay terhelése egy teljes terhelés sorozat esetén:
for i in {10..1}
do
#kigyűjtöm a processzor kihasználtságát:
cat $i.vmstat.txt | awk '{print $18}' | sed '/id/d' | sed '/^$/d' | awk '{print 100-$1}' >> cpu.txt
#átlagos processzorkihasználtság:
awk '{osszeg+=$1 ; osszes++$1} END {print osszeg/osszes}' cpu.txt > avgcpu.txt
#processzorkihasználtság szórása:
awk '{osszeg+=$1; negyzetosszeg+=$1*$1} END {print sqrt(negyzetosszeg/NR - (osszeg/NR)*(osszeg/NR))}' cpu.txt > szorascpu.txt
#memória foglalkozás Kbyte-ban:
echo $((($cat $i.vmstat.txt|awk '{print $4}'|sed '/fre/d'|sed '/disks/d'|sort -rg|head -1)-$(cat $i.vmstat.txt|awk '{print $4}'|sed '/fre/d'|sed '/disks/d'|sort -g|head -1)))/1024|bc -l > mem.txt
#kigyűjtöm a terhelés eredményeit:
cat avgcpu.txt szorascpu.txt mem.txt >> $h/EREDMENYEK/$i.gep_eredmeny.txt
#törölöm a számítások során létrejött szükségtelen fájlokat:
rm -rf cpu.txt avgcpu.txt szorascpu.txt mem.txt
done
```

59. ábra terheles.netbsd.vmstat.sh

```
#!/bin/bash
h=/home/v
#A relay terhelése egy teljes terhelés sorozat esetén:
for i in {10..1}
do
#kigyűjtöm a processzor kihasználtságát:
cat $i.vmstat.txt | awk '{print $15}' | sed '/id/d' | sed '/^$/d' | awk '{print 100-$1}' >> cpu.txt
#átlagos processzorkihasználtság:
awk '{osszeg+=$1 ; osszes++$1} END {print osszeg/osszes}' cpu.txt > avgcpu.txt
#processzorkihasználtság szórása:
awk '{osszeg+=$1; negyzetosszeg+=$1*$1} END {print sqrt(negyzetosszeg/NR - (osszeg/NR)*(osszeg/NR))}' cpu.txt > szorascpu.txt
#memória foglalkozás Kbyte-ban:
echo $((($cat $i.vmstat.txt|awk '{print $4}'|sed '/fre*/d'|sed '/-*io-*/d'|sort -rg|head -1)-$(cat $i.vmstat.txt|awk '{print $4}'|sed '/fre*/d'|sed '/-*io-*/d'|sort -g|head -1)))/1024|bc -l > mem.txt
#kigyűjtöm a terhelés eredményeit:
cat avgcpu.txt szorascpu.txt mem.txt >> $h/EREDMENYEK/$i.gep_eredmeny.txt
#törölöm a számítások során létrejött szükségtelen fájlokat:
rm -rf cpu.txt avgcpu.txt szorascpu.txt mem.txt
done
```

60. ábra terheles.linux.vmstat.sh

```
#!/bin/bash
h=/home/v
#A relay terhelése egy teljes terhelés sorozat esetén:
for i in {10..1}
do
#kigyűjtöm a processzorkihasználtságot:
cat $i.vmstat.txt | tr "," "\t" | awk '{print $19}' | sed '/id/d' | sed '/^$/d'
| awk '{print 100-$1}' >> cpu.txt
#átlagos processzorkihasználtság:
awk '{osszeg+=$1 ; osszes++$1} END {print osszeg/osszes}' cpu.txt > avgcpu.txt
#processzorkihasználtság szórása:
awk '{osszeg+=$1; negyzetosszeg+=$1*$1} END {print sqrt(negyzetosszeg/NR -
(osszeg/NR)*(osszeg/NR))}' cpu.txt > szorascpu.txt
#memória foglalkozás Kbyte-ban:
echo $((($ (cat $i.vmstat.txt|tr "," "\t"|awk '{print $5}'|sed '/fre/d'|sed
'/faults/d'|sort -rg|head -1)-$(cat $i.vmstat.txt|tr "," "\t"|awk '{print
$5}'|sed '/fre/d'|sed '/faults/d'|sort -g|head -1)))/1024|bc -l > mem.txt
#kigyűjtöm a terhelés eredményeit:
cat avgcpu.txt szorascpu.txt mem.txt >> $h/EREDMENYEK/$i.gep_eredmeny.txt
#törölöm a számítások során létrejött szükségtelen fájlokat:
rm -rf cpu.txt avgcpu.txt szorascpu.txt mem.txt
done
```

61. ábra terheles_bsd.vmstat.sh

9.4. Mellékletek

Végezetül szeretném megemlíteni a dolgozatomhoz csatolt tömörített állományt, mely a fájl mellékleteket tartalmazza. Az általam szükségesnek vélt helyeken a gyökér könyvtárakon belül található .txt kiterjesztésű fájlok gondoskodnak a mappákban lévő fájlok-, és azok tartalmának a magyarázatáról.

A zip fájl tartalma:

- beállítások:
 - mérésekhez szükséges beállítások/állományok
 - operációs rendszer specifikus opciók
 - tesztelt 6to4 host konfigurációk
- alkalmazott PC-k hardverleírása
- alkalmazott routerek hardverleírása, egyéb információk az eszközökről
- scriptek
 - mérésekhez szükséges scriptek
 - nyers eredmények mentéséhez szükséges scriptek
 - 6to4 hostok automatikus konfigurálásához szükséges scriptek
 - ssh scriptek
 - kiértékeléshez szükséges scriptek
- nyers eredményfájlok