Lencse Gábor, Répás Sándor, Arató András

IPv6
ÉS BEVEZETÉSÉT TÁMOGATÓ TECHNOLÓGIÁK

1. kiadás

HunNet-Média Kft.

Budapest, 2015

HunNetMédia

IPv6 és bevezetését támogató technológiák

Szerzők: **Dr. Lencse Gábor**
egyetemi docens, PhD

Répás Sándor
okleveles villamosmérnök,
Certified Cisco Systems Instructor (CCSI#34630),
Microsoft Certified Trainer (MCT),
MikroTik Certified Inter-networking Engineer (MTCINE)

Arató András
okleveles mérnök-informatikus,
Cisco Certified Internetwork Expert (CCIE#3297)

Lektorálta: **Dr. Almási Béla**
egyetemi docens, PhD

A könyv újabb verziójának kiadásáról tájékoztatást adunk, valamint a könyvvel kapcsolatos észrevételeket, hibabejelentéseket köszönettel fogadunk: http://ipv6ready.hu/konyv/.

ISBN: 978-963-12-3272-1

DOI: 10.18660/ipv6-b1.2015.9.1

Kiadja a HunNet-Média Kft. 1132 Budapest, Victor Hugó utca 18-22. Felelős kiadó a Kft. mindenkori ügyvezetője.

Tartalomjegyzék

1. Bevezetés

Bár az internet protokoll 6-os verzióját (IPv6) már 1998-ban definiálták (RFC 2460), általános elterjedése az egész világon, így hazánkban is sokáig késett, és csak a publikus IPv4 címtartomány kimerülésével vált halaszthatatlanná. Az IP új verziójára való zökkenőmentes és mielőbbi áttérés megköveteli, hogy a hálózatokkal foglalkozó szakemberek birtokában legyenek a szükséges ismereteknek. Ez nem pusztán az IPv6 protokoll ismeretét jelenti, hanem a két protokoll együttműködéséhez szükséges különféle ún. IPv6 áttérési technológiák (IPv6 transition technologies) és azok különféle implementációinak alapos ismeretét is. Nem tudunk olyan magyar nyelvű szakkönyvről, ami ezt a témát kellő mélységben tárgyalná; célunk ennek a hiánynak a betöltése. Munkánkkal gyakorlati szakemberek, egyetemi hallgatók, oktatók és kutatók számára egyaránt segítséget szeretnénk nyújtani. Az olvasóról feltételezzük, hogy birtokában van az alapvető számítógép-hálózati ismereteknek, beleértve mind a TCP/IP protokollcsalád (TCP/IP protocol stack), mind a hordozóhálózatként használt különféle fizikai/adatkapcsolati szintű hálózati megvalósítások ismeretét[1].

Az IPv6 témakörben az alapoktól indulunk, bemutatjuk az IPv6 címzését, az IPv6 datagram szerkezetét és az IPv6 protokoll működését, fokozott figyelmet szentelve az IPv4-től eltérő megoldásoknak (pl. ARP helyett NDP). Az elmélet mellett gyakorlati példákkal illusztráljuk az egyes megoldások beállítást, használatát, tesztelését. Külön fejezetben foglalkozunk az útválasztás kérdéseivel. Az áttérési technológiák közül azokat tárgyaljuk részletesen, amelyeket jelenleg fontosnak, előremutatónak tartunk. Néhány áttérési technológia általunk kiválasztott implementációival kapcsolatban bemutatjuk saját kutatási eredményeinket is. Reméljük, hogy ezek hasznosak lesznek mind a gyakorlati szakemberek számára (segítenek a megfelelő technológia és implementáció kiválasztásában), mind az akadémiai szféra résztvevőinek (további kutatásokra motiválnak). Bár a könyv terjedelmének jelentős részét a kutatási eredményeink bemutatása teszi ki, hangsúlyozzuk, hogy ez messze nem teljes; számos probléma vizsgálatára van még szükség, melyre ezúton is keresünk együttműködő partnereket.

Köszönetünket fejezzük ki a könyv hivatalos lektorának, Almási Bélának a kézirat átolvasásában, javításában végzett alapos munkájáért, valamint mindenkinek, aki a bírálati példányt átolvasta, és hibákat jelzett, különösen Varga Györgynek, akitől számos hibajavítást, javaslatot kaptunk.

Munkánknak ez az első kiadása, amit szeretnénk a továbbiakban folytatni, bővíteni. A könyv újabb kiadásáról tájékoztatást adunk, valamint a könyvvel kapcsolatos észrevételeket, hibabejelentéseket köszönettel fogadunk: http://ipv6ready.hu/konyv/.

Reméljük, hogy munkánk hozzájárul az IPv6 mielőbbi magyarországi elterjedéséhez.

Lencse Gábor, Répás Sándor, Arató András

[1] Ha valaki ezen a téren szeretné a tudását kiegészíteni, akkor az idő- és költséghatékony egyetemi jegyezettől [1] a vaskos angol nyelvű könyvek magyar fordításáig [2] és [3] széles palettáról válogathat.

2. IPv6 alapok

2.1. IPv6 címek

Az IPv6 protokoll egyik legszembetűnőbb eltérése az IPv4-hez képest a címek mérete. Az IPv6 128 bites címeket használ, így még a címtartomány nem hatékony felhasználása esetén is hosszú időre elegendő címet biztosít a várható és az előre nem látható feladatokra.

2.1.1. IPv6 címek írása

Rugalmas formátum

Az IPv6 címek szöveges leírásában az RFC 4291 meglehetősen rugalmas, a felhasználónak számos lehetőséget nyújt az egyszerűsítésre, de ezeket nem teszi kötelezővé. Az IPv6 cím 128 bitjét 16 bites csoportonként négy hexadecimális jeggyel írjuk, a csoportokat kettősponttal választjuk el egymástól. A csoportok elején a nullák mindig elhagyhatók. Így amennyiben egy 16 bites csoportban négy nulla áll egymás után, azt mindig helyettesíthetjük egy nullával. Ha egymás utáni 16 bites csoportok csak nullákat tartalmaznak, az összes ilyen csoportot helyettesíthetjük dupla kettősponttal („::”), de ez csak egyszer alkalmazható, hogy a dekódolás egyértelmű legyen. Lássunk egy példát:

2001:0DB8:0000:00AE:0000:0000:0000:ABCD
↓
2001:DB8:0:AE:0:0:0:ABCD
↓
2001:DB8:0:AE::ABCD

Amennyiben egy IPv6 címet IPv4 címből képeztünk, és az IPv6 cím az IPv4 címet az utolsó 32 bitjén tartalmazza, akkor az IPv4 címet tartalmazó részt írhatjuk az IPv4 kanonikus formátumában is, például a 64:FF9B::C000:20A cím írható úgy is, hogy: 64:FF9B::192.0.2.10.

A hexadecimális számjegyekben szereplő betűkkel kapcsolatban sincs megkötés, hogy kis- vagy nagybetűk legyenek, a példákban RFC 4291 nagybetűket használ.

Gyakran van szükség egy IPv6 cím első *n* bitjének a megadására, amit *n* méretű *előtag*nak (prefix) nevezünk és az előtag után „/*n*” megadásával jelölünk. Fontos, hogy prefix írásakor az utolsó olyan 16 bites csoportot, ami nem csupa 0-t tartalmaz, mindig végig ki kell írni.

Példa: 2001:DB8:AB00::/40 a helyes írásmód, mert ha 2001:DB8:AB::/40-et írnánk, az azt jelentené, hogy: 2001:0DB8:00AB::/40, ami pedig valójában: 2001:0DB8::/40.

A dupla kettőspont használatával prefixeknél különösen körültekintően kell bánni, például a 2001:DB8:0:A::/64 nem rövidíthető tovább úgy, hogy 2001:DB8::A/64, mert az teljesen mást jelent, konkrétan azt, hogy: 2001:DB8:0:0:0:0:0:A/64.

Az RFC 3986 definiálja az IPv6 címek használatát URL-ekben. Példák:

- http://[2001:DB8:FFFF:FFFF::192.224.128.1]:8080/index.html
- ftp://[2001:DB8:2C01:8000::15]

Kanonikus formátum

Az RFC 4291 rugalmassága nagyon kényelmes, így egy IPv6 címnek sokféle írásmódja lehetséges attól függően, hogy a felhasználó mely lehetőségekkel kíván élni, és melyekkel nem. Azonban az RFC 5952 leírja, hogy ez a szabadság számos esetben problémákat okozhat a gyakorlatban. Például gondoljuk arra, hogy egy naplófájlban a 2001:db8::a:0:0:1 címet szeretnénk megkeresni, miközben az olyan formában szerepel benne, hogy: 2001:0DB8:0:0:A::1. A probléma megoldására bevezetik az IPv6 címek szöveges írásának *kanonikus* formáját azzal, hogy egy program bemenetként minden

olyan címet *köteles* (MUST) elfogadni, amit RFC 4291 megenged, de kimenetén *erősen ajánlott* (SHOULD) a kanonikus forma használata. Továbbá azt tanácsolják, hogy az IPv6 címek leírásakor mi emberek is használjuk a kanonikus formát. Ennek szabályai a következők:

- Az egyes 16 bites csoportokban a vezető nullákat el *kell* (MUST) hagyni.
- A dupla kettőspontot a maximális kapacitásáig ki *kell* (MUST) használni.
- A duplakettőspontot *tilos* (MOST NOT) egyetlen 16 bites csoport rövidítésére használni.
- Amennyiben több csupa nullás csoport van, akkor a leghosszabbat, ha pedig a csoportok hossza egyenlő, akkor balról jobbra haladva az elsőt *kell* (MUST) dupla kettősponttal helyettesíteni.
- Hexadecimális számokban kisbetűket *kell* (MUST) használni.

Ebben a könyvben a továbbiakban törekszünk a fentiek betartására, de címekben (vagy definíciós címkékben) szándékosan csupa nagybetűt használunk, és máshonnan átvett anyagokat (pl. ábrák, RFC részletek) sem módosítunk.

2.1.2. Az IPv6 címzési architektúrája

Az IPv6 címtartomány különféle célokra való felosztását prefixek segítségével lehet megadni. A következő főbb kategóriákat definiálták (RFC 4291):

Cím típusa	Bináris prefix	IPv6 jelöléssel
Unspecified	00...0 (128 bit)	::/128
Loopback	00...1 (128 bit)	::1/128
Multicast	11111111	FF00::/8
Link-Local unicast	1111111010	FE80::/10
Global Unicast	(minden más, ami nem speciális)	

A következőkben az egyes kategóriákat részletesen megvizsgáljuk.

Speciális IPv6 címek/prefixek

Az alábbi két címet minden host használja:

::/128 – Unspecified Address
Meghatározatlan cím, akkor használjuk, amikor egy host inicializálja magát.

::1/128 – Loopback Address
A 127.0.0.1 IPv4 loopback címhez hasonlóan használt loopback cím.

FF00::/8 – Multicast Address
Az IPv4-ben broadcastnak és multicastnak nevezett funkciók megvalósítására egyaránt használt címtartomány. Részletesen foglalkozunk vele.

FE80::/10 – Link-Local IPv6 Unicast Addresses
Olyan címek, amik csak egy adott fizikai linken érvényesek, például egy Ethernet szegmensen. Az ilyen forráscímmel rendelkező IPv6 csomagokat az útválasztók (routerek) nem továbbítják. Olyan esetekben használjuk, amikor például egy IPv6 hálózatban nincs router, vagy Neighbour Discovery (lásd később) esetén. Minden hálózati interfészhez kötelezően tartozik ilyen cím. Amikor ilyen címet használunk egy programban forráscímként, akkor kötelezően meg kell adni, hogy melyik interfészen menjen ki a csomag (hiszen a link-lokális címből nem derül ki).

FEC0::/10 – Site-Local IPv6 Unicast Addresses – ÉRVÉNYTELEN!

Ez a tartomány ma már nem használt, érvénytelenített. Eredetileg az *egy adott site körzetén belüli* címzésre definiálták (mint IPv4-nél az RFC 1918 szerinti lokális címeket), de a site fogalmának nehéz meghatározhatósága miatt már nem használatos, lásd: RFC 3879.[2]

FC00::/7 – Unique Local IPv6 Unicast Adresses

Az RFC 4193 definiálja. Az *egyedi lokális címek* (a továbbiakban ULA) használatával olyan helyeken is lehet IPv6-ot használni, ahol nincs hivatalosan kiosztott IPv6 prefix, azaz felhasználásuk célja teljesen hasonló az RFC 1918-ban meghatározott privát IPv4 címekhez. Nagy különbség azonban, hogy az ilyen címeknél a *hálózatcím* (Network ID) meghatározásához egy *álvéletlen* (pseudo-random) számot használunk, amit az RFC-ben megadott módon lehet előállítani. Több szervezeti egység is generálhat így magának címet, és a pseudo-random algoritmusnak köszönhetően nagyon kicsi valószínűséggel fognak ezek a címek ütközni. (Segítségükkel tehát sikerült kiküszöbölni az érvénytelenített site-local unicast címek hibáit.)

Felépítésük az alábbi:

```
| 7 bits |1|  40 bits   |  16 bits  |          64 bits           |
+--------+-+------------+-----------+----------------------------+
| Prefix |L| Global ID  | Subnet ID |        Interface ID        |
+--------+-+------------+-----------+----------------------------+
```

Ahol:

- Prefix: fc00::/7
- L: kötelezően 1 értékű
- Global ID: véletlenszerűen választott, igen jó eséllyel globálisan egyedi[3]
- Subnet ID: a hálózatcím használója osztja ki a hálózatainak
- Interface ID: a hálózati interfész 64 bites azonosítója, lásd később

Az *IPv4-ről IPv6-ra való átállás* (IPv6 transition) támogatására szánták az alábbi IPv6 címtípusokat azzal a céllal, hogy IPv6 hálózatokban IPv4 címeket reprezentáljanak:

::/96 – IPv4-Compatible IPv6 Addresses – ÉRVÉNYTELEN!

IPv6 hálózatokban az x.y.z.w IPv4 címet reprezentálta volna a következő alakban: ::x.y.z.w (az elején 96 db 0 értékű bit volt). Eredetileg az IPv4 → IPv6 átmenethez való felhasználásra szánták, de már más megoldást (lásd következő) találtak ki, ezért érvénytelenítették.

::FFFF:0:0/96 – IPv4-Mapped IPv6 Addresses (ez használható!)

IPv6 hálózatokban az x.y.z.w IPv4 címet reprezentálja a következő alakban: ::ffff:x.y.z.w (az elején 80 db 0 értékű bit van). A korábbi ::x.y.z.w helyett azért kellett másik, mert változott a módszer, amit az IPv4 → IPv6 átmenethez szeretnének alkalmazni.

---

[2] Az RFC azt is leírja, hogy a másik probléma, ami IPv4-nél is előjött, az az, hogy a lokális(nak szánt) IP-címekkel rendelkező datagramok esetenként mégis kiszivárognak, ilyenkor aztán mindenféle gondot okoznak, ráadásul még azt sem lehet megtalálni, hogy honnan jöttek.

[3] Megjegyzés: létezik olyan szolgálatás (https://www.sixxs.net/tools/grh/ula/), ahol generálnak számunkra ULA IPv6 címet, és be is lehet azt jegyeztetni, de mivel használata nem kötelező, ezért semmilyen garanciát sem nyújt a cím egyediségére.

Megjegyzés: a fenti prefix végén szükséges volt kiírni a két darab 16 bites (csupa 0 értékű) csoportot jelölő „:0:0” részt, mert a „::ffff/96” jelölésnél a 16 darab 1 értékű bit a 128 bites IPv6 cím legalsó 16 bitjének a helyére kerül. (Lásd az RFC 6890 hibajegyzékében a 2. bejegyzést.)

Hálózati alkalmazások készítése esetén ennek a címtípusnak a használatával lehetséges az, hogy egyetlen socketet használjunk mind IPv6, mint IPv4 kommunikációra. Ilyenkor elegendő a socket kezelő függvényeknek csak az IPv6-os verzióját használni. Ha IPv4-mapped IPv6 címeket adunk meg, akkor a (Linux) kernel IPv4-es csomagokat küld ki, illetve ha az alkalmazás által figyelt portra IPv4 csomag érkezik, akkor ilyen IPv6 címekkel kapja meg az alkalmazás a csomagot. Ezzel a megoldással igen jelentős mennyiségű programozási munkát takaríthatunk meg.

2001:DB8::/32 – IPv6 szabványos dokumentációs prefix

Annak érdekében, hogy a dokumentációkban szereplő példák ne okozzanak zavart (a fejekben) vagy működő rendszerekkel való ütközést, lefoglaltak egy globális unicast prefixet kifejezetten dokumentációs célra. Ez a 2001:db8::/32 (RFC 3849). Ezt a prefixet soha senkinek sem fogják kiosztani és nem is routolják. De természetesen ettől még nem számít lokális unicast prefixnek!

(Így ha egy példában ezt a prefixet használják, és valaki a példát szó szerint begépeli, az nem fog kárt okozni, mert a prefixet a valós életben soha semmire nem használjuk. A prefixet helyi hálózatokban is kiszűrhetik; ezért senki ne számítson rá, hogy működni fog, ha használni próbálná!)

Megjegyzés: IPv4-nél is vannak ilyenek, az RFC 5737 szerint: 192.0.2.0/24, 198.51.100.0/24, 203.0.113.0/24.

A speciális célú IPv4 és IPv6 címekről összefoglaló található: RFC 6890

Az IPv6 multicast címzése

Fontos, hogy IPv6-ban az IPv4-gyel ellentétben broadcast cím nincs. Multicast címzésre az ff00::/8 tartomány használható. A multicast címek felépítése az alábbi:

bitek száma	8	4	4	112
mező neve	*prefix*	*flags*	*scope*	*group ID*

Az egyes mezők jelentése:

- **prefix**: a cím csoportcím voltát jelző *előtag*, értéke: ff
- **flags**: ebben a mezőben *jelzőbitek*et definiáltak: 0, R, P, T
- **scope**: a cím érvényességének a *hatókör*ét fejezi ki (0-f)
- **group ID**: az egyes *multicast csoportok azonosítására* használható bitek

A flags mezőben található jelzőbitek értelmezése:

- **0**: fenntartott (reserved)
- **R**: A csoporthoz tartozó *randevú pont* (Rendezvous point) címe a csoportcímbe beágyazott-e (RFC 3956), 1: igen, 0: nem.
- **P**: A csoportcímet az azt létrehozó szervezet *hálózatának előtagja* (Prefix) alapján generálták-e (RFC 3306), 1: igen, 0: nem.
- **T**: A csoportcím dinamikus-e (Transient), 1: igen, 0: nem, azaz az IANA által kiosztott well-known multicast address. Lásd: http://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xml

A csoportcímek érvényességi körét kifejező 4 bites scope mező lehetséges értékei:

- **0**: Reserved – fenntartott
- **1**: Interface-local – multicast loopback átvitelére használható

- **2**: Link-Local – hatóköre a fizikai hálózat broadcast domainje
- **4**: Admin-Local – hatóköre a lehető legkisebb értelmes adminisztratív tartomány
- **5**: Site-Local – hatóköre egy fizikai telephely
- **6-7**: Unassigned – a hálózati adminisztrátorok definiálhatják
- **8**: Organization-Local – egy szervezet összes telephelyére kiterjed
- **9-D**: Unassigned – a hálózati adminisztrátorok definiálhatják
- **E**: Global – globális
- **F**: Reserved – fenntartott

Az IANA által kiosztott néhány általános célú csoportazonosító (group ID):

- **FF02::1** – link local all nodes – az összes eszköz az adott fizikai hálózaton (broadcast domainben)
- **FF02::2** – link local all routers – minden útválasztó a fenti körben
- **FF02::5** – link local all OSPF routers – minden OSPF útválasztó a fenti körben
- **FF02::D** – link local all PIM routers – minden PIM útválasztó a fenti körben
- **FF02::16** – link local all MLDv2 routers – minden MLDv2 útválasztó a fenti körben
- **FF05::2** – site local all routers – a telephely összes útválasztója

Vegyük észre, hogy az FF02::1 jelentése nem más, mint IPv4 esetén az aktuális hálózatra vonatkozó broadcast cím! Megjegyzés: IPv4-ben is van ilyen csoportcím: 224.0.0.1

Vannak minden scope esetén érvényes csoportazonosítók is, például:

- **FF0x::C** – SSDP – Az SSDP (Simple Service Discovery Protocol) által használt IPv6 csoportcím. (IPv4 alatt a 239.255.255.250 csoportcímre „szemetel" az SSDP.)

Meg kell ismernünk még egy fontos csoportcímet, amelyre később fontos szerep hárul majd. Ez a kérdéses *eszköz* (node) IPv6 címének felhasználásával képzett link-lokális hatókörű *solicited-node multicast address*. Előállításához az ff02:0:0:0:0:1:ff00::/104 prefixhez kell hozzáírni a kérdéses node IPv6 címének legkisebb helyiértékű 24 bitjét. Például a 2001:db8::800:abba:edda IPv6 címhez tartozó ilyen multicast cím: ff02:0:0:0:0:1:ffba:edda. Mivel egy eszköznek csatlakozni kell az összes IPv6 címéhez tartozó solicited-node multicast csoporthoz, a csoportcímnek az IPv6 cím utolsó 24 bitjéből való képzése előnyös lehet, ha az interfész többféle prefix használatával képzett IPv6 címmel is rendelkezik[4], ugyanis ekkor csak egyetlen csoportról van szó (amennyiben az IPv6 címeinek utolsó 64 bitjét a hálózati interfész azonosítójából képezték). Azt pedig a későbbiekben fogjuk látni, hogy egy adott multicast csoportban nem fognak túlságosan sokan tartózkodni, sőt gyakran csak egy hálózati interfész tartozik bele (az összes IPv6 címével).

Itt említjük még meg, hogy az IPv6 multicast Ethernet szinten a csoportcímben a „33:33" prefixet használja, utána pedig az IPv6 csoportcím utolsó 32 bitje következik. (Tehát fentiek alapján a solicited node multicast address esetén az Ethernet csoportcím prefixe „33:33:ff" lesz.)

Az IPv6 Anycast címzése

Az IPv6 címzési arcitektúrája háromféle címtípust sorol fel: unicast, multicast és anycast. Az anycast címek szintaktikailag unicast címek, csak a használatuk módja eltérő. Míg unicast esetén minden hálózati interfész címe különböző, addig anycast esetén ugyanazt a címet több különböző számítógép hálózati interfészéhez is hozzárendelik. Egy anycast címre küldött csomagot az adott anycast címmel rendelkező interfészek közül (a hálózatban használt routing protokoll metrikája szerint) a forráshoz legközelebb található interfésznek kézbesítik. (Az anycast címzést IPv4-nél is alkalmazzák.)

[4] Ami meglehetősen tipikus, hiszen egy interfésznek mindenképpen van egy link-lokális IPv6 címe, valamint célszerűen egy globális (vagy legalább egy ULA) IPv6 címe is. Sőt egy interfésznek több globális IPv6 címe is lehet.

Ismereteink rendszerezésére tekintsük most át a címzési módszereket!

- **Unicast**: A csomag pontosan egy általunk kiválasztott állomásnak szól.
- **Broadcast**: A csomag (az adott hálózaton vagy tartományban) az összes állomásnak szól.
- **Multicast**: A csomag egy csoport összes tagjának szól.
- **Anycast**: A csomag egy csoport tagjai közül egynek szól, de nem a feladó, hanem a hálózat dönti el, hogy melyik legyen az. Tipikus választás, hogy legyen a küldőhöz legközelebb eső csoporttag.

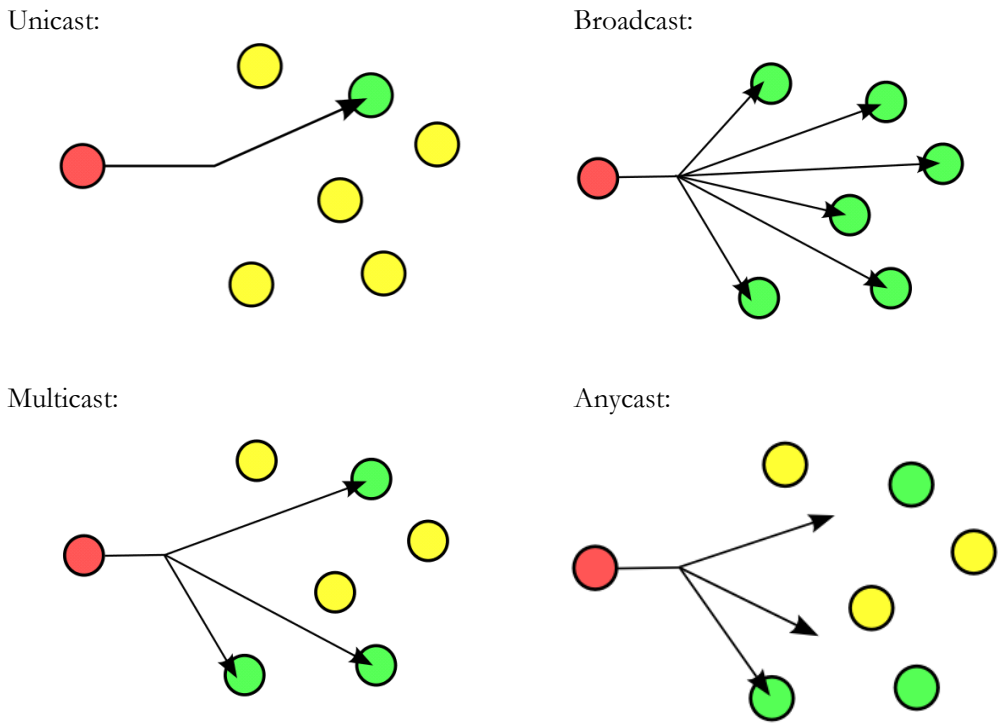
Az egyes módszerek működését illusztrálja az 1. ábra.

Az anycast címzés megvalósítása:

- Ugyanazt az IP-címtartományt (közönséges unicast címek!) több helyen is alkalmazzák és hirdetik az Interneten (BGP-vel). (Természetesen akár egy szolgáltató hálózatán belül is alkalmazható az Anycast címzés.)
- Az IP datagramok a legközelebbi célhoz fognak megérkezni.

Az Anycast címzés használata a gyakorlatban:

- DNS kiszolgálók esetén: ugyanolyan nevű (IP-című) legfelső szintű névkiszolgáló a világ több pontján is létezik. A kliensek a legközelebbit érik el.
- Az *IPv4-ről IPv6-ra való átállás* (IPv6 transition) során használt 6to4 megoldás is ezt használja a legközelebbi 6to4 átjáró elérésére.
- *Tartalomszolgáltató hálózat*ok (CDN: Content Delivery Network) is alkalmazzák abból a célból, hogy a kliens a legközelebbi szerverről tudja letölteni a médiatartalmat.



1. ábra: Címzési módszerek illusztrációja

Globális Unicast címek

Történeti érdekesség: Mivel az IPv6 címek 128 bitesek, kellően sok van belőlük. Tervezéskor nem látták előre, hogy milyen szempontok fognak felmerülni a címek struktúrájának kialakításával kapcsolatban. Két szempont kellően logikusnak tűnt:

- támogassuk a kettőnél több szintű struktúrát
- hagyjunk szabadságot arra, hogy mi legyen a strukturálási szempont

Természetesen a tervezőknek valamerre el kellett indulniuk, így eredetileg definiáltak például olyan címcsoportot, ahol földrajzi alapon és olyat is, ahol szolgáltatók szerint osztják ki a hálózatokat. Akkor úgy tűnt, hogy ez jól szolgálja az aggregálhatóságot, de aztán felülbírálták és megszüntették. Jelenleg inkább a Regional Internet Registryknek (RIR) osztanak ki nagyobb tartományokat és azok osztják szét saját hatáskörben a kérelmezőknek.

Bár jelenleg az IANA még csak a 2000::/3 tartományból delegált a RIR-eknek, ez elvileg semmiben sem különbözik a többi globális unicast IPv6 címtartománytól.

A jelenleg érvényes szabvány az RFC 3587: „IPv6 Global Unicast Address Format".

Általános felépítésük:

```
|         n bits         |   m bits   |        128-n-m bits         |
+------------------------+------------+-----------------------------+
| global routing prefix | subnet ID |        interface ID          |
+------------------------+------------+-----------------------------+
```

Ahol az egyes mezők:

- global routing prefix: hierarchikus felépítésű, az aggregációról a RIR-ek, LIR-ek és az ISP-k gondoskodnak, a site-ok az ISP-ktől kapják
- subnet ID: hierarchikus felépítésű, az aggregációról a site-ok adminisztrátorai gondoskodnak
- Interface ID: a hálózati interfészt azonosítja

Az IPv6 címzési architektúráját definiáló RFC 4291 szerint a 0000/3 tartomány kivételével az Interface ID mérete 64 bit. Így felépítésük az alábbi:

```
|         n bits         | 64-n bits |       64 bits               |
+------------------------+-----------+-----------------------------+
| global routing prefix | subnet ID |       interface ID          |
+------------------------+-----------+-----------------------------+
```

Ha a jelenleg használt 2000::/3 tartományban az egyes szervezetek a már elavult (obsolete) RFC 3177 által ajánlott /48-as tartományokat kaptak, akkor a címeik felépítése a következő:

```
| 3 |     45 bits        |  16 bits  |       64 bits               |
+---+--------------------+-----------+-----------------------------+
|001|global routing pre.| subnet ID |       interface ID          |
+---+--------------------+-----------+-----------------------------+
```

Például a BME tartománya is ilyen méretű, a prefixe: 2001:738:2001::/48

A címek szerkezetét korábban az RFC 2374 írta le „An IPv6 Aggregatable Global Unicast Address Format" címmel. Erről fontos tudni, hogy *elavult* (obsolete), ezért nem használjuk.

A címallokáció megfontolásai

Az *elavult* (obsolete) RFC 3177 szerint a site-ok számára az alapértelmezett allokációs egység mérete a /48 volt. Az új irányelvet az RFC 6177 „IPv6 Address Assignment to End Sites" fogalmazza meg. Ezt röviden az alábbiakban foglaljuk össze:

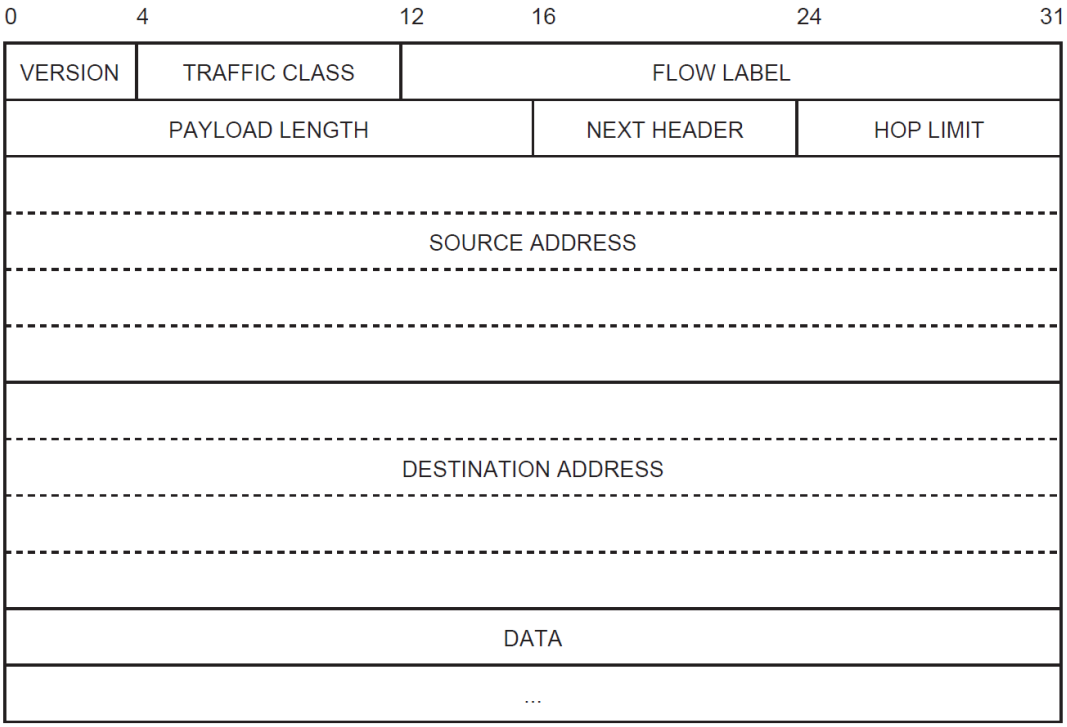
- Kis felhasználóknál nem szükséges a pazarló /48 méret.
- A /64 viszont nem elég, mert idővel több fizikai hálózatra lehet szükség.
- Szempont a takarékosság is, de az is, hogy később se legyen szükség NAT-ra (a site kapjon elegendő címet).
- A reverse DNS feloldás szempontjából megfontolandó a 4 bites határra való illeszkedés.
- Megemlíti az egyes RIR-ek gyakorlatát (/56-os méret), de alapelv, hogy nincs új alapértelmezett méret.
- Bár a /128 bizonyos esetekben elfogadott, site esetén semmiképpen sem javasolt.

2.2. Az IPv6 datagram felépítése

Az IP 6-os verziójában a címek méretének növekedése mellett a másik jelentős változás a 4-es verzióhoz képest az, hogy jó néhány mező hiányzik. Így például a kötelező, fix fejrészből teljesen hiányoznak a tördelést szolgáló mezők, a tördelés lehetőségét a hálózati átvitel közben teljesen meg is szüntették (csak a forrás tördelhet, lásd a fejrész kiterjesztéseknél). Ugyancsak hiányzik az ellenőrző összeg, erre a mai átviteli megoldások mellett nincs igazán szükség, és így nem lassítja feleslegesen a routereket. A kötelező fejrész hossza fix, így annak hosszát sem kell megadni. Opciók ettől még lehetségesek, ezt ügyes trükkel oldották meg (lásd: next header). Ennyi bevezető után nézzük meg egy, csak a kötelező fejrészt tartalmazó IPv6 datagram felépítését (2. ábra).

Az IPv6 datagram mezőinek jelentése:

- **Version** (4 bit) – verziószám – Értéke természetesen: 6, bár érdemben nem használjuk, lásd később.
- **Traffic Class** (8 bit) – forgalmi osztály – Az IPv4 *Type of Service* mező utódja. Az RFC 2474 szerinti *Differentiated Services* megoldást használjuk.
- **Flow Label** (20 bit) – folyamcímke – Adott forrástól adott cél IP-cím (utóbbi multicast vagy anycast is lehet) felé irányuló forgalmon belül *folyam*ok (flow) elkülönítését és kezelését hivatott támogatni. Jelenleg érvényes specifikációja: RFC 6437.
- **Payload Length** (16 bit) – adatmező hossza – Mivel 16 bit hosszú, így értéke legfeljebb 65535 lehet. Az IPv4-gyel ellentétben itt a fejrész 40 oktettje nem számít bele, ezt fejezi ki a neve is: „hasznos teher hossza".
- **Next Header** (8 bit) – következő fejrész – Megmutatja, hogy mit hordoz az IP címek utáni rész. Ez a mező egyrészt tartalmazhatja az IPv6 fölötti protokoll típusának megadását (ilyen értelemben az IPv4 *Protocol* mezőjének utódja), másrészt ezzel ki lehet terjeszteni a fejrészt, ahova további mezők kerülhetnek (ilyen értelemben az IPv4 *Options* mezőjét is helyettesíti). Ez utóbbi esetben a *fő fejrész* (a címekkel befejeződő 40 oktett) után jön egy *következő fejrész*.
- **Hop Limit** (8 bit) – átugrás korlát – Funkciója megfelel az IPv4 *Time To Live* mezőjének, de az elnevezése jobban kifejezi a funkcióját, és a mértékegysége sem másodperc (hanem darab).
- **Source Address** (128 bit) – forrás IPv6-os címe – Az IPv6 címekkel már részletesen foglalkoztunk.
- **Destination Address** (128 bit) – cél IPv6-os címe



2. ábra: IPv6 datagram felépítése

Megjegyzések:

1. Elvileg a *verziószám*ból derülne ki, hogy az utána következő mezőket nem IPv4, hanem IPv6 mezőknek kell tekinteni. Gyakorlatilag nem így történik, mert már a *hordozóhálózat*-ban (link layer) az IPv4-től (0x0800) eltérő EtherType protokoll azonosítót használnak az IPv6-ra (0x86DD).
2. Egy folyam klasszikus definíciója például a következő öt számmal adható meg: forrás és cél IP-címek, az IP fölötti protokoll száma (TCP vagy UDP), valamint a forrás és cél portszámok. A folyamcímke lehetőséget ad arra, hogy a folyamot sokkal egyszerűbben, tisztán az IPv6 protokoll mezői alapján azonosítsuk. (Esetünkben a két IP-cím és a folyamcímke egyértelműen azonosítják a folyamot.)
3. Az adatmező hosszát alapesetben a 16 bites érték erősen korlátozza, de lehetőségünk van úgynevezett jumbogram kialakítására is a Jumbo Payload option segítségével (RFC 2675). Ez 64kB-nál jóval nagyobb csomag is lehet, így a routereknek nem kell 64kB adatonként a fejrészt feldolgozniuk, ezáltal jelentősen gyorsulhat az adatátvitel. Ugyanakkor a nagy méretű csomag sokáig foglalja a linket, ami szolgáltatásminőségi (QoS) problémákat vethet fel.

2.2.1. Az IPv6 fejrész kiterjesztése

A fejrész kiterjesztése nagyon szellemes megoldás[5]: mivel a next header mező mérete 8 bit, és az IP fölötti protokollok száma messze elmarad a 8 biten kifejezhető 256-tól, ezért jó néhány értéket

[5] Ugyanakkor számos kockázattal is jár. Növeli az útválasztók terhelését, problémát okozhat a tűzfalaknál, különféle támadásokra ad lehetőséget, például az RFC 7113 2.1. pontjában is található ilyen.

más célra használhatunk: ezekkel különféle fejrész kiterjesztéseket adhatunk meg. Közülük néhányat már az RFC 2460-ban definiáltak:

- **Hop-by-Hop Options** (0) – Ha van ilyen, akkor ennek kell a legelsőnek lennie. Amint a nevéből is kiderül, ezt minden egyes routernek (hop) meg kell vizsgálnia (míg az egyebeket általában csak a célnál).
- **Routing** (43) – Eredetileg a 0-s típusú változatát definiálták (Type mező értéke 0, röviden RH0-nak is nevezik), ami az IPv4 *Loose Source and Record Route* opciójának a megfelelője volt, segítségével olyan csomópontok IP-címeit lehetett megadni, amin a csomagnak keresztül kell haladnia. Mivel ez kiváló lehetőséget nyújtott hálózati torlódást előidéző, tulajdonképpen *szolgáltatásmegtagadás* (Denial of Service, DoS) típusú támadásra, ezért érvénytelenítették (RFC 5095). Mobil IPv6-hoz a 2-es típusa használható.
- **Fragment** (44) – Csomagok tördelésére lehet használni, de erre IPv6-ban kizárólag a forrásnál van lehetőség. A tördelés megvalósítására használt mezők teljesen hasonlóak, mint IPv4 esetén, mélyebben nem foglalkozunk vele.
- **Destination Options** (60) – Az általa szállított információt csak a cél csomópontban kell megvizsgálni. (Multicast esetén több cél is lehet.)

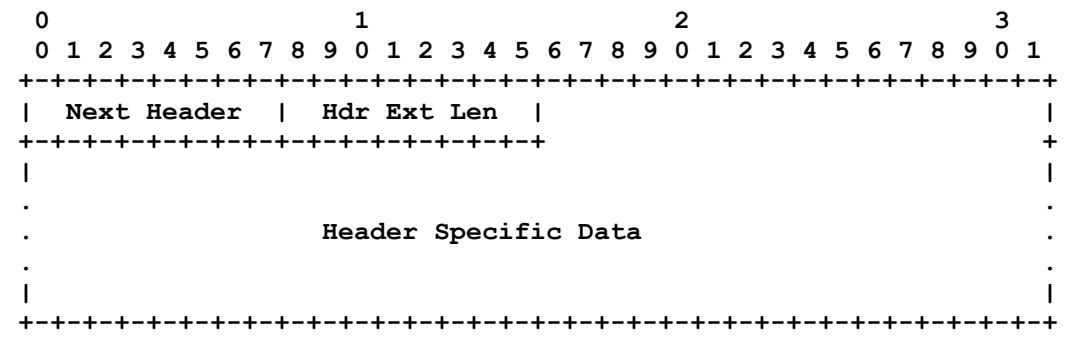
A fentieken túl még két fontos fejrészkiterjesztést említünk meg:

- **Authentication** (51) – Az adatok *eredet*ét (origin) és *változatlanság*át (integrity) igazolja és *visszajátszás* (replay) ellen is véd (RFC 4302).
- **Encapsulating Security Payload** (50) – Az adatok *titkosság*ának (confidentality), *változatlanság*ának (integrity) és (áttételesen) *eredet*ének (origin) védelmére használható (RFC 4303).

Megjegyezzük, hogy az utolsó kettőt használó IPsec (RFC 4301) eredetileg minden teljes IPv6 implementáció *kötelező* (MUST) része volt, de az RFC 6434 *erősen ajánlott*á (SHOULD) minősítette át azzal az indoklással, hogy vannak más megoldások is (például TLS, SSH), és az IPsec nem minden esetben az ideális megoldás.

Az egyes opciók szerkezetét, működését mélyebben nem vizsgáljuk, annyit jegyzünk meg róluk, hogy sorrendjük (közelítőleg) kötött, méretük oktettben kifejezve mindig 8-cal osztható, és az első oktettjük mindig a következő kiterjesztés vagy az IP fölötti protokoll azonosítója. (Vegyük észre, hogy IPv4 esetén 32-bites, IPv6 esetén pedig 64-bites egységekben gondolkozunk.)

Később újabb kiterjesztéseket is definiáltak, sőt a fejrész kiterjesztés általános formátumát is definiálták az RFC 6564-ben az alábbiak szerint (de ez csak az későbbiekre kötelező):



A fejrész kiterjesztés mezőinek jelentése:

- **Next header** (8 bit) – következő fejrész – A közvetlenül utána következő fejrészt azonosítja. Ez lehet további fejrész kiterjesztés vagy valamely az IP fölötti protokoll azonosítója.

- **Hdr Ext Len** (8 bit) – fejrész kiterjesztés hossza – Ez a 8 bites előjel nélküli egész szám az adott fejrész kiterjesztés méretét adja meg, 8 oktettes egységekben mérve úgy, hogy az első 8 oktettet nem számítjuk bele.
- **Header Specific Data** (változó hosszban) – kiterjesztés specifikus adatok – Kiterjesztéstől függően más-más adatok.

2.3. Internet Control Message Protocol version 6 (ICMPv6)

Az ICMP IPv6-os implementációja. Üzenetei az IPv6 felett utaznak (next header=0x3A, decimálisan 58), de minden IPv6 implementáció kötelező része! Aktuális dokumentáció: RFC 4443.

Üzenetei két típusba sorolhatók: hibaüzenetek és információs üzenetek.

Az ICMPv6 üzenetek formátuma egyedi. Ami közös bennük, az az ICMPv6 fejrész első 32 bitjén található három adatmező:

- **Type** (8 bit) – típus – 0-127: hibaüzenetek, 128-255: információs üzenetek
- **Code** (8 bit) – altípus – a típuson belül további fajtát jelölhet, ha egyáltalán van ilyen
- **Checksum** (16 bit) – ellenőrző összeg

A további rész kiosztása függ az üzenet típusától.

Az alábbiakban felsoroljuk a fontosabb ICMPv6 üzeneteket, az első szám a típus mező értéke.

- **1 – Destination Unreachable** – cél nem elérhető – Mint az azonos nevű ICMP üzenet.
- **2 – Packet Too Big** – a csomag mérete túl nagy – Az IPv6 útközben nem tördel. Ha egy csomag nem fér bele az MTU-ba, akkor a router eldobja, és visszajelzést küld.
- **3 – Time Exceeded** – időtúllépés – Mint az azonos nevű ICMP üzenet.
- **4 – Parameter Problem** – paraméter értelmezési hiba – A hiba okát a Code mezőben jelzi:
  - 0: Hibás IPv6 fejrész mező,
  - 1: Ismeretlen Next Header típus,
  - 2: Ismeretlen IPv6 opció.
- **128 – Echo Request** – visszhang kérés – Mint az azonos nevű ICMP üzenet.
- **129 – Echo Reply** – visszhang válasz – Mint az azonos nevű ICMP üzenet.
- **130 – Multicast Listener Query** – csoporttagok lekérdezése – Multicast cím megadása nélkül: mely multicast csoportoknak vannak tagjai az adott hálózaton? Multicast cím megadásával: a megadott című multicast csoportnak vannak-e tagjai az adott hálózaton? MLDv2 esetén pedig még forrás specifikus lekérdezésre is lehetőséget nyújt.
- **131 – Multicast Listener Report** (MLDv1) – csoporttagság jelzése – A csoporttagok ezzel az üzenettel jelzik igényüket a multicast forgalomra. (MLDv2-ben a 143-as típusú Multicast Listener Report vette át a funkcióját!)
- **132 – Multicast Listener Done** (MLDv1) – multicast csoportból kilépés – A csoporttagok ezzel az üzenettel jelzik, hogy már nem tartanak igényt az adott multicast csoport forgalmára. (MLDv2-ben a 143-as kódó Multicast Listener Report vette át a funkcióját, ami így kétféle funciót is megvalósít.)
- **133 – Router Solicitation** – router információ kérése – Az NDP része, részletesen tárgyaljuk majd.
- **134 – Router Advertisement** – router információ hirdetése – Az NDP része, lehet kérésre válasz, de a routerek kérés nélkül is hirdetik. A kéretlen hirdetés szándékosan nem pontosan periodikus.

- **135 – Neighbor Solicitation** – MAC-cím kérése – Az NDP része, az ARP megfelelője (pl. ARP Request és Probe funkciók).
- **136 – Neighbor Advertisement** – MAC-cím hirdetése – Lehet kérésre válasz, ekkor Solicited Neighbor Advertisement. Kérés nélkül is küldhető (pl. IP-cím változásakor), ez az Unsolicited Neighbor Advertisement.
- **137 – Redirect** – jobb útvonal közlése – Mint az azonos nevű ICMP üzenet.
- **143 – Multicast Listener Report** (MLDv2) – csoporttagság jelzése – Az MLDv2 protokoll esetén az MLDv1 131-es és 132-es típusú üzenetének funkcióját is ellátja.

2.4. A Neighbor Discovery Protocol (NDP)

Az NDP (Neighbor Discovery Protocol, RFC 4861) a TCP/IP referenciamodell internet rétegében működik. Működéséhez ICMPv6 protokoll üzeneteket használ, azaz az IPv6 fejrészben a *következő fejrész* (next header) mező értéke 58, és aztán az ICMPv6 fejrész *típus* (type) mezője azonosítja a konkrét üzentet (például: Neigbor Solicitation, Neighbor Advertisement).

Egy host a működéséhez szükséges azonosítókhoz, paraméterekhez különféle módon juthat hozzá: beállíthatók statikusan, megszerezhetők az állapotmentes automatikus címkonfiguráció (SLAAC, lásd lent), valamint a DHCPv6 segítségével.

Az NDP többek között képes:

- Állapotmentes automatikus címkonfigurációra
- Címfeloldásra (IPv6 címből MAC cím)
- Routerek címének megállapítására
- DNS szerverek címének megállapítására
- Címduplikáció ellenőrzésére, azaz annak vizsgálatára, hogy egy IPv6 címet már használnak-e (DAD: Duplicate Address Detection)
- Az adott linken érvényes prefixek és MTU kiderítésére

Küzülök néhányat a továbbiakban részletesen bemutatunk.

2.4.1. Állapotmentes automatikus címkonfiguráció

Az állapotmentes automatikus címkonfiguráció (SLAAC: Stateless Address Autoconfiguration, RFC 4862) lehetővé teszi, hogy a számítógépek (host) emberi beavatkozás nélkül kapjanak IPv6 címet állapot alapú kiszolgáló (DHCPv6) nélkül is. A folyamat több lépésből áll, a host először link-lokális IPv6 címet hoz létre, majd annak segítségével szerez egy routertől prefixet a globálisan egyedi IPv6 címhez. Mind a link-lokális, mind a globális IPv6 cím esetén az IPv6 cím utolsó 64 bitjében található Interface ID-t a hálózati interfész MAC-címéből állítja elő a *módosított EUI-64 cím*et előállító algoritmussal (lásd lent). Használat előtt mindkét IPv6 cím egyediségét ellenőrzi a DAD (Duplicate Address Detection) algoritmussal (külön tárgyaljuk).

Az SLAAC lépései:

- Link-lokális IPv6 cím generálása (fe80::/64 + módosított EUI-64 azonosító)
- Link-lokális IPv6 cím egyediségének ellenőrzése (DAD)
- Hálózati prefix kérése (ICMPv6 Router Solicitation)
    - Az üzenet küldéséhez forráscímként már használható a link-lokális IPv6-cím
    - A célcím pedig a link-lokális hatókörű (link-local scope) all routers IPv6 multicast cím (ff02::2)

- Hálózati prefix információ vétele (ICMPv6 Router Advertisement)
    - A router ezt a saját link-lokális IPv6 címéről általában a link-local scope all nodes IPv6 multicast címre (ff02::1) küldi. (De az RFC 4861 6.2.6. pontja szerint küldheti a host unicast címére is, ha az nem az unspecified (::) IPv6 cím.)
- Globálisan egyedi IPv6 cím (global unicast IPv6 address) előállítása (a kapott prefix + módosított EUI-64 azonosító). (Ehhez /64 hosszúságú kapott prefix szükséges!)
- Globálisan egyedi IPv6 cím egyediségének ellenőrzése (DAD)

Figyeljük meg, hogy a link-lokális és a globális unicast címek utolsó 64 bitje azonos lesz, hiszen mindegyikben a módosított EUI-64 azonosító szerepel.

Az SLAAC biztonsági kockázattal jár: hamis Router Advertisement üzenettel a kliens megtéveszthető, lásd:

- SLAAC Attack http://resources.infosecinstitute.com/slaac-attack/
- RFC 6104: „Rogue IPv6 Router Advertisement Problem Statement"

A módosított EUI-64 címet előállító algoritmus

Célja a hálózati interfész 48 bites MAC címéből 64 bites EUI (Extended Unique Identifier) azonosító létrehozása. Az algoritmus lépései:

- A 48 bites címet középen kettévágva a két fél közé beszúrjuk az FFFE 16 bites értéket.
- A MAC cím első bájtjának második legkisebb (azaz 2-es) helyiértékű bitjét 1-re állítjuk. (Ez a MAC cím OUI részének U/L bitje, tehát azt jelezzük vele, hogy nem univerzálisan egyedi, hanem lokálisan adminisztrált címről van szó.[6])

Az algoritmus működését egy példával illusztráljuk:

- Az eredeti MAC cím: 00:C1:C0:0B:0C:0D
- Az első lépés eredménye: 00:C1:C0:**FF:FE**:0B:0C:0D
- A második lépés eredménye: 0**2**:C1:C0:FF:FE:0B:0C:0D (kész)
- Az IPv6-nál használt alakban: 2C1:C0FF:FE0B:C0D

A kapott EUI-64 azosítót az IPv6-nál használt alakban egy 64 bites prefix után írva megkapjuk az IPv6 címet.

2.4.2. IPv6-cím egyediségének ellenőrzése

Az IPv6 *cím egyediségének ellenőrzése* (DAD: Duplicate Address Detection) elvi szinten hasonlóan történik, mint IPv4-nél az ARP Probe üzenettel. IPv6 esetén az ICMPv6 Neighbor Solicitation (NS) üzenetet használják, és ha nem érkezik rá válasz, akkor a címet még senki sem használja.

A megvalósítás viszont bonyolultabb. Ugyanis míg az NS üzenetben a forráscím (az ARP-hez hasonlóan) érvénytelen (::/128), addig a célcím a *vizsgált IPv6 címhez* (tentative address) tartozó *solicited-node multicast address*. Ennek haszna nyilvánvaló: míg IPv4 esetén az ARP *üzenetszórás*t (broadcast) használ, és így az adott *üzenetszórási tartomány* (broadcast domain) összes gépét terheli, addig az ND multicast használatával megkíméli a gépek döntő többségét. Az adott multicast csoportban elvileg többen is tartózkodhatnak, de SLAAC esetén ezek száma várhatóan alacsony. (Mivel a módosított EUI-64 azonosító utolsó 24 bitje alapján képezik a csoportcímet, ezért nem zárható ki, hogy egy adott hálózaton különböző gyártóktól származó hálózati interfészek esetén ez a 24 bit azonos legyen, de várhatóan nem lesz belőlük túlságosan sok.)

Megjegyzés: Azt már láttuk, hogy az Ethernet szinten a solicited node multicasthoz használt csoportcím prefixe: „33:33:ff". Amennyiben az IPv6 címet SLAAC használatával hoztuk létre, akkor

[6] Fontos, hogy ez a kitétel csak a MAC címre vonatkozik, ettől még a generált IPv6 cím lehet link lokális és globális is.

az utolsó 24 bitje pedig a módosított EUI-64 azonosító utolsó 24 bitje, ami viszont nem más, mint a hálózati interfész MAC címének az utolsó 24 bitje.

A DAD során a vizsgált címet használni kívánó interfészének még a fenti NS üzenet kiküldése előtt csatlakoznia kell két multicast csoporthoz, ezek az all-nodes multicast (azért, hogy ha valaki már használja a vizsgált címet, akkor megkapja a Neighbor Advertisement üzenetet) és a vizsgált címhez tartozó solicited-node multicast (azért, hogy ha valaki más is szeretné haszálni a vizsgált címet, akkor hallják egymást). Az RFC még számos egyéb feltételt és részletet leír (például a különböző véletlenszerűen megválaszott késleltetésekkel kapcsolatban), de ilyen szintű részletekbe most nem megyünk bele (a részletek megtalálhatók az RFC 4862 5.4. részében).

Végül, ha válaszként Neighbor Advertisement üzenet érkezik, akkor a vizsgált cím nem egyedi, tehát tilos (MUST NOT) az interfészhez rendelni (használni) és erősen ajánlott (SHOULD) az eseményt naplózni.

2.4.3. Címfeloldás

Bár a számítógépek IP-címmel azonosítják egymást, egy csomag tényleges (fizikai) elküldéséhez szükség van annak az interfésznek az adatkapcsolati szintű (link layer) címére (MAC-címnek is szoktuk nevezni) amely számára közvetlenül (link szinten) küldeni szeretnénk a csomagot szállító adatkapcsolati szintű adategységet, pl. Ethernet keretet. Az IP 4-es verziója erre a célra a jól ismert ARP-t alkalmazza. IPv6 esetén pedig a Neighbor Solicitation (NS) és Neighbor Advertisement (NA) ICMPv6 üzenteteket használjuk. (Az IPv6 címfeloldás részletes leírása az RFC 4861 7.2. részében található.)

Amikor egy (multicast képes) hálózati interfészt engedélyeznek, az köteles (MUST) csatlakozni az all-nodes multicast csoporthoz, valamint az összes IPv6 címéhez tartozó solicited-nodes multicast address által meghatározott csoporthoz is. Ha az IPv6 címe megváltozik, akkor köteles (MUST) tagja lenni az új IPv6 címéhez tartozó solicited-nodes multicast address cím által meghatározott csoportnak, illetve elhagyni az olyan csoportot, amiben már nem illetékes. Megjegyzés: amint már korábban említettük, lehetséges, hogy több különböző IPv6 címhez ugyanaz a csoportcím tartozik. (Például tipikusan ez a helyezet, ha egy fizikai interfészen többféle prefix alapján képzett interfésze is van, és ezek utolsó 64 bitjét a hálózati interfész azonosítójából képezték.) Így nem minden esetben történik ténylegesen csoportba való belépés illetve onnan való kilépés. Amennyiben ilyen történik akkor erre az MLD (Multicast Listener Discovery) protokoll 1-es vagy 2-es verzióját használják.

Az RFC számos lehetőséget leír, hogy NS/NA üzenetváltáskor milyen címeket használhatnak, itt most egy tipikus példát mutatunk be. A kérdező gép az NS üzenetet a saját link-lokális IPv6 címéről a kérdéses IPv6 címhez tartozó solicited-node multicast címre küldi, és az üzenet *Target Address* mezőjében benne van a kérdéses IPv6 cím, valamint a kérdező a *Source link-layer address* opcióval megadja a saját MAC címét is. Az IPv6 cím gazdája egy NA üzenettel válaszol, amit a saját link-lokális IPv6 címéről a kérdező link-lokális IPv6 címére küld, benne van a *Target Address* mezőben a kérdéses IPv6 cím, és a hozzá tartozó MAC cím pedig a *Target link-layer address* opcióban található.

2.5. Multicast Listener Discovery (MLD)

Amint a címzésnél már láttuk, az IPv6 kifinomultan támogatja a *többeskküldés* (multicast) megvalósítását, és amint az ND-nél láttuk, az IPv6 alaposan ki is használja a multicast lehetőségeit, kiváltva vele a broadcastot.

IPv6-ban az egyes állomások az MLD (Multicast Listener Discovery, MLDv1: RFC 2710, MLDv2: RFC 3810) protokoll segítségével tudják kifejezni igényüket valamely multicast csoport

forgalmára. (Az MLDv1 az IGMPv2-nek, az MLDv2 pedig az IGMPv3-nak az IPv6-os megfelelője. Az újabb verzió célja mindkét esetben az volt, hogy a vevők kifejezhessék azon igényüket, hogy az adott multicast csoport forgalmából csak egy adott forrás adására tartanak / nem tartanak igényt.)

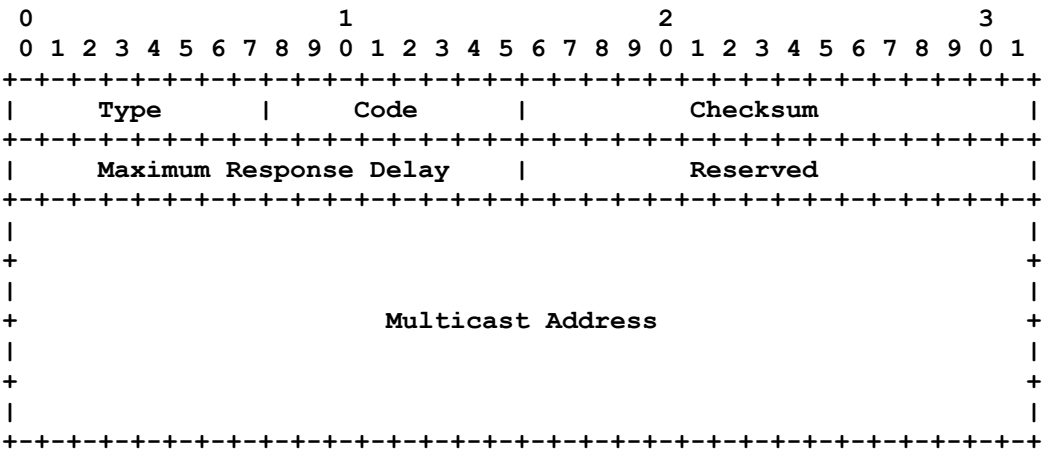
Az MLD ICMPv6 üzeneteket használ, azaz az IPv6 fejrészben a *következő fejrész* (next header) mező értéke 58, és aztán az ICMPv6 fejrész *típus* (type) mezője azonosítja a konkrét üzentet. Ezek a v2-ben a v1-hez képest változtak, az 1. táblázatban röviden összefoglaljuk, hogy melyik verzió milyen célra milyen üzeneteket használ. Az egyes üzenetek után megadjuk az ICMPv6 típus mező decimális értékét is, amire a különböző verziószámú, ezért eltérő Multicast Listener Report üzenetek azonosításához van szükség.

verzió	MLDv1	MLDv2
csoporttagság lekérdezése	Multicast Listener Query (130)	Multicast Listener Query (130)
csoportagság jelzése	Multicast Listener Report (131)	Multicast Listener Report (143)
csoportból kilépés	Multicast Listener Done (132)	Multicast Listener Report (143)

1. táblázat: Az MLDv1 és az MLDv2 által használt ICMPv6 üzentek típusa

Tehát a 2. verzióban a csoporttagság jelentésére és a csoportból való kilépés (kérdés nélküli) bejelentésére ugyanazt az üzenetet használják, amely azonban eltér az 1. verzióban a csoporttagság jelzésére használt üzenettől. A táblázatból nem látszik, de a 2. verzióban a lekérdezésre használt üzenet formátuma is változott (bővült).

Az MLDv1-ben mindhárom üzenet formátuma azonos:



Ahol az egyes mezők jelentése:

- **Type** (8 bit) – típus – Lehetséges értékei: 130, 131, 132. Az üzenet típusát határozza meg.
- **Code** (8 bit) – nem használják – Csak az ICMPv6 fejrész szerkezete miatt szerepel. A küldő az értékét 0-ra állítja, a vevő pedig nem veszi figyelembe.
- **Checksum** (16 bit) – ellenőrző összeg – A szabványos ICMPv6 ellenőrző összeg.
- **Maximum Response Delay** (16 bit) – maximális megengedett késleltetés – Csak a kérdésben (query) használják, értéke ms-ban adja meg, hogy mennyi időn belül kell a választ küldeni. A többi üzenetben az értékét 0-ra állítja a küldő és a vevő nem veszi figyelembe.
- **Reserved** (16 bit) – további fejlesztésre fenntartva – Értékét a küldő 0-ra állítja és a vevő nem veszi figyelembe.

- **Multicast Address** (128 bit) – multicast cím – Lekérdezés (query) esetén két lehetőség van. Ha az értéke 0, akkor *általános lekérdezés*ről (General Query) van szó, a kérdező arra kíváncsi, hogy mely csoportoknak vannak tagjai. Ha értéke 0-tól különböző, akkor *csoportcím specifikus lekérdezés*ről (Multicast-Address-Specific Query) van szó, a kérdező azt szeretné megtudni, hogy az adott című IPv6 multicast csoportnak van-e tagja. Válasz (Report) vagy kilépés (Done) esetén pedig azt a csoportot adja meg, amelyik forgalmára a válasz küldője az igényét bejelenti (Report) vagy éppen amelyiket el kívánja hagyni (Done).

A 2. verzió az 1. verzióval képes együttműködni (interoperable), a bővítés célja a *forrásspecifikus multicast* (SSM: Source-Specific Multicast, RFC 3569) támogatása. A lekérdezés (Query) üzenet típusa (type) nem változik, de számos további mezővel bővül, valamint a benne található Maximum Response Delay mező neve *Maximum Respons Code*ra változik, miközben értelmezése 0-32767 értékek esetén változatlan, fölötte viszont lebegőpontos számként kell értelemezni (lásd: RFC 3810 5.1.3 rész). A válaszüzenet típuskódja és formátuma is teljesen megváltozik. Ezeket a továbbiakban nem részletezzük. (A gyakorlatban találkozhatunk velük, a Wireshark képes dekódolni.)

2.6. Path MTU discovery (PMTUD)

A hálózati átvitel hatékonysága szempontjából kívánatos, hogy a lehető legnagyobb csomagméretet használjuk. Egy adott *útvonal legnagyobb megengedett csomagméretének kiderítése* (PMTUD: Path MTU Discovery) mind IPv4 (RFC 1191), mind IPv6 (RFC 1981) esetén felmerülő kérdés. Míg IPv4 esetén egy adott útvonalon a közbülső routerek tördehetnek, ha a DF (Don't Fragment) bit nincs beállítva, így IPv4-nél túl nagy csomagméret esetén „csak" a hatékonyság csökken (esetleg a csomagok összerakásánál lehet probléma, ha pl. egy tűzfal eldobja a töredékeket), addig IPv6 esetén kizárólag a forrás tördelhet, tehát a csomagméret megválasztása itt még fontosabb kérdés. Amennyiben IPv6 esetén egy csomag mérete túl nagy, és ezért egy közbülső router eldobja, akkor az adott router ezt a problémát a Packet Too Big (PTB) ICMPv6 üzenettel *köteles* (MUST) jelezni a forrás számára (lásd RFC 1885, 3.2). A PMTUD (első körben) ezen üzenetek használatán alapul.

Mivel az útvonalak időközben megváltozhatnak, a PMTUD-t időről időre újra el kell végezni. Amennyiben egy host PTB üzetetet kap, akkor *köteles* (MUST) az adott útvonalra küldött üzenetek méretét csökkenteni, és a közeljövőben (10 perc a szokásos idő, 5 percnél semmiképpen sem lehet kisebb) nem is küldhet ilyen méretű csomagot. (Az RFC a csökkentés konkrét mértékét nem határozza meg.) Egy adott útvonal MTU értékének csökkenésére tehát azonnal kell reagálni, az esetleges növekedést pedig 10 percenként ajánlott megvizsgálni.

Azonban előfordulhat, hogy valamely útvonalon egy tűzfal nem engedi át a PTB üzeneteket. Ez a gyakolatban például azt jelentheti, hogy egy TCP kapcsolat felépül ugyan, de aztán a forgalom nem halad át rajta. ICMPv6 üzenetek hiányában történő PMTUD-re ad megoldást az RFC 4821. A benne leírt PLPMTUD (Packet Layer PMDUD) alapötlete (TCP-ben gondolkozva) az, hogy a TCP szegmens méretet kis értékről kezdve növeli, és az első izolált csomagvesztést nem torlódásnak tekinti (nem csökkenti a congestion window értékét), hanem azt feltételezi, hogy a szegmens mérete túl nagy volt, ezért eldobásra került, de a PTB üzenet elveszett. A megoldással bővebben nem tudunk foglalkozni, csak jeleztük a probléma és a megoldás létét.

Mivel az 1280-as MTU-t minden IPv6 node-nak kötelezően támogatnia kell, és Ethernet esetén 1500 bájt az általános MTU, ezért a mindennapi életben az esetek többségében az 1280-1500 az MTU lehetséges intervalluma. Speciális esetben lehet komoly jelentősége a nagyobb MTU használatának.

2.7. Domain Name System

Az IPv6 protokoll alkalmazása magával hozta a DNS rendszer kiterjesztését is. Megjelent egy új erőforrás rekord típus (Resource Record, RR), valamint az IPv6 reverz delegációhoz egy új domain. A DNS működését leíró RFC 1034 és RFC 1035 került kiegészítésre az RFC 3596-tal.

2.7.1. IPv6 névfeloldás

Az IPv6 névfeloldáshoz bevezetésre került az „AAAA" rekordtípus. Könnyű megjegyezni, hiszen az IPv6 cím hossza éppen négyszerese az IPv4 cím hosszának, így az IPv4-nél alkalmazott „A" rekord helyett itt a „négy A" (quad A) rekord került bevezetésre. Az AAAA rekord alkalmazása teljesen megegyezik az IPv4-es párjával, annyi különbséggel, hogy IPv6 cím követi. Például egy WWW rekord megadása egy IPv6 címhez egy tetszőleges zónában így nézhet ki:

```
www     IN AAAA       2001:DB8::10
```

Ha egy hoszt rendelkezik IPv4 és IPv6 címmel is, akkor mindkét típusú rekordot létre kell hozni a zónában. Egy ilyen esetre példa az alábbi zónafájl részlet:

```
www     IN A          192.0.2.10
        IN AAAA       2001:DB8::10
```

A DNS szerver nem veszi figyelembe, hogy a kérés IPv4, vagy IPv6 protokollal érkezik-e hozzá, hanem a kérdezett rekordtípushoz tartozó választ adja vissza.

2.7.2. IPv6 reverz névfeloldás

Az IPv6-os címek fordított irányú (reverz) feloldása IP-címből szimbolikus név (DNS domain name) megállapítására szolgál, ami további döntések alapját is képezheti (például hozzáférés engedélyezése a domain alapján). Megvalósítása is az IPv4-es párjához hasonló annyi eltéréssel, hogy míg az IPv4 esetében az in-addr.arpa domaint kell használni, itt az ip6.arpa alkalmazása szükséges. A létrehozott zónában ugyanúgy a PTR típusú rekordokat kell használni. A domain létrehozásánál fontos, hogy minden négy bites csoportot jelölni kell, nem hagyhatóak el a nullák sem. A számjegyek sorrendjét pedig az IPv4-hez hasonlóan meg kell fordítani. Nézzük meg két példán keresztül, hogy hogyan állíthatjuk elő a szükséges domaint és bejegyzést:

Az IPv6 cím:	2001:db8::10
A rövidítés nélkül:	2001:0db8:0000:0000:0000:0000:0000:0010
Domain név:	0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa.

Az IPv6 cím:	2001:abcd:e001:1::2
A rövidítés nélkül:	2001:abcd:e001:0001:0000:0000:0000:0002
Domain név:	2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0.1.0.0.e.d.c.b.a.1.0.0.2.ip6.arpa.

Az első példához létrehozható a

```
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.d.b.0.1.0.0.2.ip6.arpa.
```

nevű zóna, a

```
0.1.0.0.0.0.0.0     IN     PTR    www.isp.hu.
```

PTR rekorddal, míg a másodikhoz a

```
0.0.0.0.0.0.0.0.1.0.0.0.1.0.0.e.d.c.b.a.1.0.0.2.ip6.arpa.
```

nevű zóna, a

```
2.0.0.0.0.0.0.0     IN     PTR    www.isp2.hu.
```

PTR rekorddal.

2.8. IPv6 címkiosztás

2.8.1. RIPE adminisztráció

Az interneten használt egyedi azonosítókat (IP címek, protokoll azonosítók, portok, domain nevek, stb.) az Internet Assigned Numbers Authority (IANA, http://www.iana.org/) adminisztrálja, mely az Internet Corporation for Assigned Names and Numbers (ICANN, https://www.icann.org/) nonprofit szervezet részlege. Az IANA az IPv4 és IPv6 címek (és AS számok) adminisztrálását földrajzi területek alapján a 2. táblázatban felsorolt öt regionális szervezethez delegálta (Regional Internet Registry, RIR).

rövidítés	név	terület	weboldal
AFRINIC	African Network Information Center	Afrikai régió	http://www.afrinic.net/
APNIC	Asia Pacific Network Information Centre	Ázsiai és csendes-óceáni régió	https://www.apnic.net/
ARIN	American Registry for Internet Numbers	Észak-amerikai régió	https://www.arin.net/
LACNIC	Latin America and Caribbean Network Information Centre	Latin-Amerika és Karib-szigetek	http://www.lacnic.net/
RIPE NCC	The Réseaux IP Européens Network Coordination Centre	Európa, Közép-Kelet, Közép-Ázsia	https://www.ripe.net/

2. táblázat: Az öt RIR adatai

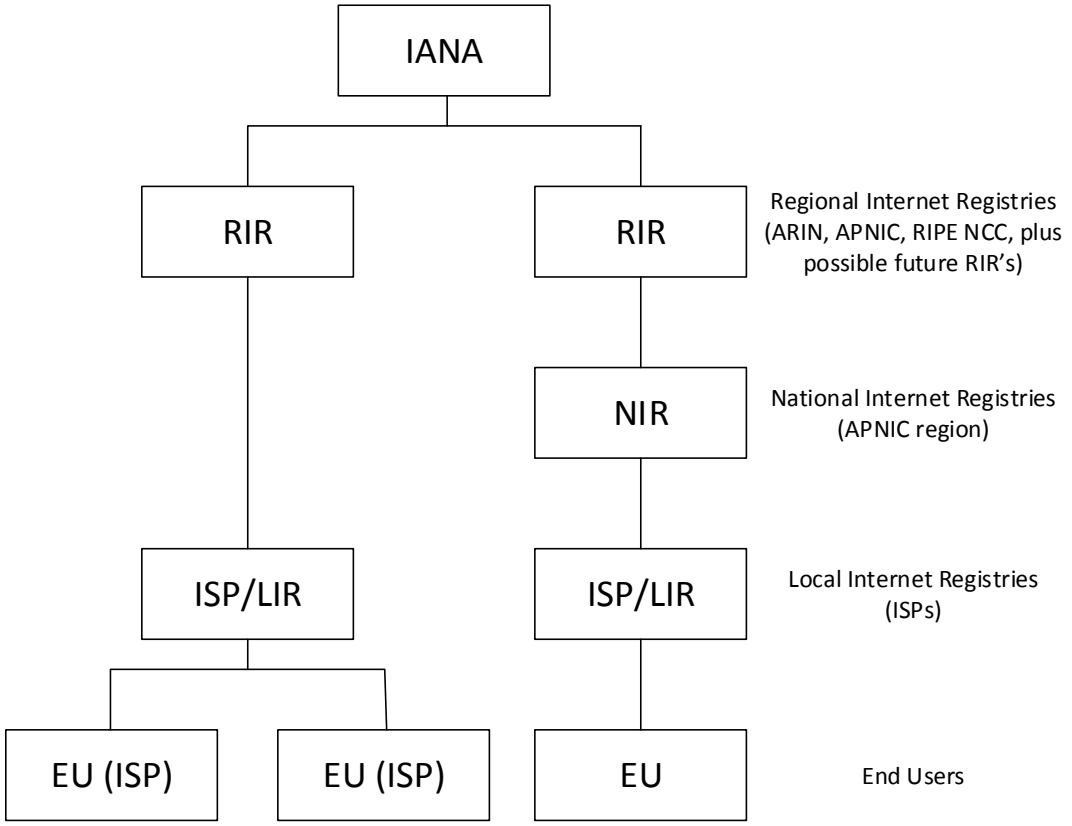
Az IANA viszonylag nagy önállóságot biztosított az 5 RIR részére, így az IP címhez jutás feltételei és módszerei területenként eltérőek. A következőkben a RIPE szabályaival foglalkozunk.

A RIPE Network Coordination Centre (RIPE NCC) az IPv6 címhez (és még más azonosítóhoz) jutást tagsághoz kötötte (mely természetesen tagdíjfizetéssel jár). Az IPv6 címekre vonatkozó szabályokat jelenleg a RIPE-641 (https://www.ripe.net/publications/docs/ripe-641) dokumentum szabályozza. Az IPv6 címeket a felhasználók nem közvetlenül a RIPE NCC-től kell, igényeljék, hanem az internetszolgáltatójuktól (ISP). A nagyobb ISP-k általában helyi internet regisztrátorok (Local Internet Registries, LIR) is egyben. RIPE NCC tagsággal rendelkeznek, és ők végzik az IPv6 címek adminisztrálását ügyfeleik számára. A címek adminisztrálásának hierarchikus felépítése látható a 3. ábrán.

Az adminisztráció megértéséhez fontos két fogalom pontos definíciója:

- Allokálás (Allocation): A címtartomány kiosztása IR-ek részére, hogy azok tovább oszthassák azokat.
- Kijelölés (Assignment): A címtartomány hozzárendelése ISP-hez, vagy előfizetőhöz kizárólag a meghatározott és dokumentált felhasználás céljából. A kijelölt címtartományból, annak felosztásával nem végezhetőek újabb kijelölések mások részére.

Tehát a csak allokált címtartományban lévő IPv6 címeket még nem használhatja senki, míg a már kijelölt címek (kizárólag a kijelölés céljára) már használhatók.



3. ábra: IPv6 címkiosztás hierarchikus felépítése (https://www.ripe.net/publications/docs/ripe-641)

Az IPv6 címtartomány kezelésének több, néhol egymásnak ellentmondó célja van:

- Gondosság (Prudent manner): Az IPv6 címtartomány egy közös erőforrás, melynek gondos kezelése szükséges.
- Egyediség (Uniqueness): Minden allokációnak és kijelölésnek biztosítania kell az IPv6 címek teljes világra kiterjedő egyediségét, hiszen ez szükséges az IPv6 internet problémamentes működéséhez, az egyedi hosztok azonosításához.
- Regisztráció (Registration): Az IPv6 címeket minden esetben rögzíteni kell egy adatbázisban, biztosítandó az egyediséget, és a visszakereshetőséget.
- Aggregálás (Aggregation): Mindenhol, ahol lehetséges, a hálózat felépítéséhez igazodva, hierarchikusan kell kiosztani az IPv6 címeket, ezáltal biztosítva az aggregálás lehetőségét. Az aggregálás alkalmazása csökkenti az útválasztókra jutó terhelést, azáltal, hogy kevesebb routing bejegyzés kerülhet a routing táblázatokba. (Az IPv6 protokoll esetében ez a szempont kerül előtérbe.)
- Megőrzés (Conservation): Annak ellenére, hogy az IPv6 címtér hatalmas, kerülni kell a pazarlást.
- Méltányosság (Fairness): Biztosítani kell az igazságos és egyenlő IPv6 címtér használatot az alkalmazott szabályok és gyakorlatok segítségével az internet jelenlegi és lehetséges felhasználói számára, helytől, nemzetiségtől, mérettől, és minden egyéb tulajdonságuktól függetlenül.
- Minimális költség (Minimised overhead): Minimalizálni kell az IPv6 címhez jutás terheit.

Fontos, hogy az egyedi IPv6 unicast címek nem kerülnek a felhasználók tulajdonába, azokat ők csak használatba kapják.

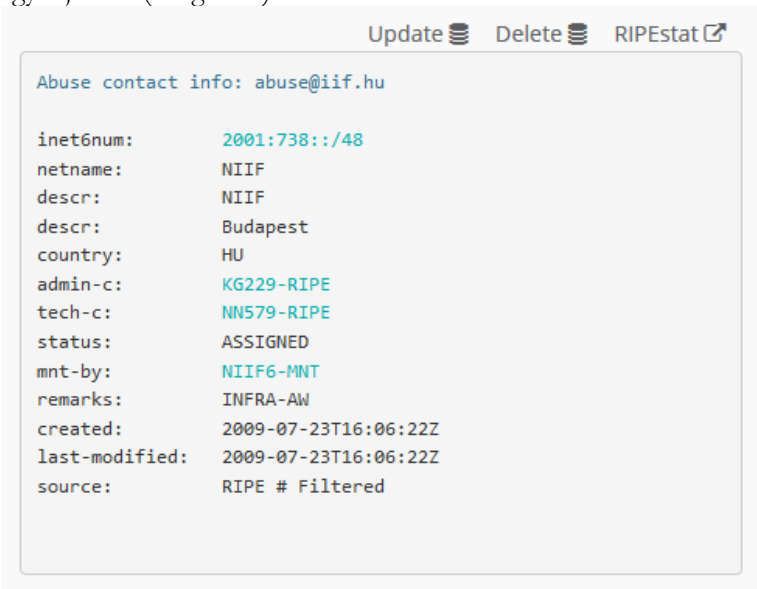
A legkisebb IPv6 allokáció prefixe /32. Ez azt jelenti, hogy ha egy LIR IPv6 címtartomány allokációt kér, ő legalább ekkora címteret kap. Az allokációnál figyelembe lehet venni a használt IPv4 címtartományokat, melyek indokolhatják a rövidebb IPv6 prefix (nagyobb címtartomány) allokációját is. Az IPv6 címek allokációjakor alkalmazott megoldás (nem közvetlen egymás után kerültek allokálásra a címtartományok) lehetővé teszi, hogy szükség esetén a /32-es tartomány (összefüggő) /29-esre bővíthető legyen. Ezt a bővítést jelenleg indoklás nélkül kérheti bármely LIR, ezáltal megnyolcszorozva a rendelkezésére álló címtartomány méretét.

A felhasználók részére biztosított prefix mérete az internetszolgáltató döntésén múlik, azonban egy végpontra a /64 (tehát az egy alhálózat) a legkisebb méretű címtartomány. A /64 és /48 hosszak közötti prefixek esetében nincs szükség a felhasználótól dokumentáció beszerzésére, azonban a /48-as prefixnél nagyobb címtartomány kijelölése esetében már a felhasználónak bizonyítania kell, hogy valóban szüksége van ilyen méretű tartományra. Ilyen esetekben a RIPE NCC hagyja jóvá az allokációt.

Az IPv6 címek kijelölését regisztrálni kell a RIPE NCC adatbázisában. A LIR ezt két féle módon csinálhatja. Ha az egyes felhasználók részére /48-nál nagyobb tartományt biztosít, úgy ezt a tartományt a felhasználó részére kell regisztálnia az adatbázisban. Ha /48 és /64 közötti tartományokat jelöl ki, azt nagyobb tartományokként is regisztrálhatja, azonban, ilyenkor meg kell adnia az „AGGREGATED-BY-LIR” attribútumot is az objektumhoz.

2.8.2. RIPE NCC adatbázis

Legegyszerűbben a RIPE weboldalán (https://www.ripe.net/) keresztül kérdezhetjük le az adatbázis tartalmát. Egy kijelölés (assigment) látható a 4. ábrán.



4. ábra: IPv6 assignment a RIPE adatbázisában

A saját objemtumok módosítását és törélést is elvégezhetjük a webupdate segítségével, ami az objektum melletti Update gombra kattintva töltődik be. (A webupdate mellett, e-mail-ben is módosíthatók az objektumok.) Természetesen a módosításhoz hitelesítésre is szükségünk van.

A reverze delegációt a /32-es prefixekhez a domain objektum segítségével rögzíthetjük az adatbázisban. Erre mutat példát az 5. ábra.

Update Delete

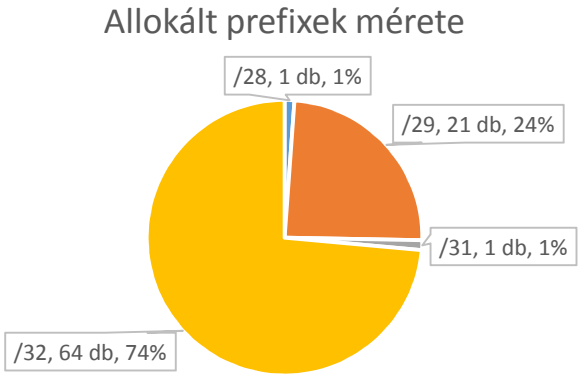
```
domain:          8.3.7.0.1.0.0.2.ip6.arpa
descr:           Reverse delegation for HU-HUNGARNET IPv6 zone
admin-c:         KG229-RIPE
admin-c:         MN24-RIPE
tech-c:          KG229-RIPE
tech-c:          JM12705-RIPE
zone-c:          KG229-RIPE
zone-c:          JM12705-RIPE
mnt-by:          NIIF6-MNT
created:         2002-09-13T08:16:26Z
last-modified:   2012-09-24T12:53:31Z
source:          RIPE # Filtered
nserver:         ns2.sztaki.hbone.hu
nserver:         kubiac.iif.hu
nserver:         ns2.iif.hu
```

5. ábra: IPv6 reverz delegáció

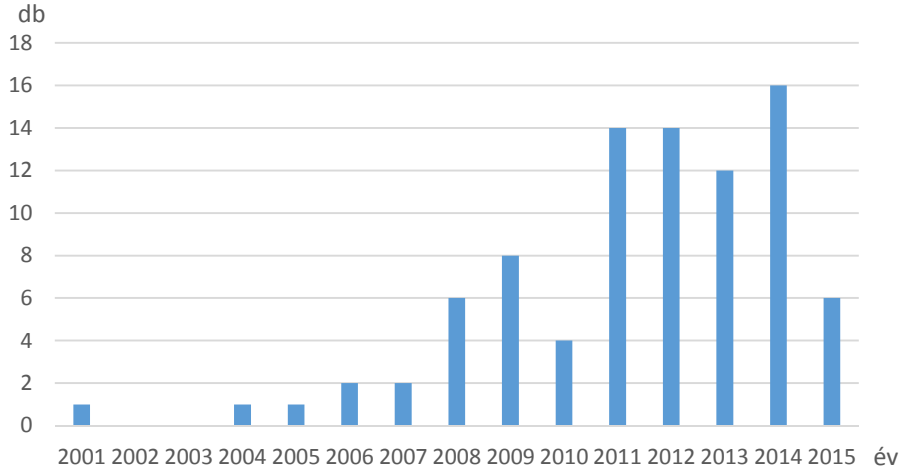
A RIPE NCC szerverén (ftp://ftp.ripe.net/ripe/stats/membership/alloclist.txt) mindig elérhető az egyes LIR-ek részére allokált IPv4 és IPv6 címtartományok listája. A könyv írásakor (2015.5.30.) 112 darab „hu.”-tal kezdődő LIR volt megtalálható. Ez a lista nem pontosan egyezik meg a hazánkban internetszolgáltatással foglalkozó cégekkel, azonban jó közelítést ad a magyarországi helyzetről. A „hu.” alatti adatok vizsgálatának eredményeit a 3. táblázatban foglaljuk össze.

hu. alatti LIR-ek száma	112 db
ebből IPv6 allokációval rendelkezők	86 db (76,79%)
csak IPv6 allokációval rendelkezők száma	1 db (0,89%)
két IPv6 allokációval rendelkezik	1 db (0,89%)

3. táblázat: A RIPE hu. alatti ISP-k IPv6 allokációi



6. ábra: A RIPE hu. alatti ISP-k IPv6 allokációinak mérete



7. ábra: RIPE hu. alatti ISP-k IPv6 allokációk éves eloszlása

Az IPv6 allokációk prefixhosszának eloszlását a 6. ábrán láthatjuk. Míg az IPv6 allokációk éves eloszlása a 7. ábrán látható.

Ezek az adatok bizakodásra adhatnak okot, hiszen a 112 LIR 77%-a már rendelkezik IPv6 címallokációval. Azonban, ha megvizsgáljuk, hogy a 87 darab IPv6 prefixből hány található meg a globális routing táblában, az eredmény sokkal rosszabb, mivel kiderül, hogy az allokált IPv6 prefixeknek csak kevesebb, mint felét használják.. A könyv írásakor a globális routing táblában közülük csak 33 darab prefix szerepelt. Az pedig az összes „hu." LIR-nek csak 29,46%-a, míg a 87 darab allokált prefixnek csak 37,93%-a.

2.8.3. Ajánlások

A korábban ismertetettek alapján, az előfizetői telephelyek részére kijelölt címtartomány ajánlott mérete a minimális /64 és a maximális /48 közé esik. Az átláthatóságot növeli (és a reverz DNS beállítását egyszerűsíti), ha a méretezésnél figyelembe vesszük a byte, vagy a félbyte (4 bit) határokat. Ezek alapján vizsgáljuk meg, hogy egy /32 prefixméretű kezdeti allokációt felhasználva egy LIR hány felhasználó részére tud IPv6 címet kijelölni.

kijelölt prefix méret	ügyfelek száma	ügyfelek lehetséges alhálózatainak száma
/64	2^{32}=~4,3 milliárd	2^{0}=1
/60	2^{28}=~286,4 millió	2^{4}=16
/56	2^{24}=~16,8 millió	2^{8}=256
/52	2^{20}=~1 millió	2^{12}=4096
/48	2^{16}=65536	2^{16}=65536

4. táblázat: Az ügyfelek és hálózataik lehetséges száma a kisztottt prefix méretének függvényében, ha a LIR prefixének mérete /32

A 4. táblázat összeállításánál nem vettük figyelembe, hogy az internetszolgáltatók saját infrastruktúrája is igényel IP címeket, valamint minden ügyfél részére ugyanakkora tartomány került kijelölésre. A RIPE NCC jelenlegi szabályozása szerint a /32 méretű kezdeti allokáció egyszerűen /29-re növelhető, ami a táblázatban lévő adatokat 2^{3}-nal szorozva, 8-szorosára növeli a rendelkezésre álló címtartományt. Ezek alapján, ha egy magyarországi LIR úgy dönt, hogy az egyszerűség kedvéért

a byte határokat betartva, mindenkinek /56-os címtartományt jelöl ki, akkor sem reális, hogy kifogyjon a részére allokált IPv6 címtartományból. Ezzel a módszerrel minden ügyfele számára biztosítja azt is, hogy az ügyfelek legalább 256 alhálózatot alakíthassanak ki saját hálózatukon belül.

Természetesen egy hálózat tervezésénél sok szempontot figyelembevétele szükséges, hiszen nagyon fontos az aggregálás lehetősége, a szolgáltató, valamint az ügyfél hálózatának földrajzi elhelyezkedése, az ügyfél hálózatának mérete, stb. Ugyanakkor jól látható, hogy az IPv6 protokoll alkalmazása esetén nem a takarékosság kell, hogy a vezérlőelv legyen. (A technikai szempontok mellett, a marketing szempontok figyelembevétele is szükséges.)

3. Útválasztás

3.1. OSPFv3

Az Open Shortest Path First (OSPF) 2-es verzióját sok helyen alkalmazzák. Előnye, hogy nyílt szabvány, mely hálózati eszközök széles skáláján implementált, így heterogén, több gyártótól származó eszközökből kialakított hálózat kialakítása sem okoz problémát. Az OSFPv2 azonban nem támogatja az IPv6-os protokollt, így szükségessé vált továbbfejlesztése, melynek során először az OSPF for IPv6, majd az IP protokoll mindkét verzióját támogató OSPFv3 került szabványosításra. Elterjedt megoldásként az OSPFv2-t az IPv4 protokoll, míg az OSPFv3-at az IPv6 protokoll útválasztására használják, annak ellenére, hogy az újabb eszközök már támogatják az OSPFv3-as kétprotokollos működését is (AF: Address Families). Az OSPFv2-t jelenleg az RFC 2328, míg az OSPF for IPv6-ot az RFC 5340, végül az OSPFv3 és AF-et az RFC 5838 definiálja. Az OSPFv3 AF nem minden rendszerben implementált. Például a Cisco IOS Release 15.1(3)S és 15.2(1)T előtt is működött az OSPFv3 IPv6, de OSPFv3 IPv4 alkalmazására csak ezektől a felsorolt verzióktól van lehetőség.

3.1.1. OSPFv2 és OSPFv3 összehasonlítása

Mivel az OSPFv3 az OSPFv2 kibővítése az IPv6 támogatás érdekében, így nagyon sok egyezés található a két protokoll működésében, ugyanakkor az IPv6 támogatása érdekében elvégzett változtatások mértéke miatt nem kompatibilis a két verzió, ezért került sor a verziószám növelésére.
Hasonlóságok:

- Mindkettő két szintű hierarchiát alkalmaz, és a kötelező *gerinchálózat terület*re (Backbone area) (0 vagy 0.0.0.0) csatlakozik a többi area.
- Mindkettőben vannak *területhatár útválasztó*k (Area Border Router, ABR) és *AS-határ útválasztó*k (Autonomous System Boundary Router, ASBR).
- Mindkettő a Shortest Path First (SPF) módszert alkalmazza az optimális útvonal kiválasztására, melyet Edsger Dijkstra SPF algoritmusának alkalmazásával számít ki.
- Metrikája mindkettőnek az útvonalak *költség*e (cost).
- Mindkettő 5 csomagtípussal rendelkezik:
    - Hello
    - Database description (DBD)
    - Link-state request (LSR)
    - Link-state update (LSU)
    - Link-state acknowledgment (LSAck)
- Azonos interfész típusokat használnak:
    - Broadcast
    - P2P (point-to-point)
    - P2MP (point-to-multipoint)
    - NBMA (non-broadcast multiple access)
    - Virtual-Links
- Mindkettő azonos időzítéseket használ.

Különbségek:

- Az OSPFv2 csak IPv4, míg az OSPFv3 csak IPv6, vagy mindkét IP protokoll.
- OSPFv3 esetében új LSA típusok.
- Eltérő csomagformátumok.

- OSPFv3 eltérő flooding scope biteket használ.
- OSPFv3 szomszédságok link-lokális IPv6 címekkel.
- OSPFv3 linkenként, nem pedig alhálózatonként működik. (Megjegyzés: általános IPv6 jellemző, hogy nem alhálózat, hanem link terminológiát használ.)
- Több OSPFv3 példány támogatott egy linken.
- OSPFv3 multicast címek: FF02::5 – *az adott linken minden OSPF router* (link-local all OSPF routers), FF02::6 – *az adott linken minden a routing információk cseréjére kijelölt OSPF router* (link-local all OSPF Designated Routers, DRs).
- OSPFv3 protokoll fejlécből eltávolításra kerültek az AuType és Authentication mezők. Így az OSPFv3 alapesetben nem támogatja a szomszédok hitelesítését. Erre a célra csak az IPSec AH és ESP volt használható, majd megjelent az RFC 7166-ban rögzített Authentication Trailer.
- OSPFv3 alatt a Router ID ugyanúgy 4 byte hosszú, és a megfelelő működés érdekében hálózaton belüli egyedi, 0.0.0.0-ától eltérő értékkel kell rendelkezzen.

3.1.2. OSPFv3 működése

Az OSPF *kapcsolatállapot* (link-state) alapú, *autonóm rendszeren belüli útválasztási protokoll* (interior gateway protocol, IGP). Népszerűségének oka a gyors konvergencia, a jó skálázhatóság és a széleskörű támogatás, melynek következtében a többgyártós, heterogén hálózatok kialakítása sem okoz gondot. A következőkben az OSPFv3 működését ismertetjük.

Kapcsolatállapot (link-state) alapú útválasztó protokollok

A link-state routing protokollok számos előnnyel rendelkeznek a *távolságvektor* (distance-vektor) alapú protokollokkal szemben:

- Hierarchikus felépítésükből kifolyólag jobban skálázhatóak, így segítségükkel lényegesen nagyobb hálózatok alakíthatók ki.
- Minden router ismeri a teljes hálózat felépítését (több area alkalmazása esetén, csak a saját area felépítését), beleértve a többi routert, az azok közti összeköttetéseket, valamint azok metrikáit is. Ezeket az információkat felhasználva, a routerek képesek kiválasztani a számukra legmegfelelőbb útvonalat.
- A hálózati topológia változásakor (LSA segítségével) értesítik egymást a routerek, ami gyors konvergenciát eredményez, valamint átviteli kapacitást takarít meg.
- A szomszédos routerek egymással szomszédsági kapcsolatokat alakítanak ki. Ha két szomszéd közt a kommunikáció megszakad, a többi routert az ennek következtében beálló topológiaváltozásról (LSA segítségével) értesíti a problémát érzékelő eszköz.

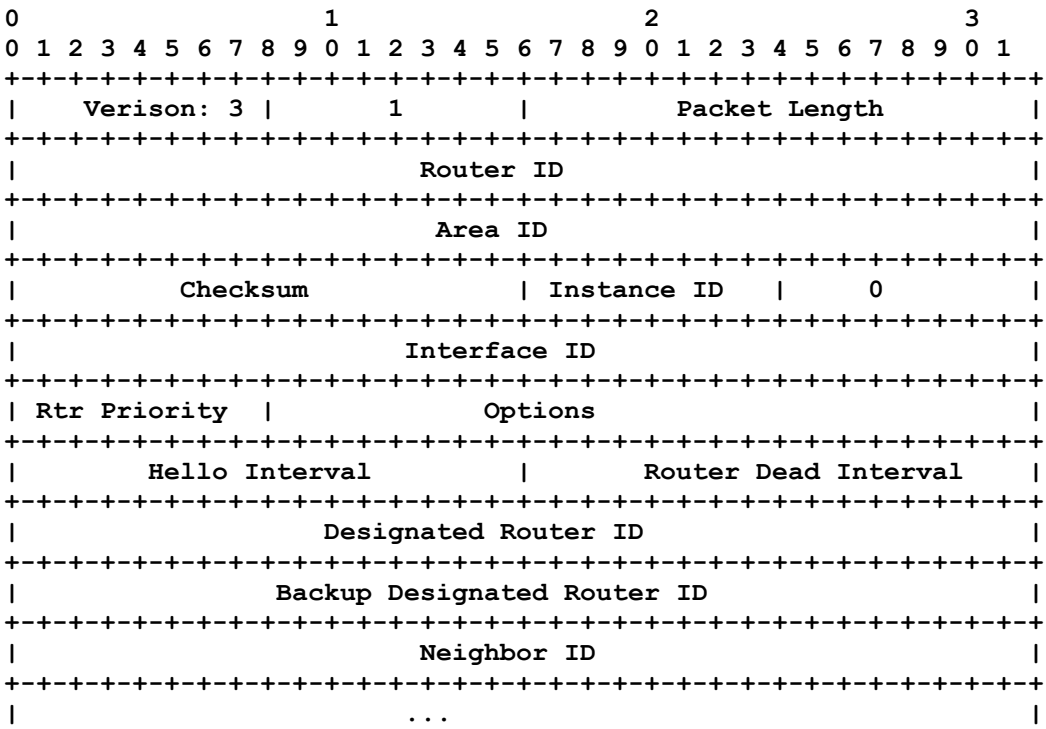
OSPF áttekintése

A routerek Link-State Advertisement (LSA) csomagokat küldenek ki minden állapotváltozás esetén. A többi router irányából fogadott LSA csomagokból kinyert információkat pedig a Link-State Database (LSDB)-ben tárolják le. (Ezt az adatbázist hívhatják még topológia adatbázisnak is, hiszen a hálózat felépítését képezi le a router számára értelmezhető formában.)

Az LSDB alapján az SPF eljárás segítségével, az OSPF esetében alkalmazott *költség* (cost) metrikák figyelembevételével kiválasztják a legolcsóbb *teljes költségű* (cumulated cost) útvonalat. Utolsó lépésben pedig a routing táblába bejegyzik a legolcsóbb útvonalakhoz tartozó *következő router*eket (next-hop).

Szomszédsági kapcsolat kialakítása

Ahogy már korábban is említettük, az OSPF routerek egymással szomszédsági kapcsolatokat alakítanak ki. Erre a célra *Hello* csomagokat használnak, melyeket a FF02::5 multicast IPv6 címre küldenek ki. Az erre a multicast címre érkező csomagokat az összes OSPF router figyeli és feldolgozza. Az OSPF IPv6 Hello packet felépítése az RFC 5340 alapján:



Az OSPFv2 Hello csomaghoz képest a legfontosabb eltérések a következők:

- A verzió mező értéke 3.
- Hiányzik az authentikációs mező, hiszen OSPFv3 esetén erre a célra az IPSec (RFC 4302 , RFC 4303) az RFC 4552-ben megadott módon, vagy az Authentication Trailer (RFC 7166) használható.
- Hiányzik a Network mask, ellenben megjelent az Interface ID.
- Megjelent az Instance ID, melyre elsődlegesen a több AF egyidejű támogatása miatt van szükség (lehetséges értékei: RFC 6969).

A fontosabb mezők:

- **Area ID**: Azt adja meg, hogy az adott interfész melyik area része. Értéke ugyanúgy 32 bit hosszú, mint az OSPFv2 esetében.
- **Router ID**: értéke az OSPFv2-vel megegyezően itt is 32 bit hosszú. Kikötés, hogy értéke 0.0.0.0-tól eltérő, és a hálózatban egyedi legyen.
- **Hello Interval**: A másodpercben megadott időközönként küldik ki (és várják egymástól) a routerek a Hello csomagokat.
- **Router Dead Interval**: Amennyiben ennyi ideig nem kap Hello csomagot egy router a szomszédjától, akkor úgy értékeli, hogy a szomszéd már nem elérhető (az összeköttetés megszakadt). Értéke általában a négyszerese a Hello Interval értékének.

- **Neighbor ID**: Ebből a mezőből annyi található, ahány routerrel a Hello csomagot küldő router a hálózaton, már sikeresen kialakította a kapcsolatot. A mezők értékei, pedig a szomszédos routerek Router ID-ivel egyeznek meg. Ebből a mezőből tudja egy szomszédos router, hogy a Hello csomagot küldő routerhez előzőleg eljutott a saját Hello csomagja..
- **Rtr Priority**: Értéke a *routing információk cseréjére kijelölt útválasztó* (Designater Router, DR) és a *tartalék DR* (Backup Designated Router, BDR) kiválasztásánál jut szerephez. 0 esetén a router biztosan nem lesz DR, vagy BDR.

A szomszédsági kapcsolat kialakításához fontos, hogy a következő mezők értékei megegyezzenek: Area ID, Hello Interval, Router Dead Interval. Az alapértelmezett Hello és Router Dead Interval másodperc értékeket az 5. táblázatban adjuk meg.

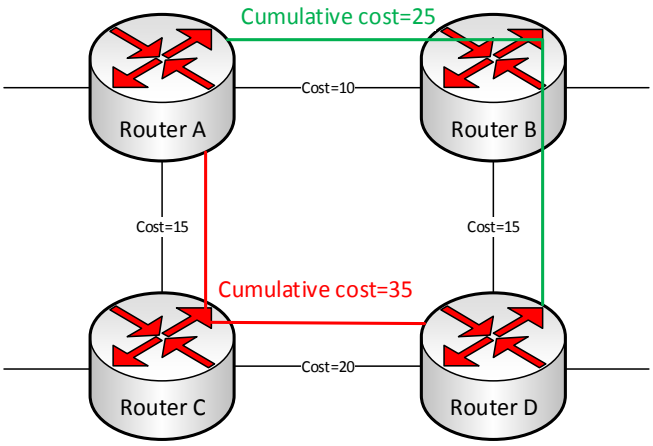
OSPF Hálózat típus	Hello	Router Dead Interval
Broadcast	10	40
Non-broadast	30	120
Point-to-Point	10	40
Point-to-Multipoint	30	120
Point-to-Multipoint Non-broadcast	30	120

5. táblázat: Az OSPF alapértelmezett Hello és Router Dead Interval értékei másodpercben

SPF algoritmus

OSPF routing prtokoll esetében a Shortest Path Sirst (SPF) algoritmus felel a legjobb útvonal kiválasztásáért. A routing protokollok különböző metrikákat alkalmaznak a legjobbnak vélt útvonal kiválasztásához. OSPF esetében ez a metrika az útvonalon lévő összes interfész összesített költsége.

Nézzük meg a működést a 8. ábrán látható példán keresztül.



8. ábra: OSPF útvonalválasztása

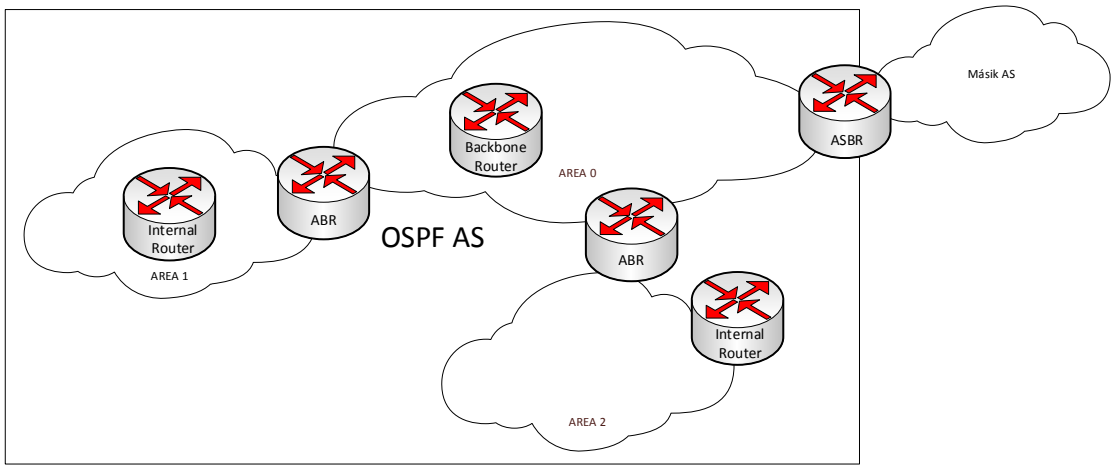
Látszik, hogy a Router A és a Router D között két lehetséges útvonal van. Ha Router A-ból Router B felé indulunk, majd onnan tovább a Router D felé, úgy az útvonal teljes költsége 25 (zölddel jelölt útvonal), míg ha a Router C felé indulunk, majd tovább a Router D-hez, úgy a költség 35 (pirossal jelölt útvonal). Mivel költségről beszélünk, és „az életben is alacsonyabb költségre törekszünk", így a router is az alacsonyabb költséggel járó útvonalat fogja kiválasztani. A költségek

kialakítása az alkalmazott router típusától függhet. Lehet beállítható manuálisan, de akár az adott interfész sebességének és egy referenciasebességnek a felhasználásával is automatikusan előállíthatja a router.

Az SPF algoritmus Edsger W. Dijkstra, dán tudós által 1956-ban megalkotott, és róla elnevezett Dijkstra algoritmus segítségével az LSDB adatait felhasználva felépít egy olyan gráfot, melynek kiindulási pontjában a router van, és minden egyes alhálózat felé a legalacsonyabb költségű útvonalon lehet eljutni. Tehát egy area esetében minden router LSDB-je megegyezik, azonban a gráfok eltérnek, hiszen a gráfok kiindulási pontjában minden esetben más router található. Az SPF algoritmus viszonylag nagy erőforrásigénye miatt régebben sok kisméretű area kialakítása volt célszerű, mára azonban a Routerek fejlődésével már lényegesen nagyobb area is gond nélkül kialakítható. (Fontos ugyanakkor, hogy a nagyobb Area, nagyobb hálózati forgalmat is generál.)

OSPF Area

Az OSPF segítségével nagyméretű, sok alhálózatból álló hálózatok is kialakíthatóak. Nagy hálózatok esetében azonban érdemes a hálózatot több részre, úgynevezett areára osztani. Fontos, hogy Backbone Area (0.0.0.0, vagy 0-ával jelölt) kialakítása minden esetben szükséges, és minden area kapcsolattal kell, hogy rendelkezzen a Backbone Area irányában. Tehát a Backbone Area biztosítja a tranzitforgalom továbbítását a többi Area között. Ezért sokszor nevezik tranzit areának is.



9. ábra: OSPF AS felépítése

A 9. ábrán egy három area segítségével felépített OSPF autonóm rendszer (Autonomous System, AS) látható. (AS-nek nevezzük az egy adminisztratív és/vagy műszaki fennhatóság alá tartozó, közös routing politikát alkalmazó hálózatot. Általában egy cég hálózata.) Az ábrán szereplő routerek:

- **Backbone Router**: Olyan router, melynek minden interfésze az Area 0 része.
- **Internal Router**: Olyan router, melynek minden interfésze azonos Area része, de az nem a Backbone.
- **Area Border Router (ABR)**: Interfészei különböző Areák részei. Mivel minden Area kapcsolódik a Backbonehoz, így ezen routerek egyik interfésze a Backbone Area része.
- **Autonomous System Border Router (ASBR)**: Más AS-ekhez biztosítja a kapcsolatot. (Az OSPF AS értelmezése eltér a BGP routing protokoll esetén alkalmazottól. Az OSPF protokollt általában nem alkalmazzuk EGP módon, valós AS-ek közti forgalom

kicserélésére, inkább egy AS-en belül kerülhetnek kialakításra OSPF, vagy más IGP „szigetek”.)

Designated és Backup Designated Router

Többszörös hozzáférésű hálózatokban (multiple access networks) fontos az adott linken kialakuló szomszédsági viszonyok darabszámának alakulása. (Például egy switchre több router csatlakozik.) A kidolgozott megoldás célja, hogy a létrejövő szomszédsági viszonyok darabszámát négyzetesről lineárisra csökkentse. Az OSPF ezért alkalmazza a *routing információk cseréjére kijelölt* (tartalék) útválasztókat ((Backup) Designated Router, DR és BDR). Ezeken a hálózatokon a nem DR és nem BDR routerek csak a DR és BDR routerekkel cserélnek LSA csomagokat, így nagymértékben csökkentik a hálózati forgalmat, valamint tehermentesítik a többi routert is. A leírtakból is egyértelmű, hogy pont-pont OSPF interfész esetén nincs DR és BDR szerepkör, hiszen csak két router csatlakozik egy összeköttetésre, így alkalmazása nem csökkentené a szomszédsági viszonyok számát. A DR és BDR kiválasztást a Hello csomag Rtr Priority mező értékével tudjuk befolyásolni. Általában a nagyobb teljesítményű, nagyobb memóriával rendelkező routereknek érdemes nagyobb prioritást adni, vagy a 0 prioritás segítségével a nagyon kis kapacitású routereken letiltani a DR/BDR szerepkört. Azonos prioritás értékek esetén a nagyobb ID-vel rendelkező router veszi fel a (B)DR szerepkört.

3.2. BGP-MP

A Border Gateway Protocol (BGP) egy AS-ek közti routing protokoll (interdomain routing protocol, IDRP, más kifejezéssel: interautonomous system, inter-AS). Jelenleg a 4-es verziója van használatban (Border Gateway Protocol 4, BGP-4), melyet az RFC 4271 definiál. A BGP-4 IPv6-os kiterjszetésével (Multiprotocol Extensions for BGP-4) az RFC 4760 foglalkozik. Fontos megjegyezni, hogy a két említetten kívül még számtalan RFC kapcsolódik a BGP-4 protokollhoz. A BGP egy nagyon megbízható, sokoldalúan konfigurálható, rendkívűl jól skálázható, az internetszolgáltatók, valamint a nagy hálózatokat üzemeltető szervezetek körében széleskörben alkalmazottt routing protokoll. Mind AS-ek közt (exterior gateway protocol, EGP[7]), mind pedig AS-en belül (interior gateway protocol, IGP) alkalmazható (és alkalmazott), ugyanakkor AS-en belüli alkalmazása nehézkes, ezért általában más IGP protokollal (pl. OSPF) közösen használják. Könyvünkben csak a BGP IPv6-os alkalmazásához és annak megértéséhz feltétlenül szükséges alapokat mutatjuk be, bővebb információkért érdemes a kapcsolódó RFC-ket tanulmányozni.

Mivel a BGP elsődlegesen EGP, és a másik AS-ből érkező információk nem feltétlenül megbízhatóak, de akár tévedésből is előfordulhatnak hibás információk, a BGP nagyon sok szűrési lehetőséget támogat. A BGP sokrétű szűrése teszi lehetővé az AS tulajdonosa számára a részére megfelelő routing politika (policy) technikai megvalósítását. A szűrések mellett képes a BGP peer hitelesítésére is. Erre a célra általában TCP MD5-öt (RFC 2385) alkalmaznak, de akár IPsec segítségével is biztosítható a BGP routerek közti kommunikáció biztonsága.

3.2.1. BGP alapok

A BGP fejlesztése során nagy hangsúlyt fektettek a skálázhatóságra, megbízhatóságra, és a biztonságra. A könyv írásakor körülbelül 585.000 prefix volt megtalálható a teljes IPv4 routing táblában, melyek információit az internetszolgátatók egymással BGP protokoll segítségével cserélik ki. Ez a mennyiség ugyanakkor a BGP-nek nem okoz gondot, a legnagyobb problémát a routerek

[7] Az AS-ek közti forgalom kicserélésére az 1980-as évek elejétől az Exterior Gateway Protocol 3-as verzióját (EGP3) használták (RFC 827 és RFC 904). Innen is ered a manapság használt BGPv4 meghatározása EGP protokollként. Fontos azonban, hogy ne keverjük össze a régi, elavult protokollt és az AS-ek közti működésre utaló elnevezést.

memóriamérete szokott okozni, mert az ilyen mennyiségű adat tárolása csak kellően nagy memóriával rendelkező routerekben lehetséges.

A BGP egy *kibővített távolságvektor* (enhanced distance-vector) alapú routing protokoll [4], amely csak triggered update-eket alkalmaz és nagyon sokféle metrika (ezek a metrikák a BGP attribútumok, melyekről később lesz szó) alkalmazását támogatja. A triggered update-eket nem azonnal továbbítja, hanem kötegekbe összefogva. AS-en kívüli szomszédok[8] felé (alapértelmezetten) 30 másodpercenként, míg AS-en belüli irányban (alapértelmezetten) 5 másodpercenként. (A kötegelt továbbítás nagyobb skálázhatóságot, ugyanakkor lassabb konvergenciát is eredményez.) A kommunikációhoz a megbízható adatávitelt biztosító TCP-t alkalmazza, valamint (alapértelmezetten 60 másodpercenként) keep-alive csomagok segítségével győződik meg egy másik BGP peer elérhetőségéről. (Hiszen egy BGP szomszédsági viszony nem feltétlenül jelent link szomszédságot is.)

BGP attribútumok

Az OSPF ismertetése során már megismerkedtünk a metrika fogalmával, és azt is tudjuk, hogy az egyes routing protokollok a metrikák alapján döntik el azt, hogy egy-egy csomagot milyen irányban továbbítanak. A BGP esetében a metrikák a *BGP útvonal attribútumok* (BGP path attributes). Ezek az attribútumok lehetnek kötelezően (well-known) és opcionálisan (optional) támogatottak. A well-known attribútumok a következők:

- Mandatory (kötelezően továbbítandók)
    - Origin: a hálózat irányába mutató route forrása (IGP, EGP, Incomplete)
    - AS-path: a hálózat mely AS-eken keresztül, és milyen sorrendben érhető el, használható például optimális útvonal kiválasztására, és segíti a hurkok elkerülését is
    - Next-hop: a következő router IP címe a hálózat irányába
- Discretionary (kötelezően támogatandóak, de nem kötelező továbbítani):
    - Local preference: csak azonos AS-be tartozó peerek részére (tehát IBGP esetben) kerül továbbításra, alapértéke 100, a routerek a nagyobb értékekkel rendelkező *útvonalakat* (route) preferálják azonos hálózatok irányába
    - Atomic aggregate: az útvonal (route) aggregálással (több kisebb hálózat összevonásával) került előállításra

Az opcionális attribútumok két csoportra oszthatók:

- Tranzitív: továbbítása megtörténik, valamint a felismerés megjelölésre kerül
- Nemtranzitív (Non-transitive): csak felismerés esetén kerül továbbításra

A legfontosabb opcionális attribútum a community, mely tranzitívan viselkedik. A community egy olyan numerikus érték, mely a route-hoz kapcsolható, annak továbbításakor. Segítségével „megjelölhetőek" az egyes route-ok, és a jelölések később felhasználhatók a használni kívánt route-ok kiválasztására.

3.2.2. BGP kapcsolat kialakítása

A BGP esetében minden egyes szomszéd (neighbor) manuális beállítása szükséges. Míg az OSPF esetében a szomszédok képesek egymást automatikusan felderíteni, egy EGP esetében ennek a módszernek az alkalmazása nem célszerű (pl. biztonsági okokból), így a routereken az egyes peerek paramétereinek manuális beállítása szükséges. Ilyen paraméterek lehetnek például:

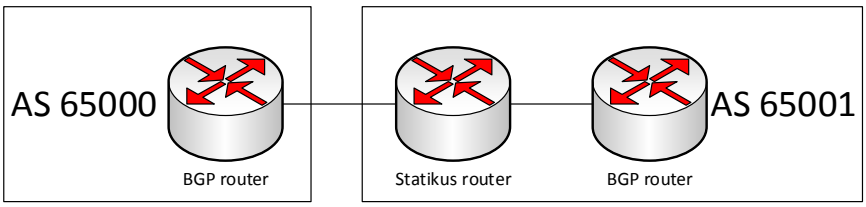
- a neighbor IPv4 vagy IPv6 címe (kötelező)
- a neighbor AS-e (kötelező)
- authentikáció esetén az alkalmazott jelszó (MD5 shared secret)

[8] Ahol az OSPF-től eltérően nem feltétlenül közvetlen szomszédságról van szó.

- a kommunikáció során felhasznált forrás IP-cím
- a szomszédtól maximálisan elfogadott prefixek-száma
- kimenő és bejövő szűrések

Egy BGP router a szomszédaival a 179-es TCP porton kommunikál. Érdekes hálózati konfigurációk is kialakíthatóak, ha figyelembe vesszük azt is, hogy két szomszéd BGP routernek nem kell feltétlenül ugyanazon alhálózatban lennie. Erre mutat példát a 10. ábra. (Megjegyezzük, hogy az RFC 5398 rögzít dokumentációs célokra AS azonosítókat, a jobb áttekinthetőség érdekében mégsem azokat, hanem az RFC 6996 szerinti privát AS számokat használunk a példákban.)

Ezen topológia alkalmazásakor figyeljünk arra, hogy a statilkus routeren a megfelelő routing tábla bejegyzéseket létrehozzuk!

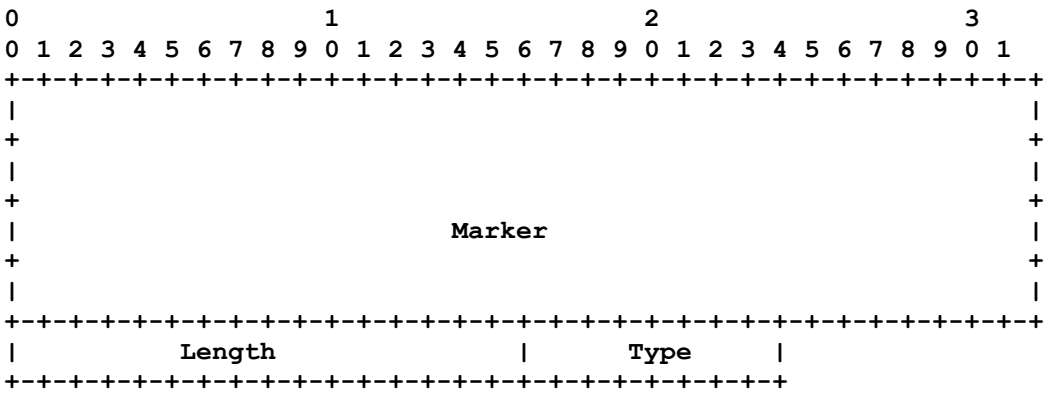


10. ábra: Multihop BGP kialakítása

3.2.3. BGP kommunikáció

BGP fejléc formátum

A BGP kommunikáció során az üzenetek a következő (RFC 4271 által meghatározott), fix 19 byte hosszúságú fejléc alkalmazásával kerülnek továbbításra:

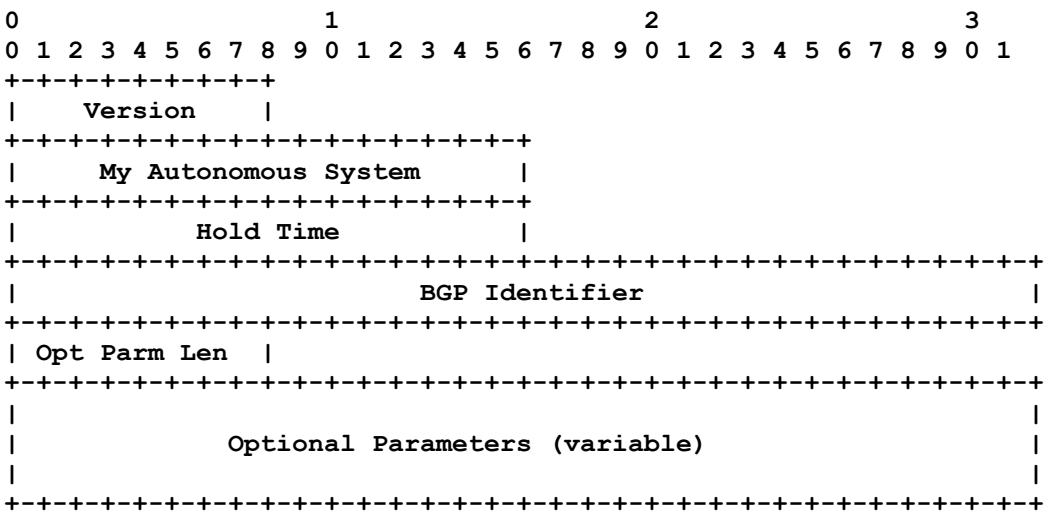


Az egyes mezők jelentése a következő:

- Marker: kompatibilitási okokból került alkalmazásra, értéke csupa 1-es.
- Length: az üzenet teljes hosszát adja meg, beleszámolva a headert is.
- Type: Értéke az 5 féle üzenettípust azonosítja:
  - 1 - OPEN
  - 2 - UPDATE
  - 3 - NOTIFICATION
  - 4 - KEEPALIVE
  - 5 - ROUTE-REFRESH

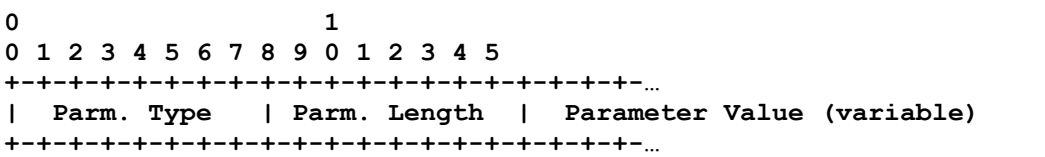
BGP open üzenet

A BGP kapcsolat kialakítása során elsőként a BGP open üzenet kerül továbbításra. Az open üzenet RFC 4271 alapján a következő módon épül fel:

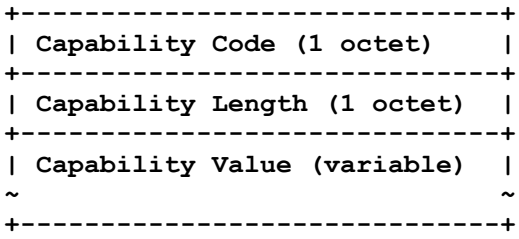


Az egyes mezők tartalma és felhasználása a következő:

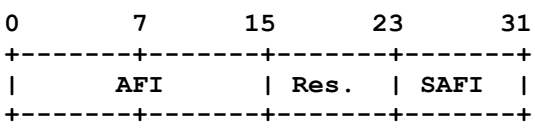
- Version: Megadja a router által használt BGP verziószámát. Értéke 4.
- My Autonomous System: Megmutatja, hogy a router mely AS-ben található. Fontos megjegyezni, hogy az RFC 4893 (már elavult), majd az RFC 6793 (érvényes) kibővítette a BGP alkalmazása esetén használható AS-ek számát 4 byte hosszra (ami a kompatibilitás megtartása miatt nem volt triviális feladat).
- Hold Time: A küldő router által javasolt megőrzési időtartam, mely azt mondja meg, hogy a címzett router mennyi ideig kezelje érvényesként a küldő routert. 0 értéke esetén nem kerülnek kiküldésre a keepalive üzenetek.
- BGP azonosító: Eredetileg a router egyik IPv4-es címe, majd az RFC 6286 megjelenésétől, bármely nem nulla értékű, négy byte hosszú, előjel nélküli egész szám, mely az adott AS-ben előforduló BGP routerek közt egyedi.
- Optional Parameters Length: Az opcionális paraméterek teljes hossza.
- Optional Parameters: Az opcionális paraméterek a következő (RFC 4271 szerinti) formátumban:



Számunkra az opcionális paraméter mezők a különösen érdekesek, hiszen itt helyezhető el annak az információnak a hirdetése is, hogy a router képes az IPv6 támogatására. A BGP capability hirdetésének módját az RFC 5492 írja le. Amennyiben az Optional parameter type mező értéke 2, akkor az „Capabilities Optional Parameter”-t azonosít a következő (RFC 5492 szerinti) formában (ismétlődés esetén mindhárom mező ismétlődik):



A BGP Multiprotocol extension hirdetése esetén a Capability Code mező értéke 1, míg a Capability Length mező értéke 4. A Capability value felépítése (RFC 4760 alapján) a következő:



Az egyes mezők a következő jelentéssel bírnak:

- AFI - Address Family Identifier: 1-es értéke IPv4-et, míg 2-es értéke IPv6 protokollt jelez (http://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml).
- Res. – Reserved: értékét a küldőnek 0-ára kell állítania, a fogadónak pedig figyelmen kívül kell hagyina a mezőt.
- SAFI - Subsequent Address Family Identifier: a használt értékek megtalálhatók az IANA oldalán (http://www.iana.org/assignments/safi-namespace/safi-namespace.xhtml).

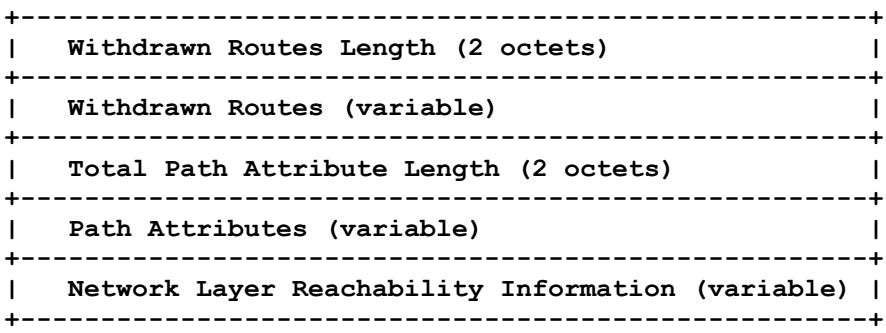
BGP keepalive üzenet

A BGP open üzenet kiküldése után a küldő router válaszként szintén egy open üzenetet vár. Ha ez az üzenet megérkezik, és a benne lévő paraméterek megfelelőek, létrejöhet a BGP kapcsolat a két szomszéd router közt. Ahhoz, hogy a kapcsolat ténylegesen létrejöjjön a routereknek még keepalive üzenetekkel is nyugtázniuk kell azt. Annak érdekében, hogy a kapcsolat ellenőrzése folyamatosan megtörténjen rendszeres időközönként BGP keepalive üzenetek kerülnek továbbításra. Abban az esetben, ha egy BGP szomszéddal a hold-time időtartamán belül nem sikerül keepalive üzenetet cserélni, akkor a router a kapcsolatot megszűntként kezeli és megteszi a szükséges további lépéseket. (Pl: Routing tábla módosítása, többi neighbor értesítése, stb.)

A keepalive üzeneteket hold-time/3 időtartamonként ajánlott kiküldeni. A keepalive üzenet csak a 19 byte hosszú BGP headerből áll.

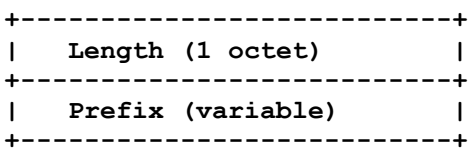
BGP update üzenet

Feladata az útválasztási információk továbbítása az egyes BGP routerek közt. Egy update üzenet továbbíthat olyan lehetséges útvonalakat, melyek ugyanolyan útvonal attribútumokon osztoznak, vagy visszavonhat megszűnt útvonalakat. Egy update üzenet minden esetben tartalmazza a BGP fejlécet, valamint a következő üzenetfajták egy részét (RFC 4271):

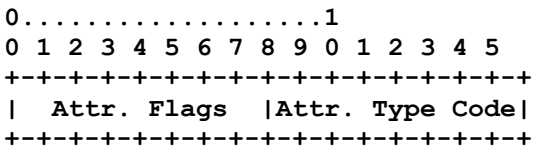


Az egyes mezők jelentése a következő:

- Withdrawn Routes Length: az útvonal visszavonási információk hosszát adja meg byte-ban. 0 esetén nincs visszavonás;
- Withdrawn Routes: maguk a visszavont útvonalak a következő formátumban:



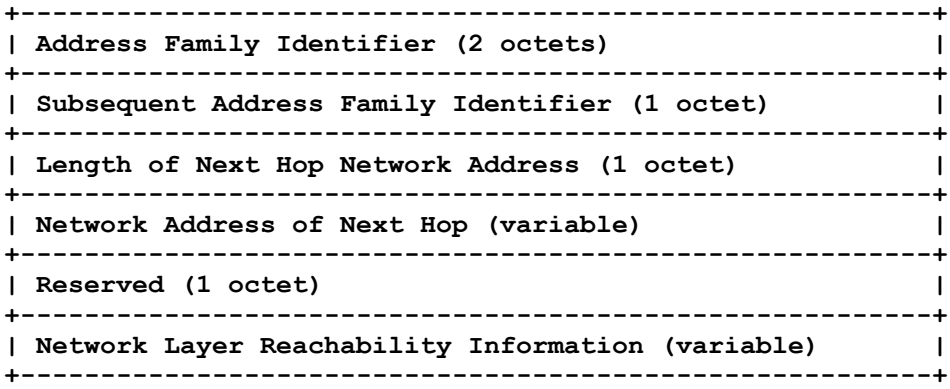
  - Length: az IPv4 cím prefix bitjeinek számát adja meg,
  - Prefix: az IPv4 cím prefixe, mely a hálózatot azonosítja (byte határra kiegészítve),
- Total Path Attribute Length: az útvonal attribútumok hosszát adja meg;
- Path Attributes: maguk az útvonal attribútumok a következő formátumban (visszavonás esetén nem része az üzenetnek, nem tárgyaljuk részletesen):



- Network Layer Reachability Information (NLRI): az IPv4 cím prefixek listája a Withdrawn Routes mezőnél megadott formátumban.

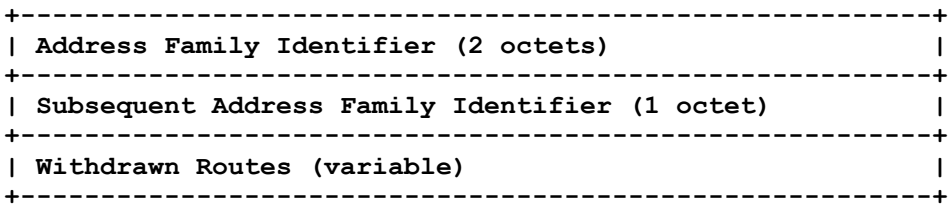
A fenti információk alapján az RFC 4271-ben definált BGP-v4 nem képes IPv6 routing információk továbbításra, ezért az RFC 4760 kiegészítette azt az IPv6 támogatásához szükséges két további opcionális és nem tranzitív attribútummal (így ha egy MP-BGP-t nem támogató router kap ilyen attribútumot, akkor azzal nem foglalkozik, és nem is továbbítja):

- Multiprotocol Reachable Network Layer Reachability Information (MP_REACH_NLRI): az elérhető hálózatok, és a next-hop információkat továbbítja a következő formátumban:



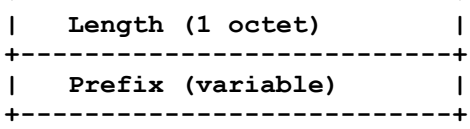
- Address Family Identifier és Subsequent Address Family Identifier: korábban már ismertettük,
- Length of Next Hop Network Address: a next hop címének hossza byte-ban,
- Network Address of Next Hop: a next hop címe,
- Reserved: a küldő kötelezően 0 értékre állítja, míg a címzett figyelmen kívül hagyja,
- Network Layer Reachability Information (NLRI): a lehetséges útvonalak listája;

- Multiprotocol Unreachable Network Layer Reachability Information MP_UNREACH_NLRI: az elérhetetlenné vált hálózatok listáját továbbítja a következő formátumban:



- Address Family Identifier és Subsequent Address Family Identifier: korábban már ismertettük,
- Withdrawn Routes: a visszavont útvonalinformációk listája a Network Layer Reachability Information (NLRI) mezővel megegyező formátumban.

A Network Layer Reachability Information (NLRI) listában egy, vagy több prefix kerül továbbításra a következő formátumban (RFC 4760):

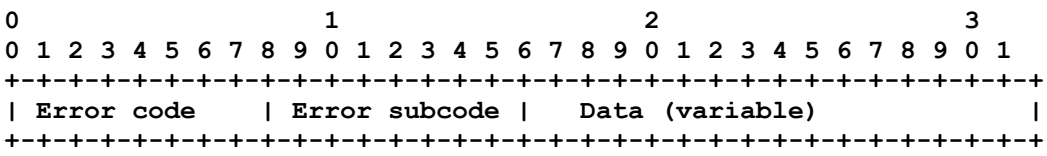


A két mező jelentése a következő:

- Length: a cím prefix bitjeinek számát adja meg;
- Prefix: a cím prefixe, mely a hálózatot azonosítja (byte határra kiegészítve).

BGP notification üzenet

Hiba észlelése esetén a router BGP notification üzenetet kiküldésével értesíti a szükséges router(eke)t, majd lezárja a kapcsolatot. RFC 4271 szerinti formátuma a következő:



Az egyes mezők jelentése a következő:

- Error code: az értesítés típusát határozza meg
- Error subcode: részletesebb információkat nyújt a hiba természetéről (bővebben: RFC 4271)
- Data: információk a hiba okának feltárásához. Pontos tartalma az Error code és Error subcode mező értékétől függ. Bővebben: RFC 4271

3.3. PIM-SM

3.3.1. Bevezető

Számos IP multicast routing protokoll létezik. Ilyenek például az OSPF „multicast párja" az MOSPF (Multicast Open Shortest Path First, RFC 1581), a DVMRP (Distance Vector Multicast Routing Protocol, RFC 1075), és a Core-Based Trees (RFC 2189) is. A PIM (Protocol Independent Multicast) pedig egy protokollcsalád. A PIM multicast routing protokoll valamely unicast routing protokolltól (pl. OSPF) szerzett routing információ alapján épít multicast fákat, ezért is hívják *protokoll független*nek (protocol independent). Négy változata van:

1. **PIM-DM** (PIM – Dense Mode, RFC 3973) Úgy épít multicast fákat, hogy az egész hálózatot elárasztja multicast forgalommal, és azután levágja azokat az ágakat, ahol nincsenek multicast vevők. Amint a neve is mutatja, használata akkor előnyös, ha a hálózatban „sűrűn" helyezkednek el a multicast forgalomra igényt tartó állomások.
2. **PIM-SM** (PIM – Sparse Mode, RFC 4601) Nem tételez fel mindenhol csoporttagokat, csak arra küld multicast forgalmat, ahonnan arra igényt jelentettek be. A forgalom továbbítására az ún. *randevú pont*ban (Rendezvous Point) gyökerező, több forrás forgalma által megosztottan használt fákat (shared trees) épít. Opcinálisan használhat forrásonként legrövidebb utat megvalósító fákat is. Működését részletesen bemutatjuk.
3. **PIM-SSM** (PIM – Source-Specific Multicast, RFC 4607) Egyetlen forrásban gyökerező multicast fákat épít fel, így nincs szüksége RP-re. A PIM-SM-nél elvileg szűkebb körben, tipikusan valamilyen tartalom broadcast jellegű továbbítására használható. IPTV szolgáltatásra viszont ez is elég, ezért a jövőben várhatóan terjedni fog a használata.
4. **BIDIR-PIM** (Bidirectional PIM, RFC 5015) A PIM-SM olyan változata, amely a forrásokat és a vevőket összekötő kétirányú megosztott fákat épít. A PIM-SM-mel ellentétben sohasem épít forrásonként legrövidebb utat megvalósító fákat.

Ezek közül a PIM-SM protokollt IPTV rendszerekben elterjedten használják, ezért azt fogjuk részletesen bemutatni.

3.3.2. A PIM-SM működése

A protokoll működését [5] cikkünk alapján mutatjuk be. Amint említettük, a PIM-SM nem rendelkezik saját topológiafelderítő algoritmussal, hanem az adott autonóm rendszerben használt unicast routing protokoll routing adatbázisát (RIB: Routing Information Base) használva építi fel a saját multicast routing adatbázisát (MRIB: Multicast Routing Information Base). Míg a unicast routing adatbázisok egy adott hálózat felé a *következő útválasztó*t (next hop router) adják meg, addig a multicast routing adatbázisban az adattovábbítással ellentétes irányú (reverse-path) információt tárolják (később látni fogjuk, hogy miért: erre haladnak a PIM Join/Prune üzenenetek).

A PIM-SM protokoll működésének három fázisa van, a továbbiakban röviden ismertetjük, hogy mi történik ezekben a fázisokban, de előbb még megismerünk egy fontos fogalmat.

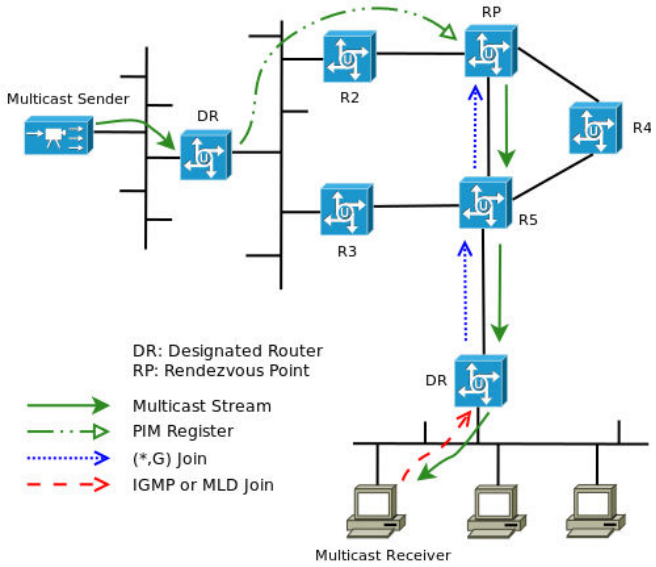
A randevú pont

Mivel a PIM-SM a *bármely forrású multicast* (ASM: Any-Source Multicast) routing protokollok közé tartozik, a vevőknek valahogy meg kell találniuk a multicast forrást vagy forrásokat. Ezt a célt szolgálja a *találkozási pont* vagy *randevú pont* (RP: Rendezvous Point). Az RP-t statikusan beállíthatja az adott AS adminisztrátora vagy egy megfelelő algoritmussal megválaszhatják az adminiszrátor által *RP jelölt*nek (Candidate RP) beállított routerek közül. Egy AS-ben vagy multicast tartományban multicast csoportonként logikailag csak egy RP lehet. Ennek nem mond ellent, hogy létezik az ún. *Anycast RP* (RFC 4610) megoldás, amikor több RP példány van egy multicast tartományban, és ezek ugyanazt az IP-címet használják (anycast címzéssel).

1. fázis: RP-tree

Az első fázisban a *randevú pontban gyökerező fa* (Rendezvous Point Tree) épül fel a következő módon. A vevők az *MLD* (Multicast Listener Discovery, RFC 3810) protokoll használatával *Multicast Listener Report* üzenetekkel jelzik az igényüket valamely multicast csoport forgalmára. A vevő *kijelölt route*r (DR: Designated Router, ezt korábban már megválasztották a helyi routerek közül) ennek az üzenetnek a hatására *PIM Join* üzenetet küld az igényelt multicast csoport RP-jének. Ez a PIM Join üzenet áthalad a hálózatban az RP felé vezető út routerein, amelyek elkészítik a megfelelő MRIB bejegyzéseket, és így felépül az RP-fa. Egy PIM Join üzenet jelölése (S, G), ahol az első elem a forrás IP-címe, a második elem pedig a csoport IP címe (ennek a prefixe természesen FF00::/8). A (*, G) jelölés pedig azt jelenti, hogy a vevő a G multicast csoport minden forrására igényt tart. Fontos, hogy a forrás IP-címe ismeretének hiányában a DR először mindig (*, G) Join üzenetet küld! (Majd a 3. fázisban lesz lehetősége a forrás alapján való finomításra.) A PIM Join üzeneteknek nem mindig kell az RP-ig eljutniuk, hanem elég, ha elérnek egy olyan pontot, ahol az RP-fa már kiépült. Az RP-fát azért hívják *megosztott fá*nak (shared tree), mivel minden forrás multicast forgalma ugyanazt a fát használja. A PIM Join üzeneteket a DR periodikusan küldi, amíg létezik az adott csoportnak legalább egy tagja (van olyan vevő, aki az adott multicast csoport forgalmára igényt tart). Amikor az RP-fának egy *levél* (leaf) – azaz a fában csak egyetlen kapcsolattal rendelkező – hálózatában az utolsó tag is elhagyja a G csoportot, akkor a DR egy (*, G) jelölésű *PIM Prune* üzentet küld az RP felé, és a fát vissszametszik addig a pontig, ahol még vannak aktív vevők.

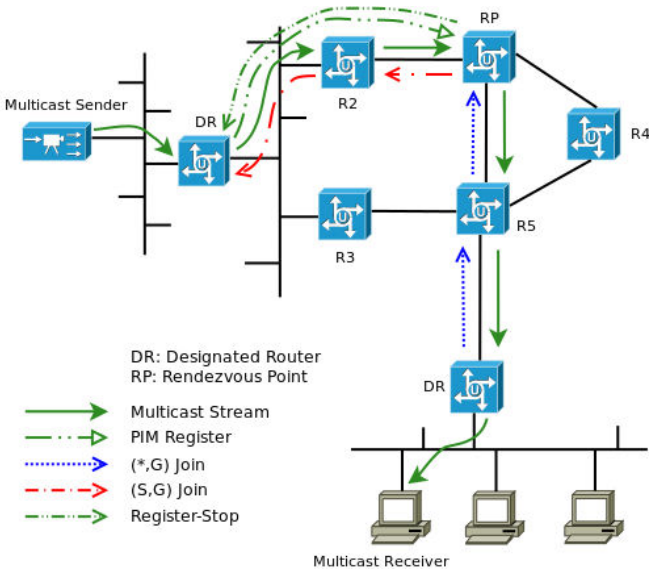
Amikor egy S forrás adni kezd egy G multicast csoport számára, akkor a forrás DR-e a multicast adatcsomagokat *regisztrációs üzenetek*nek (Register messages) nevezett unicast csomagokba ágyazza be, amelyeket az RP-nek címez. (Az RP router ezekből az üzenetekből szerez tudomást arról, hogy az S forrás szeretne multicast forgalmat küldeni.) Az RP kibontja a regisztrációs üzeneteket, és az RP-fa mentén továbbítja a megfelelő multicast csoport számára (ha létezik legalább egy tagja). A folyamat működését a 11. ábra mutatja be. A multicast továbbítás ekkor már funkcionálisan működik, a további két fázis csak a hatékonyságot szolgálja.



11. ábra: A PIM-SM működése – 1. fázis [5]

2. fázis: register-stop

Az RP egy (S, G) Join üzenetet küld az S forrásnak. Amint ez az üzenet a S forrás felé halad, a routerek bejegyzik az (S, G) párt a multicast routing adatbázisukba (MRIB), ha eddig még nem szerepelt benne. Amikor a Join üzenet a forrás hálózatába vagy egy olyan routerhez ér, amelynél az MRIB-ben már szerepelt az (S, G) bejegyzés, akkor a multicast tartalom az S forrástól RP félé multicast módon is áramlani fog. (Így az RP-hez az adás duplán érkezik meg.) Ezzel már felépült egy *forrás specifikus muticast fa* az S forrástól az RP-ig. Ezután az RP egy *regisztráció vége* (Register-Stop) üzenetet küld, jelezve a forrás DR-ének, hogy már nem kell küldenie a regisztrációs üzeneteket

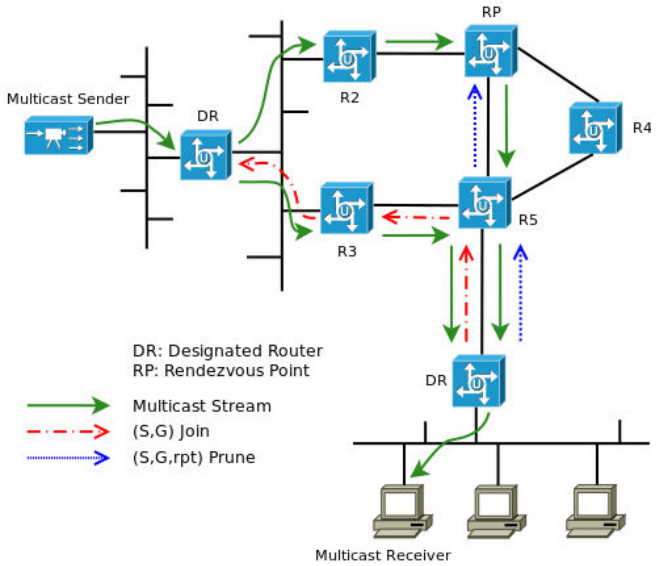


12. ábra: A PIM-SM működése – 2. fázis [5]

(amelyek a multicast forgalmat unicast csomagokba ágyazzák be). A 2. fázis működését a 12. ábra mutatja be.

3. fázis: shortest-path tree

Lehetséges, hogy a multicast csomagok útja a forrástól az RP-n át a vevőkig nem optimális. Ennek a kiküszöbölése érdekében a vevő DR-e kezdeményeheti (MAY, fontos, hogy ez opcionális) egy *forrás specifikus legrövidebb utat megadó fa* (source specific shortest-path tree) felépítését a forrás felé (amely esetleg nem tartalmazza az RP-t). A vevő DR-e ezt úgy oldja meg, hogy egy (S, G) Join üzentet küld az S-nek. Amikor ez az üzenet megérkezik az S hálózatába vagy egy olyan routerhez, amelynél az MRIB-ben már szerepel az (S, G) bejegyzés, akkor a multicast forgalom az S forrástól a vevő felé az új forrás specifikus SPT-n át is áramlani fog. Ekkor a vevő DR-e minden adatcsomagot kétszer kap meg, ezért az RP-fán keresztül érkező, az S forrástól a G csoportnak szóló multicast csomagokat eldobja. Ennek az állapotnak a kiküszöbölésére a vevő DR-e egy (S, G) Prune üzenetet küld az RP-nek. (Ezt az üzenetet úgy is hívják, hogy (S, G, rpt) Prune.) Ez a Prune üzenet lemetszi a szükségtelen faágakat és a multicast adás többé nem fog az RP-fán keresztül érkezni a vevőhöz. Megjegyzés: mindezt (beleértve a csomageldobást és a Prune küldését) a vevő DR-e helyett egy tőle a forrás felé levő (upstream) router is elvégezheti, ha a duplán érkezés jelenségét észleli. A 3. fázis működését a13. ábra mutatja be.



13. ábra: A PIM-SM működése – 3. fázis [5]

3.3.3. A PIM-SM hibatűrése

A PIM-SM hibatűrésének fontos eleme, hogy az RP-t nem szükséges kézzel beállítani, hanem az RP automatikusan megválasztható az *RP-jelölt*ként (C-RP: Candidate RP) beállított routerek közül.

A választás az RFC 5059-ben leírt bootstrap mechanizmust használja. Ehhez még egy előzetes lépésre van szükség, melynek során először egy úgynevezett *BSR router*t választanak meg azok közül a PIM-SM routerek közül, amelyeket *BSR jelölt*nek (C-BSR: Candidate BSR) állítottak be. A BSR megválasztásához a C-BSR routerek elárasztják a multicast tartományt (multicast domain) a *bootstrap üzenet*eikkel (BSM: Bootstrap message), melyben megadják a saját prioritásukat. Amelyik a C-BSR router a sajátjánál nagyobb prioritású C-BSR router üzenetét hallja, az egy ideig nem küld további saját BSM-et. Így végül a legnagyobb prioritású nyer. A BSR megválasztása közben az összes router

(beleértve a C-RP routereket is) megtanulja a BSR IP-címét. Eztuán a C-RP routerek periodikusan küldik az *RP jelölt hirdetés* (C-RP-Adv: Candidate-RP-Advertisement) üzeneteiket a BSR-nek. (A C-RP-Adv üzenetek küldésének periódusidejét *C_RP_Adv_Period* névvel illetik, melynek alapértelmezett értéke 60 másodperc.) A BSR begyűjti a C-RP-Adv üzeneteket, és az RP jelöltekről listát készít (RP list), majd a listát egy bootstrap üzenetbe (BSM, ugyanolyan típusú üzenet, mint amit a BSR megválasztásához is használtak) ágyazva periodikusan elküldi az összes PIM-SM routernek (periódusidő: *BS_Period*, alapértelmezett érték: 60 másodperc). Minden PIM-SM router (beleértve a BSR-t és a C-RP-ket is) saját maga képes C-RP-k prioritása alapján a győztes RP meghatározására. (Ezt mindegyik PIM-SM router maga végzi el, és a végső eredmény meghatározásakor a BSM-ben kapott információn túl figyelembe veszi a benne statikusan beállított információt is.)

Megjegyzés: multicast csoportonként különböző RP-k lehetnek, az RP-ket csoportcímmel együtt hirdetik. Sőt egy RP több csoporthoz is tartozhat, ennek a hatékony megadására a csoportcímen túl *csoportmaszk*ot (group mask) is továbbítanak.

Ha az aktuális RP nem küld C-RP-Adv üzenetet a BSR-nek *RP Holdtime* (a C-RP-Adv üzenetben megadott érték: a hirdetés ennyi ideig érvényes) időn belül, akkor a BSR úgy dönt, hogy az adott RP nem működik és új RP listát kezd el hirdetni a nem működőnek ítélt RP kihagyásával.

A C-RP routerek leállásukat a BSR számára 0 értékű RP Holdtime megadásával tudják jelezni. Ilyenkor az BSR azonnal új RP listát hirdet az adott C-RP kihagyásával. A C-RP routereknek a C-RP-Adv üzenetben legalább 2,5*max(BS_Period, C_RP_Adv_Period) nagyságú RP Holdtime értéket kell (SHOULD) megadniuk, így a rendszer képes tolerálni néhány BSM és/vagy C-RP-Adv üzenet elveszését.

A C-BSR routerek arra is ügyelnek, hogy a megválasztott BSR router működik-e, ellenkező esetben új BSR-t választanak. Ha egy C-BSR router *BS_Timeout* ideig (alapértelmezett érték 130 másodperc) nem kap BSM üzenetet, akkor egy BSM üzenet kibocsátásával elindít egy BSR választást.

3.4. Linux rendszerek konfigurálása

Könyvünkben a Debian alapú Linux disztribúciók beállítását ismertetjük, aminek fő oka a Debian és az Ubuntu disztribúciók széleskörű elterjedése. Ebben a fejezetben csak a natív IPv6 konfigurálását ismertetjük részletesen. A különböző áttérési technikák (tunnel, NAT) ismertetésével és vizsgálatával külön fejezetek foglalkoznak. A leírtak alkalmazásakor figyelembe kell venni, hogy a Linux kernel IPv6 támogatása gyorsan és folyamatosan változik, fejlődik; egyre újabb funkciók jelennek meg, az alapértelmezett értékek, beállítások változnak, így a leírtaktól eltérő működés is előfordulhat.

3.4.1. Alapbeállítások

Az egyes hálózati vezérlőkhöz rendelt IPv6 címeket az IPv4 esetében is megszokott `ifconfig` parancs alkalmazásával kérdezhetjük le:

```
root@Debian:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:15:5d:20:42:33
          inet addr:10.0.0.10  Bcast:10.0.0.127  Mask:255.255.255.192
          inet6 addr: 2001:db8::215:5dff:fe20:4233/64 Scope:Global
          inet6 addr: fe80::215:5dff:fe20:4233/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:22015 errors:0 dropped:3138 overruns:0 frame:0
          TX packets:8063 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1746432 (1.6 MiB)  TX bytes:920169 (898.6 KiB)
```

Jól látható az IPv6 működéséhez szükséges link local, valamint a módosított EUI-64 alkalmazásával automatikusan használatba vett IPv6-os cím is. Ebben az esetben az útválasztón, mellyel a számítógép kapcsolatban van, engedélyezett a Router advertisement, így a korábban már ismertetett SLAAC segítségével automatikusan jutott IPv6 címhez a számítógép. Ha nem alkalmazunk SLAAC-t és DHCP-t, vagy egy könnyebben megjegyezhető címet szeretnénk fixen a számítógéphez rendelni, úgy a `/etc/network/interfaces` állományban tehetjük ezt meg az `inet6` stanza alkalmazásával (feltéve, hogy nem használunk például Network Managert). A hálózati maszk hosszát a bitek darabszámával adhatjuk meg:

```
iface eth0 inet6 static
  address 2001:DB8::10
  netmask 64
  gateway 2001:DB8::1
```

Ha rendelkezünk IPv6 DHCP szerverrel és stateful auto address konfigurációt alkalmazunk, akkor az IPv4 protokollhoz hasonlóan a `static` szócska helyére a `dhcp`-t kell írni. Ha kifejezetten az SLAAC alkalmazását szeretnénk, akkor pedig az `auto` a megfelelő szó.
A Linux verziótól függően előfordulhat, hogy nem csak a statikusan beállított címet használja számítógépünk, hanem az SLAAC segítségével használatba vettet is:

```
root@Debian:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:15:5d:20:42:33
          inet addr:10.0.0.10  Bcast:10.0.0.127  Mask:255.255.255.192
          inet6 addr: 2001:db8::215:5dff:fe20:4233/64 Scope:Global
          inet6 addr: fe80::215:5dff:fe20:4233/64 Scope:Link
          inet6 addr: 2001:db8::10/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:112 errors:0 dropped:7 overruns:0 frame:0
          TX packets:92 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16145 (15.7 KiB)  TX bytes:10324 (10.0 KiB)
```

Ezt érdemes megszüntetni, mert nem kívánt működést is előidézhet. Több megoldást is alkalmazhatunk, melyek közül talán a legegyszerűbb a hálózati vezérlők konfigurációját tartalmazó állomány módosításával a SLAAC alkalmazásának letiltása:

```
iface eth0 inet6 static
  address 2001:DB8::10
  netmask 64
  gateway 2001:DB8::1
post-up echo 0 > /proc/sys/net/ipv6/conf/default/accept_ra
post-up echo 0 > /proc/sys/net/ipv6/conf/all/accept_ra
post-up echo 0 > /proc/sys/net/ipv6/conf/$IFACE/accept_ra
post-up echo 0 > /proc/sys/net/ipv6/conf/default/autoconf
post-up echo 0 > /proc/sys/net/ipv6/conf/all/autoconf
post-up echo 0 > /proc/sys/net/ipv6/conf/$IFACE/autoconf
```

Az IPv6 névszerverek beállítását az IPv4-hez hasonlóan a `/etc/resolv.conf` állományban végezhetjük el. Használhatunk csak IPv4-es, vagy csak IPv6-os, vagy mindkét típusú névszervereket

egyaránt. A DNS működéséből adódóan mindegy, hogy milyen verziójú IP címen érhető el a névszerver, az nem befolyásolja a működést.

A routing tábla manipulációját az `ip` parancs alkalmazásával érdemes elvégezni. Amennyiben a parancs alkalmazásakor megadjuk a `-6` paramétert, úgy a kiadott parancsok az IPv6 protokollra érvényesek. Ezek alapján a routing tábla kiíratása:

```
root@Debian:~# ip -6 route
2001:db8::/64 dev eth0  proto kernel  metric 256
fe80::/64 dev eth0  proto kernel  metric 256
ff00::/8 dev eth0  metric 256
default via 2001:db8::1 dev eth0  metric 1
```

Új bejegyzés hozzáadása a táblához, tehát egy statikus útvonal (route) létrehozása az `add` paraméterrel lehetséges. Például egy olyan statikus route, mely szerint a 2001:db8:1::/64 hálózat a 2001:db8::2 IPv6 című átjárón érhető el, a következő paranccsal hozható létre:

```
root@Debian:~# ip -6 route add 2001:db8:1::/64 via 2001:db8::2
```

A parancs eredménye pedig:

```
root@Debian:~# ip -6 route ls
2001:db8:1::/64 via 2001:db8::2 dev eth0  metric 1024
2001:db8::/64 dev eth0  proto kernel  metric 256
fe80::/64 dev eth0  proto kernel  metric 256
ff00::/8 dev eth0  metric 256
default via 2001:db8::1 dev eth0  metric 1
```

Route bejegyzés törlése a del paraméterrel lehetséges, melyre példa:

```
root@Debian:~# ip -6 route del 2001:db8:1::/64 via 2001:db8::2
```

Ha a számítógép egyik hálózati vezérlőkártyájához több IPv6-os címet is szeretnénk hozzárendelni, úgy a helyes működés érdekében azt az `ip -f inet6 addr add` paranccsal végezzük, ne az IPv4-nél már megszokott `eth0:0` aliast alkalmazzuk. Például:

```
root@Debian:~# ip -f inet6 addr add 2001:db8::3456/64 dev eth0
```

Eredménye pedig a következő:

```
ifcroot@Debian:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:15:5d:20:42:33
          inet addr:10.0.0.10  Bcast:10.0.0.127  Mask:255.255.255.192
          inet6 addr: fe80::215:5dff:fe20:4233/64 Scope:Link
          inet6 addr: 2001:db8::3456/64 Scope:Global
          inet6 addr: 2001:db8::10/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:112 errors:0 dropped:7 overruns:0 frame:0
          TX packets:92 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16145 (15.7 KiB)  TX bytes:10324 (10.0 KiB)
```

A már feleslegessé vált másodlagos IPv6 cím eltávolítására példa:

```
root@Debian:~# ip -f inet6 addr del 2001:db8::3456/64 dev eth0
```

Mint már az olvasó is tisztában van vele, az IPv6 protokoll az IPv4-től eltérően nem az ARP protokollt alkalmazza az egyes hálózati (L3) címekhez tartozó fizikai (L2) címek feloldására, hanem az NDP-t. Az Linux NDP táblázatának manipulálásához az `ip -6 neigh` parancs használható. A táblázat kiíratására példa:

```
root@Debian:~# ip -6 neigh show
fe80::20c:42ff:fe9a:4b7a dev eth0 lladdr 00:0c:42:9a:4b:7a router REACHABLE
2001:db8::1 dev eth0 lladdr 00:0c:42:9a:4b:7a router REACHABLE
```

A könyv szerzőinek tapasztalata szerint sajnos sokszor előfordul, hogy számítógépünk nem talál egy szomszédos számítógépet. Ilyenkor sok esetben segít, ha a táblázatból a nem elérhető eszközt manuálisan töröljük, mely után általában megjavul a hálózati kapcsolat a két számítógép közt. Erre példa:

```
root@Debian:~# ip -6 neigh del 2001:db8::1 dev eth0
```

Az IPv6 csomagok továbbítását a `/proc` file rendszeren keresztül engedélyezhetjük legegyszerűbben:

```
root@Debian:~# sysctl net.ipv6.conf.all.forwarding=1
```

Ha szeretnénk, hogy beállításunk újraindítás után is megmaradjon, akkor célszerűen a `/etc/sysctl.conf` állományban végezzük el ugyanezt a beállítást. Tiltani pedig a 0 érték beállításával lehet.

A hálózati kapcsolatok ellenőrzése a `ping6` és a `traceroute6` parancsok segítségével lehetséges, míg a többi parancs megegyezik a két protokoll esetében:

```
root@Debian:~# ping6 cisco.com -c 4
PING cisco.com(www1.cisco.com) 56 data bytes
64 bytes from www1.cisco.com: icmp_seq=1 ttl=232 time=145 ms
64 bytes from www1.cisco.com: icmp_seq=2 ttl=232 time=145 ms
64 bytes from www1.cisco.com: icmp_seq=3 ttl=232 time=145 ms
64 bytes from www1.cisco.com: icmp_seq=4 ttl=232 time=145 ms

--- cisco.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3011ms
rtt min/avg/max/mdev = 145.316/145.400/145.434/0.469 ms
```

Ha egy parancs paramétereként IPv6 címet kell megadnunk, de nem egyértelmű, hogy hol vannak a cím határai, úgy a címet [ ] jelek közé írhatjuk. Például:

```
root@Debian:~# scp * [::1]:/home
```

3.4.2. ip6tables

Az IPv4 esetében alkalmazott `iptables` parancsnak az IPv6 protokoll esetén alkalmazott megfelelője az `ip6tables`. Hasonlóságuk miatt csak az `ip6tables` legfontosabb parancsait ismertetjük példák segítségével. Az A parancs segítségével egy meglévő lánc végéhez fűzhetünk újabb sort, míg az I parancs segítségével a lánc elejére szúrhatunk be. Törölni a D paranccsal lehet, míg az L-lel listázhatjuk ki a láncot.

Egy sor beszúrása a lánc végére, mely minden csomagot eldob, ami a 2001:db8::100 címről érkezik:

```
root@Debian:~# ip6tables -A INPUT -s 2001:db8::100 -j DROP
```

Ha ezt a sort a lánc elejére szeretnénk beszúrni:

```
root@Debian:~# ip6tables -I INPUT -s 2001:db8::100 -j DROP
```

Kiírja a láncok tartalmát a megfelelt csomagok és byteok számával együtt:

```
root@Debian:~# ip6tables -L -v
```

Törli az első lépésben létrehozott sort:

```
root@Debian:~# ip6tables -D INPUT -s 2001:db8::100 -j DROP
```

3.4.3. Automatikus IPv6 címkiosztás

Az IPv6 címek automatikus kiosztása eltér az IPv4 esetében megszokottól. Történhet stateless módon a radvd segítségével, valamint stateful módon DHCP szerver és a radvd kombinált alkalmazásával. A radvd alkalmazása kisméretű hálózatok esetén célszerű, ahol nincs szükség egy teljes értékű DHCP szerver által nyújtott funkcionalításra, elég csupán az IP cím, átjáró, DNS szerver automatikus hozzárendelése. Nagyobb méretű hálózatokban, vagy speciális igények esetén javasolt DHCP szervert is használni.

Router Advertisement Daemon (radvd)

Ha SLAAC segítségével kívánjuk a számítógépek IPv6 címét beállítani, akkor erre Linux rendszerek esetén a Router Advertisement Daemon (radvd) alkalmazása javasolt. A radvd segítségével, több más paraméter mellett automatikusan megoldható a prefix, valamint annak élettartama, és a DNS szerver(ek) IP címének beállítása is. Mivel sok helyen használnak Linux alapokon működő routert így fontosnak tartottuk bemutatni működését.

A radvd daemon beállítása a `/etc/radvd.conf` konfigurációs állomány segítségével lehetséges. A pontos szintaxist, valamint a használható definíciókat a `man radvd.conf` parancs segítségével írathatjuk ki. Mi csak a fontosabbakat mutatjuk be:

- Interfész definíciója:

```
interface name
{
  interfész specifikus opciók
  prefix definíciók
  azon kliensek (listája), akik részére hirdetés történjen
  útvonal definíciók
```

```
  rekuzív DNS szerver (RDNSS) definíciók
};
```

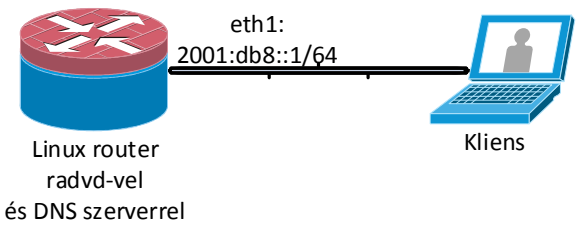
- Prefix definíció:

```
prefix prefix/length
{
  list of prefix specific options
};
```

- rekuzív DNS szerver definíció:

```
RDNSS ip [ip] [ip]
{
  rdnss specifikus információk
};
```

A 14. ábrán látható példa alapján végezzük el a beállítást.



14. ábra: Linux radvd és DNS szerver

A SLAAC beállításának első lépéseként telepítsük a radvd programot az `apt-get install radvd` parancs segítségével, majd (a már megismert módon) engedélyezzük az IPv6 csomagok továbbítását, ha az még nem lenne engedélyezve!

Hozzuk létre a `/etc/radvd.conf` állományt a következő tartalommal:

```
interface eth1
{
  AdvSendAdvert on;
  prefix 2001:db8::/64 {};
  RDNSS 2001:db8::1 {};
};
```

Az `AdvSendAdvert on` segítségével a periódikus Router Advertisement (RA) üzenetek kiküldését, valamint a válaszadást a router solicitation üzenetekre engedélyezzük. A prefix paraméter mutatja meg a SLAAC kilens számítógépnek, hogy az általa generált IPv6 cím mely hálózatba tartozzon. Az `RDNSS` segítségével pedig a DNS szerver IPv6 címét továbbíthatjuk a kliensnek (RFC 6106 alapján). Fontos információ, hogy nem minden kliens használja fel az így kapott DNS információkat. Az általunk használt Debian Linux rendszert futtató kliensen a működés érdekében előbb telepíteni kell a rdnssd csomagot (`apt-get install rdnssd`). (Megoldás lehet még az is, ha a radvd mellett telepítünk egy DHCP szervert is, amely azonban csak a DNS szerver címét segít beállítani a klienseknek. Ilyenkor a címkiosztás szintén stateless módon történik.)

Utolsó lépésként indítsuk el a radvd daemont a `/etc/init.d/radvd start` parancs kiadásával, majd vizsgáljuk meg a kliens hálózati interfészének beállításait!

A már ismert információk szerint, a SLAAC metódus során a kliens saját IPv6 címének utolsó 64 bitjét a módosított EUI-64 algoritmus segítségével önállóan állítja elő, ahogy esetünkben is jól látható:

```
eth1      Link encap:Ethernet  HWaddr 00:15:5d:28:65:35
          inet6 addr: 2001:db8::215:5dff:fe28:6535/64 Scope:Global
          inet6 addr: fe80::215:5dff:fe28:6535/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:30 errors:0 dropped:0 overruns:0 frame:0
          TX packets:34 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3769 (3.6 KiB)  TX bytes:3054 (2.9 KiB)
```

Ez a módszer azonban biztonsági problémákat idézhet elő, hiszen az IPv6 cím vége viszonylag jól klienshez köthető. Ennek a kezelésére dolgozták ki az RFC 4941-ben rögzített privacy extensions-t, melyet az általunk is használt Debian Linux kliensen a `/proc` file rendszeren keresztül engedélyezhetjük a kívánt interfészen, például a következő módon, a `/etc/network/interfaces` állományban:

```
iface eth1 inet6 auto
  pre-up sysctl -w net.ipv6.conf.eth1.use_tempaddr=2
```

A megadott 2-es érték azt eredményezi, hogy rendszerünk a kommunikáció során az ideiglenes IPv6 globális címet preferálja a módosított EUI-64 algoritmus segítéségvel generálttal szemben, míg 1-es érték esetén a helyzet fordított. 0 érték (alapértelmezés szerinti) esetén nem kerül létrehozásra ideiglenes IPv6 cím. A megadott konfiguráció esetén az interfész beállítása a következő:

```
eth1      Link encap:Ethernet  HWaddr 00:15:5d:28:65:35
          inet6 addr: 2001:db8::bd92:1aea:d646:9f30/64 Scope:Global
          inet6 addr: 2001:db8::215:5dff:fe28:6535/64 Scope:Global
          inet6 addr: fe80::215:5dff:fe28:6535/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:18 errors:0 dropped:0 overruns:0 frame:0
          TX packets:26 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2087 (2.0 KiB)  TX bytes:2310 (2.2 KiB)
```

Érdemes rá figyelni, hogy a privacy extension alkalmazása ugyan növeli a biztonságot, ugyanakkor okozhat problémákat a működés során.

Ha valamilyen okból a kliens nem állítja be automatikusan a kívánt paramétereket, akkor Debian Linuxot futtató kliens esetében jól használható hibakereső eszköz az `rdisc6` parancs, mely az nidsc6 a csomag része:

```
root@Debian:~# rdisc6 eth1
Soliciting ff02::2 (ff02::2) on eth1...

Hop limit                 :           64 (      0x40)
Stateful address conf.    :           No
```

```
Stateful other conf.      :           No
Router preference         :       medium
Router lifetime           :         1800 (0x00000708) seconds
Reachable time            :  unspecified (0x00000000)
Retransmit time           :  unspecified (0x00000000)
 Prefix                   : 2001:db8::/64
  Valid time              :        86400 (0x00015180) seconds
  Pref. time              :        14400 (0x00003840) seconds
 Recursive DNS server     : 2001:db8::1
  DNS server lifetime     :          600 (0x00000258) seconds
 Source link-layer address: 00:15:5D:28:65:34
 from fe80::215:5dff:fe28:6534
```

ISC DHCPD és radvd együttes alkalmazása

A széles körben alkalmazott, Internet Systems Consortium (ISC) által fejlesztett, szabad szoftverként elérhető ISC DHCPD a 4.x verziójától kezdődően az IPv4 mellett már az IPv6 protokollt is támogatja. Fontos azonban, hogy egy időben csak az egyik protokollal hajlandó működni, tehát ha egyidőben IPv4 és IPv6 dhcp szerverre is szükségünk van, akkor két példányban kell futtatni a daemont. A következőkben csak az IPv6 önálló alkalmazásához szükséges alapbeállításokat mutatjuk be. Példakánt a SLAAC bemutatásakor is használt topológiát használjuk azzal a különbséggel, hogy a radvd mellett DHCP szerver is fut a Linux routeren.

Az IPv4 protokoll alkalmazásakor a DHCP szerver segítségével kerülnek kiosztásra az alapátjáróval kapcsolatos információk is, azonban az IPv6 esetében nem ez a módszer került kiválasztásra, így szükség van a radvd futtatására is. Ehhez azonban a már megismerttől eltérő konfigurációs állományt kell alkalmazni. Hozzuk létre a `/etc/radvd.conf` konfigurációs állományt a következő tartalommal:

```
interface eth1
{
  AdvSendAdvert on;
  AdvManagedFlag on;
  AdvOtherConfigFlag on;
  MinRtrAdvInterval 3;
  MaxRtrAdvInterval 60;
};
```

Az `AdvManagedFlag on` segítségével engedélyezzük a stateful működést, azaz az IPv6 cím kérését a DHCP szervertől. Az `AdvOtherConfigFlag on` pedig engedélyezi a kliens számára, hogy a DHCP szervertől egyéb beállításokat is elfogadjon. A két intervallumot (`MinRtrAdvInterval` és `MaxRtrAdvInterval`) azért kell beállítani, mert az alapértelmezés szerinti értékek olyan magasak (200 és 600 másodperc), hogy a kliens sokszor csak indulása után percekkel értesülne az alapértelmezett átjáró címéről, ugyanis ezek a paraméterek állítják be a radvd-nek a router advertisement hirdetések közti időintervallumot. Indítsuk újra a radvd-t a `/etc/init.d/radvd restart` parancs kiadásával!

A DHCP szerver telepítését az `apt-get install isc-dhcp-server` paranccsal végezhetjük el. Ezután a `/etc/default/isc-dhcp-server` állományban be kell állítani az IPv6 protokoll használatát a `OPTIONS="-6"` sor segítségével. A szerver telepítése során automatikusan létrejön a `/etc/dhcp/dhcpd.conf` állomány, melyben a DHCP szerver beállításait elvégezhetjük. Az állományból nem törölve, ahhoz adjuk hozzá a következő sorokat:

```
subnet6 2001:db8::/64
{
  range6 2001:db8::100 2001:db8::200;
  option dhcp6.name-servers 2001:db8::1;
  option dhcp6.domain-search "mylan.hu";
}
```

A `subnet6` mondja meg, hogy a DHCP szerver mely interfészen fog működni, a `range6`, pedig a kiosztható IPv6 címek tartományát határozza meg. A két `option` a névkiszolgáló címét és a kereséséi zóna nevét adja meg. (Ha szeretnénk prefix delegációt is használni, úgy a `prefix6` kulcsszóval adhatjuk me az e célra felhasználható tartományt.)

Hozzuk létre a kiosztott IPv6 címekkel kapcsolatos információkat tároló állományt a `touch /var/lib/dhcp/dhcpd6.leases` parancs segítségével, majd indítsuk el a dhcpd daemont a `/etc/init.d/isc-dhcp-server start` parancs kiadásával.

Ezek után a kliens megkapja a hálózat eléréséhez szükséges adatokat. Ha Debian Linuxot futtató klienst használunk, lehetséges, hogy a kliens nem állítja be az alapátjáró címét. Ennek oka, hogy ez a kliens DHCP alkalmazása során letiltja a RA csomagok figyelését. A probléma egyszerű módon kiküszöbölhető a `/etc/network/interfaces` állományban, a megfelelő interfésznél:

```
iface eth1 inet6 auto
  post-up sysctl -w net.ipv6.conf.eth1.accept_ra=1
```

A kliens interfészének beállításainál láthatjuk, hogy a cím a megadott tartományból került kiosztásra:

```
root@Debian:~# ifconfig
eth1      Link encap:Ethernet  HWaddr 00:15:5d:28:65:35
          inet6 addr: 2001:db8::1af/64 Scope:Global
          inet6 addr: fe80::215:5dff:fe28:6535/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:93 errors:0 dropped:0 overruns:0 frame:0
          TX packets:49 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9339 (9.1 KiB)  TX bytes:5007 (4.8 KiB)
```

Míg a routing táblát kiíratva az átjáró is látható:

```
root@Debian:~# ip -6 route
2001:db8::/64 dev eth1  proto kernel  metric 256
fe80::/64 dev eth1  proto kernel  metric 256
default via fe80::215:5dff:fe28:6534 dev eth1  proto kernel  metric 1024
expires 177sec
```

Végül pedig, a névszerver beállítás is a DHCP szerveren megadott értéket vette fel:

```
root@Debian:~# cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by
# resolvconf(8)
nameserver 2001:db8::1
search mylan.hu
```

3.4.4. BIND

Szintén az ISC által fejlesztett, szabad szoftverként elérhető ISC BIND segítségével mutatjuk be a névszerver konfigurációt. Példánkban a DNS-ről szóló részben már ismertetett zónákt hozzuk létre.

Első lépésként telepítsük a BIND csomagot az `apt-get install bind9` parancs segítségével. Azért, hogy szerverünket ne használják fel DDoS támadásokhoz, korlátozzuk le rekurzív használatát saját hálózatunkra. Ehhez először egy ACL-t kell létrehoznunk az /etc/bind/named.conf.local állományban a következők szerint:

```
acl mynetwork { 127.0.0.0/8; ::1; 192.0.2.0/24; 2001:db8::/32; };
```

(Az ACL-ek létrehozása során az IPv4-es és IPv6-os címeket nyugodtan keverhetjük, nincs szükség külön ACL-ek létrehozására.)

Engedélyezzük a korlátozást a `/etc/bind/named.conf.options` állományban a következő sor elhelyezésével:

```
allow-recursion { mynetwork; };
```

Hozzuk létre a forward névfeloldáshoz szükséges `/etc/bind/db.isp.hu` állományt a következő tartalommal:

```
isp.hu. 3600    SOA     dns.isp.hu. root.isp.hu. (
                            2015022301   ; serial YYYYMMDDnn
                            86400        ; refresh (  24 hours)
                            7200         ; retry   (   2 hours)
                            3600000      ; expire  (1000 hours)
                            172800 )     ; minimum (   2 days)
                        NS      dns.isp.hu.
dns                     AAAA    2001:db8::1
www                     A       192.0.2.10
                        AAAA    2001:db8::10
```

Hozzuk létre a reverz névfeloldáshoz szükséges `/etc/bind/db.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa` állományt a következő tartalommal:

```
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa. 3600 SOA
dns.isp.hu. root.isp.hu. (
                            2015022301   ; serial YYYYMMDDnn
                            86400        ; refresh (  24 hours)
                            7200         ; retry   (   2 hours)
                            3600000      ; expire  (1000 hours)
                            172800 )     ; minimum (   2 days)
                        NS      dns.isp.hu.
0.1.0.0.0.0.0.0 IN      PTR     www.isp.hu.
```

Végül töltsük be a zónákat a `/etc/bind/named.conf.local` állományban elhelyezett alábbi sorokkal:

```
zone "isp.hu" {
        type master;
```

```
        file "/etc/bind/db.isp.hu";
};
zone "0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa" {
        type master;
        file
"/etc/bind/db.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa";
};
```

Majd indítsuk újra a névszervert a /etc/init.d/bind9 restart parancs kiadásával! Munkánkat ellenőrizzük le a `host` parancs segítségével:

```
root@Debian:~# host www.isp.hu ::1
Using domain server:
Name: ::1
Address: ::1#53
Aliases:

www.isp.hu has address 192.0.2.10
www.isp.hu has IPv6 address 2001:db8::10
root@Debian:~# host 2001:db8::10 ::1
Using domain server:
Name: ::1
Address: ::1#53
Aliases:

0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa
domain name pointer www.isp.hu.
```

Mint láttuk, egy IPv6-os rekordokat (is) tartalmazó zóna állomány létrehozása semmiben sem tér el IPv4-es megfelelőjétől, fontos azonban, hogy több IPv6-os opció jelent meg, melyeket IPv4-es megfelelőjüktől a -v6 végződés különböztet meg. Például:

```
transfer-source-v6
notify-source-v6
listen-on-v6
```

3.5. Cisco IOS alapú eszközök konfigurálása

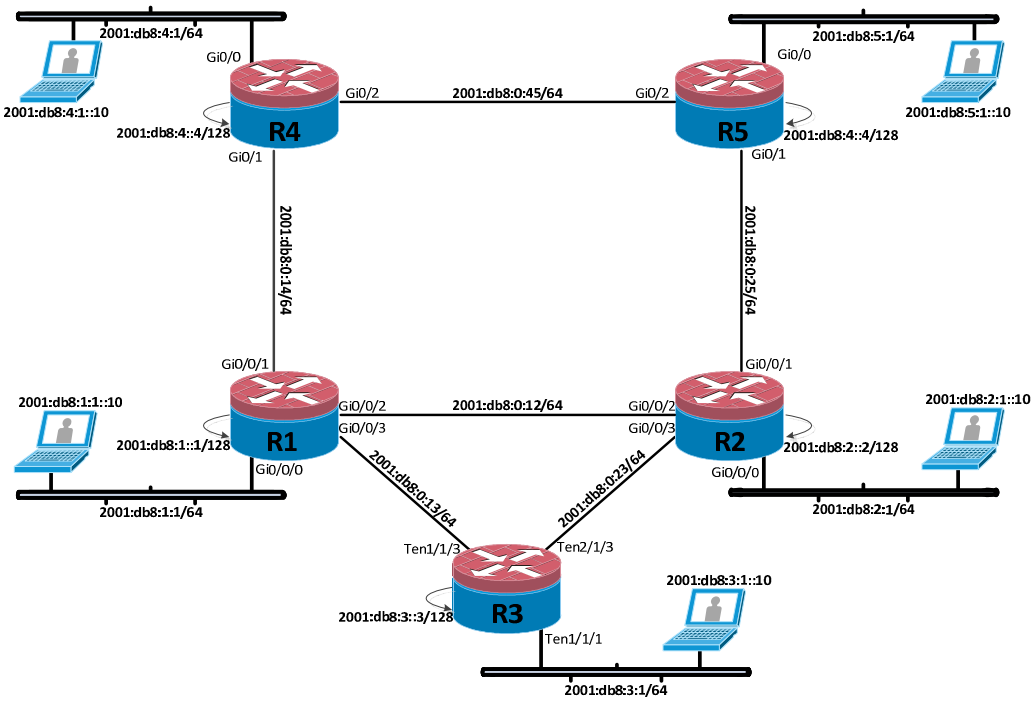
A Cisco útválasztó eszközök alkalmazása széleskörben elterjedt az Interneten és a privát hálózatokon egyaránt, így beállításuk és alkalmazásuk ismerete rendkívül fontos a gyakorlati életben. A könyv készítése során a következő népszerű hardver, és a könyv írásakor legfrissebb szoftver verziók kerültek alkalmazásra (6. táblázat).

Annak ellenére, hogy a fent felsorolt eszközök hardver konfigurációja és teljesítménye is különböző, a könyvben említett konfigurációk a közös Cisco IOS rendszer miatt megegyeznek.

Példánkban a 15. ábra szerinti topologiát fogjuk alkamazni. A topológia kialakítása során egy gyakran alkalmazott nagyvállalati megoldást használtunk, ahol minden belső hálózati pont (telephely) két központi adatközponthoz (datacenter) kapcsolódik és mindkettő adatközpont külön internet szolgáltatóval létesít kapcsolatot. Egy valós hálózat esetében természetesn több telephely (R3) lenne, és természetesen ezek a routerek is redundánsan csatlakoznának a két adatközponthoz. A 15. ábra példáját mint „állatorvosi lovat” fogjuk használni néhány elterjedt konfiguráció szemléltetéséhez.

név	hardver	IOS (XE) szoftver
Router1	ASR1001-X	03.15.00.S - 15.5(2)S
Router2	ASR1001-X	03.15.00.S - 15.5(2)S
Router3	WS-C4500X	03.07.01.E - 15.2-3.E1
Router4	CISCO2911	15.5(2)T
Router5	CISCO2911	15.5(2)T

6. táblázat: A használt Cisco eszközök paraméterei



15. ábra: BGP és OSPF példahálózat CISCO eszközökkel

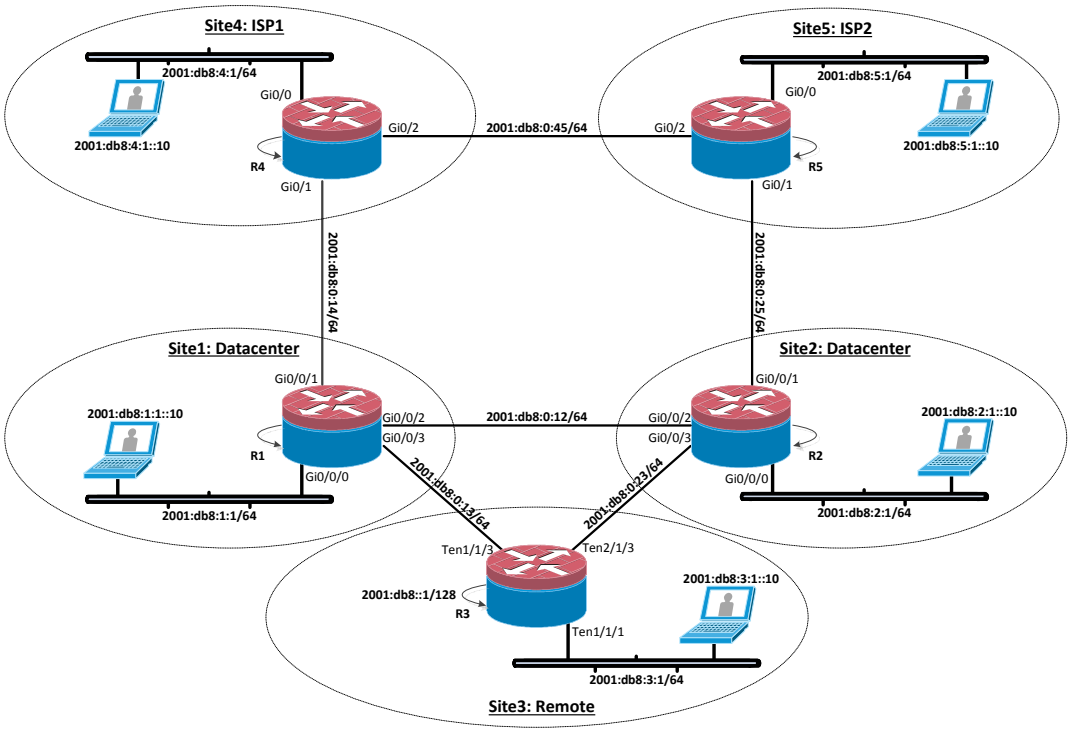
Belső hálózati routing protokollként (IGP) OSPFv3-at, míg a nyilvános Internet oldalon (EGP) BGPv4-et (MP-BGP) fogunk használni. Az OSPFv3 és a BGPv4 protokoll támogatja az IPv4 és IPv6 párhuzamos alkalmazását is, így jelenleg ez az egyik legnépszerűbb megoldás, mind a topológia mind pedig a routing protokollok kiválasztásához.

Router1 (R1) és Router2 (R2) routerek két redundáns adatközpontot jelképeznek. Ez a két router OSPFv3 routing protokollt futtat a belső hálózati forgalom, míg BGPv4 routing protokollt használ a nyilvános Internet útvonalválasztási információk cseréjéhez.

Az R3 mint „remote site" (amiből több is lehet) csak OSPFv3 routing protokollt futtat.

Az R4 és R5 routerek az internet szolgáltatókat jelképezik, BGPv4 routing protokollt használnak a nyilvános Internet útvonalválasztási információk cseréjéhez.

Éredemes megjegyezni, hogy a gyakorlatban külön routereket, tűzfalat és egyéb biztonsági-szűrési beállításokat használnak az Internet kapcsolat biztonságának növelése érdekében. A BGP-OSPF protokollok közötti hálózati címek cseréjét (redistribution) a belső hálózat biztonsági szempontjai alapján érdemes szűrni vagy engedélyezni. Az egyes telephelyek felosztása a 16. ábrán látható.



16. ábra: Eszközök elhelyezkedése az egyes telephelyeken

A példák bemutatásához terminal/text (Command line interface, CLI) alapú konfigurácót használunk, a parancsokat konfigurácós módban lehet soronkét begépelni a rendszerbe. A "`show`" parancsok használatakor csak a fontosabb sorokat fogjuk bemutatni.

3.5.1. IPv6 Konfiguráció

A Cisco IOS alapú eszközökön az IPv6 routing alapértelmezetten nincs engedélyezve, ezért a konfigurációt mindig az engedélyezéssel kell kezdeni. Ugyancsak fontos meggyőződni arról, hogy az eszközön futó szoftver alkalmas-e az IPv6, illetve a szükséges routing protokoll(ok) futtatására.

IPv6 engedélyezése

A következő sorok bemutatják a konfigurációs üzemmód és egy konfigurációs paraméter (IPv6 routing engedélyezés) változtatását. A későbbiekben csak a konfigurációs parancsok kerülnek bemutatásra.

```
R R1>enable
Password:
R1#
R1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#ipv6 unicast-routing ← ezzel engedélyezzük az unicast routingot
```

Címkonfiguráció és állapot lekérdezése

Az IPv6 cím(ek) beállítása és kiíratása a következőképpen történik:

```
R1(config)#interface GigabitEthernet0/0/0 ← interfész konfigurációs mód
R1(config-if)# ipv6 address 2001:DB8:1:1::1/64 ← IPv6 global unicast cím beállítás
R1#show ipv6 interface brief ← cím kiíratása
<...rövidítve...>
GigabitEthernet0/0/0   [up/up]
    FE80::86B8:2FF:FE1E:CA02
    2001:DB8:1:1::1
```

Jól látható, hogy a router a global unicast cím mellett a link-local címet is automatikusan engedélyezte az interfészen. A global és link-local cím manuális beállítását a következőképpen tehetjük meg:

```
R1(config)#interface gigabitEthernet 0/0/0
R1(config-if)#ipv6 address 2001:DB8:1:1::1/64
R1(config-if)#ipv6 address FE80::1 link-local ← link-local cím beállítás
R1#show ipv6 interface brief
<...rövidítve...>
GigabitEthernet0/0/0   [up/up]
    FE80::1
    2001:DB8:1:1::1
```

A továbbiak tekintsük úgy, hogy a globális IPv6 címek a fenti hálózati topológián látható módón vannak beállítva. A helyi hálózati szegmensek esetében a router interfészek a prefixhez tartozó legkisebb használható IPv6 címet kapták. Az egyes routereket összekötő hálózatokban pedig az adott router IPv6 címének utolsó számjegye megegyezik a router sorszámával (1-5).

Kapcsolat ellenőrzése

Egy kapcsolat ellenőrzéséhez használhatjuk az egyszerű `ping`, `ssh` vagy pedig a `show ipv6 neighbors` parancsot, ami az IPv6 neighbor discovery (ND) táblát mutatja meg. Az alábbi példákon bemutatjuk a kapcsolat ellenőrzését, valamint az IPv6 címek használatát mindhárom módszer alkalmazásával.

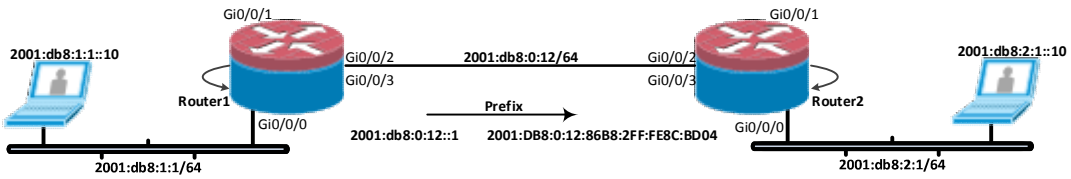
```
R1#ping 2001:DB8:0:12::2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001:DB8:0:12::2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
R1#ssh -l admin 2001:DB8:0:12::2
Password:
R2#sh ipv6 neighbors ← IPv6 szomszédok listázása
<...rövidítve...>
IPv6 Address                              Age Link-layer Addr State
Interface
2001:DB8:0:25::2                            0 a80c.0d83.4229  REACH Gi0/0/1
FE80::AA0C:DFF:FE83:4229                   89 a80c.0d83.4229  DELAY Gi0/0/1
2001:DB8:0:12::1                            0 84b8.021e.ca04  REACH Gi0/0/2
FE80::86B8:2FF:FE1E:CA04                   27 84b8.021e.ca04  STALE Gi0/0/2
2001:DB8:0:23::2                            0 0008.e3ff.fc04  REACH Gi0/0/3
FE80::208:E3FF:FEFF:FC04                    0 0008.e3ff.fc04  REACH Gi0/0/3
```

Amennyiben a helyi hálózaton más IPv6 eszközök is talalálhatóak, úgy az ICMPv6 protokoll segítségével előállított ND tábla segítségével, az IPv4 ARP táblájához hasonlóan lehet az aktív eszközök L3 és L2 címeit felderíteni.

Automatikus címek kezelése

A Cisco routerek konfigurálásakor előfordulhat, hogy a címet automatikusan szeretnénk megkapni egy másik routertől. Ilyenkor vagy a stateless automatikus konfigurációt (SLAAC), vagy a stateful DHCP módot használhatjuk.

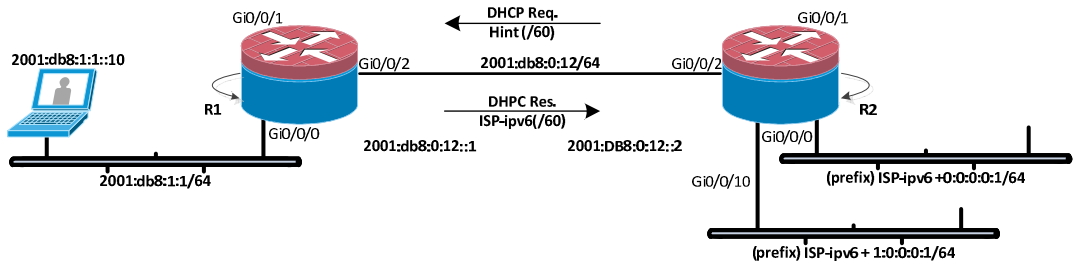
A legegyszerűbb a SLAAC módszer alkalmazása, melyet a 17. ábrán látható példán keresztül bemutatunk be, ahol R1 biztosít címet R2 részére.



17. ábra: SLAAC alkalmazása

```
R2(config)#interface gigabitEthernet 0/0/2
R2(config-if)#no ipv6 address 2001:DB8:0:12::2/64 ← állandó cím törlése
R2(config-if)#ipv6 address autoconfig ← automatikus IPv6 címbeállítás
R2#sh ipv6 interface gigabitEthernet 0/0/2
GigabitEthernet0/0/2 is up, line protocol is up
  IPv6 is enabled, link-local address is FE80::86B8:2FF:FE8C:BD04
<...rövidítve...>
  Global unicast address(es):
    2001:DB8:0:12:86B8:2FF:FE8C:BD04, subnet is 2001:DB8:0:12::/64 [EUI/CAL/PRE]
<...rövidítve...>
```

Routerek automatikus címkonfigurálásához használható a DHCPv6 protokoll is. Ezt a módszert az internetszolgáltatók általánosan elterjedt módon használják, akár a belső interfész címének beállítására is (nem csak a direkt interfész konfigurálásához). A lenti példában egy tipikus felhasználás kerül bemutatásra, ahol a szolgáltató (R1) kettő DHCP címtartományt biztosít a felhasználó részére (R2). Az R2 router az első címet a közvetlenül csatlakozó interfészére, a második címet a belső hálózati interfészére állítja be. Az R2 router akár jelezhet is (prefix delegation segítségével) a szolgáltató felé a kívánt címtartomány méretével kapcsolatban. Így a belső hálózaton akár több interfészen is konfigurálható IPv6 cím. A 18. ábra példájában az R2 router /60 címet kér, ami akár 16 darab /64 prefixű belső hálózati interfész konfigurálásához is elegendő lehet.



18. ábra: Prefix delegáció működése

R2 konfigurácó:

```
ipv6 dhcp pool ISP-pool ← IPv6 címtartomány létrehozása
 prefix-delegation pool ISP-ipv6 lifetime 1800 60 ← szolgáltatótól kapott cím
 dns-server 2001:558:FEED::1
 dns-server 2001:558:FEED::2

interface gigabitEthernet 0/0/2
ipv6 address dhcp ← szolgáltatótól kapunk címet
 ipv6 enable
 ipv6 nd autoconfig default-route
 ipv6 dhcp client pd hint ::/60 ← szolgáltatótól kérünk /60 címtartományt
 ipv6 dhcp client pd ISP-ipv6 ← ezzel a névvel azonosítjuk a /60 címtartományt

interface gigabitEthernet 0/0/0
 ipv6 address ISP-ipv6 ::0:0:0:0:1/64 ← szolgáltatótól kapott első/64 subnet
 ipv6 enable
 ipv6 nd other-config-flag
 ipv6 dhcp server ISP-pool ← általunk definiált DHCP pool, a cím a szolgáltatótól jön

interface gigabitEthernet 0/0/10
 ipv6 address ISP-ipv6 ::1:0:0:0:1/64 ← szolgáltatótól kapott második/64 subnet
 ipv6 enable
 ipv6 nd other-config-flag
 ipv6 dhcp server ISP-pool
```

3.5.2. Biztonsági beállítások

Az IPv6 címek szűrésének és kezelésének beállításánál nincs nagy különbség az IPv4 protokollhoz képest. Az alábbi példa bemutatja a firewall feature, valamint egy egyszerű Access Control List (ACL) beállítását is.

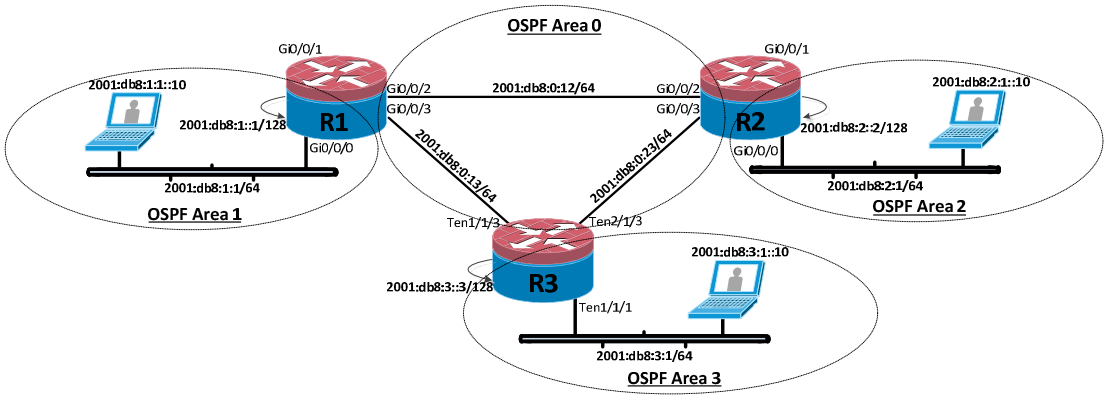
```
ipv6 inspect name TRAFFIC-V6 udp ← IPv6 firewall bekapcsolása a protokolcsaládra
ipv6 inspect name TRAFFIC-V6 tcp
ipv6 inspect name TRAFFIC-V6 icmp

ipv6 access-list INBOUNDV6 ← IPv6 példa access-list, amit az interfészen használunk majd
 permit icmp any any
 permit udp any any eq 546
 permit tcp any any established

interface gigabitEthernet 0/0/2
 ipv6 inspect TRAFFIC-V6 out ← firewall bekapcsolása a definiált protokolcsaládra
 ipv6 traffic-filter INBOUNDV6 in ← az interfészre érkező csomagokra az ACL engedélyezése
```

3.5.3. OSPFv3

Az OSPF protokoll beállításának bemutatásához 19. ábrán látható topologiát fogjuk használni. A backbone area (area 0) a központi routerek kapcsolatát biztosítja. Nagyobb hálózatok telepítésénél a diagramon látható area 1, 2 és 3 általában további routereket is magában foglal, nem csak az R1, R2 és R3 area border routereket.



19. ábra: OSPFv3 kialakítása

Konfiguráció

A Cisco OSPFv3 implementációja az IOS Release 15.1(3)S és 15.2(1)T verzióktól kezdődően támogatja az IPv6 és IPv4 protokoll együttes alkalmazását (AF), de használatához az IPv6 protokoll engedélyezése kötelező. Az OSPFv3 alapszintű konfigurációja a következő lépésekből áll:

1. OSPFv3 routing processz definiálása:
    a. A Router azonosító megadása, ami az IPv4-hez hasonlóan 32 bit hosszú, de nem kell egyeznie egyik interfész IP címével sem, csak a hálózaton belül egyedi kell, hogy legyen.
    b. Cím osztály definiálása. Itt az IPv6 kötelező, az IPv4 opcionális.
2. A protokoll engedélyezése azon az interfészen, ahol az OSPF protokollt használni szeretnénk.

Az alábbi példa bemutatja az OSPFv3 protokoll konfigurációját az R1-es routeren. Minden routeren létrehozásra került egy loopback interfész is, melynek oka, hogy ennek állapota mindig „up”, így a kapcsolódó processzek állapota nem függ egy kiválasztott valós interfész „up” vagy „down” státuszától.

A kiválasztott interfészeken a passzív üzemmód segítségével megakadályozzuk, hogy OSPF üzeneteket küldjön, vagy fogadjon a router. Ennek a megoldásnak az alkalmazása javasolt az olyan hálózati szegmenseken, ahol nincs más OSPF router, ugyanis segítségével megakadájozhatjuk, hogy hamis OSPF üzeneteket küldjenek routerünk részére.

Az alábbi konfigurácó bemutatja az R1, R2 és R3 OSPFv3 konfigurációját. Az egyszerű követhetőség érdekében a local és global címek statikusan lettek beállítva. Mind a három routernek van loopback interfésze, melyek passzív módra vannak állítva:

R1 konfiguráció:

```
interface Loopback0
 ip address 1.1.1.1 255.255.255.255
 ipv6 address FE80::1 link-local
 ipv6 address 2001:DB8:1::1/128
```

```
 ipv6 ospf 1 area 1 ← OSPF area beállítása az interfészen
!
interface GigabitEthernet0/0/0
 ipv6 address FE80::1 link-local
 ipv6 address 2001:DB8:1:1::1/64
 ospfv3 1 ipv6 area 1
!
interface GigabitEthernet0/0/1
 ipv6 address FE80::1 link-local
 ipv6 address 2001:DB8:0:14::1/64
 ospfv3 1 ipv6 area 0
!
interface GigabitEthernet0/0/2
 ipv6 address FE80::1 link-local
 ipv6 address 2001:DB8:0:12::1/64
 ospfv3 1 ipv6 area 0
!
interface GigabitEthernet0/0/3
 ipv6 address FE80::1 link-local
 ipv6 address 2001:DB8:0:13::1/64
 ospfv3 1 ipv6 area 0
 cdp enable

router ospfv3 1
 router-id 1.1.1.1 ← OSPF procesz azonostó
 !
 address-family ipv6 unicast
  passive-interface GigabitEthernet0/0/1 ← OSPF processz legyen inaktív az interfé-
szen
  passive-interface Loopback0
exit-address-family
```

R2 konfiguráció:

```
interface Loopback0
 ip address 2.2.2.2 255.255.255.255
 ipv6 address FE80::2 link-local
 ipv6 address 2001:DB8:2::2/128
 ipv6 ospf 1 area 2
!
interface GigabitEthernet0/0/0
 ipv6 address FE80::2 link-local
 ipv6 address 2001:DB8:2:1::2/64
 ospfv3 1 ipv6 area 2
!
interface GigabitEthernet0/0/1
 ipv6 address FE80::2 link-local
 ipv6 address 2001:DB8:0:25::2/64
 ospfv3 1 ipv6 area 0
!
interface GigabitEthernet0/0/2
 ipv6 address FE80::2 link-local
```

```
 ipv6 address 2001:DB8:0:12::2/64
 ospfv3 1 ipv6 area 0
!
interface GigabitEthernet0/0/3
 ipv6 address FE80::2 link-local
 ipv6 address 2001:DB8:0:23::2/64
 ospfv3 1 ipv6 area 0
!
router ospfv3 1
 router-id 2.2.2.2
 !
 address-family ipv6 unicast
  passive-interface GigabitEthernet0/0/1
  passive-interface Loopback0
exit-address-family
```

R3 konfiguráció:

```
interface Loopback0
 ip address 3.3.3.3 255.255.255.255
 ipv6 address FE80::3 link-local
 ipv6 address 2001:DB8:3::3/128
 ipv6 ospf 1 area 3
!
interface TenGigabitEthernet1/1/1
 no switchport
 no ip address
 ipv6 address FE80::3 link-local
 ipv6 address 2001:DB8:3:1::1/64
 ospfv3 1 ipv6 area 3
!
interface TenGigabitEthernet1/1/3
 ipv6 address FE80::3 link-local
 ipv6 address 2001:DB8:0:13::3/64
 ospfv3 1 ipv6 area 0
!
interface TenGigabitEthernet2/1/3
 ipv6 address FE80::3 link-local
 ipv6 address 2001:DB8:0:23::3/64
 ospfv3 1 ipv6 area 0
!
router ospfv3 1
 router-id 3.3.3.3
 !
 address-family ipv6 unicast
  passive-interface Loopback0
exit-address-family
```

Konfiguráció ellenőrzése

Az OSPF konfiguráció ellenőrzéséhez a `show ospfv3` parancs alkalmazására van szükség. Az alábbi példák bemutatják a fontosabb parancsokat és használatukat az R1 routeren:

OSPF szomszédok ellenőrzésére az alábbi parancsot használjuk. Látható, hogy R1 router a Designated Router(DR) mindkét kapcsolatra, mig R2 és R3 router Backup Designated Routerek (BDR):

```
R1#show ospfv3 neighbor

          OSPFv3 1 address-family ipv6 (router-id 1.1.1.1)

Neighbor ID     Pri   State           Dead Time   Interface ID    Interface
3.3.3.3           1   FULL/BDR        00:00:32    58              GigabitEthernet0/0/3
2.2.2.2           1   FULL/BDR        00:00:36    12              GigabitEthernet0/0/2
```

OSPF interfész ellenőrzése:

```
R1#show ospfv3 interface ← interfész OSPF paramétereinek listázása
GigabitEthernet0/0/3 is up, line protocol is up
  Link Local Address FE80::1, Interface ID 13
  Area 0, Process ID 1, Instance ID 0, Router ID 1.1.1.1
  Network Type BROADCAST, Cost: 1
  Transmit Delay is 1 sec, State DR, Priority 1
  Designated Router (ID) 1.1.1.1, local address FE80::1
  Backup Designated router (ID) 3.3.3.3, local address FE80::3
  Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
    Hello due in 00:00:00
  Graceful restart helper support enabled
  Index 1/4/4, flood queue length 0
  Next 0x0(0)/0x0(0)/0x0(0)
  Last flood scan length is 1, maximum is 6
  Last flood scan time is 0 msec, maximum is 1 msec
  Neighbor Count is 1, Adjacent neighbor count is 1
    Adjacent with neighbor 3.3.3.3  (Backup Designated Router)
  Suppress hello for 0 neighbor(s)
GigabitEthernet0/0/2 is up, line protocol is up
  Link Local Address FE80::1, Interface ID 12
  Area 0, Process ID 1, Instance ID 0, Router ID 1.1.1.1
  Network Type BROADCAST, Cost: 1
  Transmit Delay is 1 sec, State DR, Priority 1
  Designated Router (ID) 1.1.1.1, local address FE80::1
  Backup Designated router (ID) 2.2.2.2, local address FE80::2
  Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
    Hello due in 00:00:07
  Graceful restart helper support enabled
  Index 1/3/3, flood queue length 0
  Next 0x0(0)/0x0(0)/0x0(0)
  Last flood scan length is 1, maximum is 4
  Last flood scan time is 0 msec, maximum is 1 msec
  Neighbor Count is 1, Adjacent neighbor count is 1
    Adjacent with neighbor 2.2.2.2  (Backup Designated Router)
  Suppress hello for 0 neighbor(s)
Loopback0 is up, line protocol is up
  Link Local Address FE80::86B8:2FF:FE1E:CA00, Interface ID 22
  Area 1, Process ID 1, Instance ID 0, Router ID 1.1.1.1
```

```
  Network Type LOOPBACK, Cost: 1
  Loopback interface is treated as a stub Host
GigabitEthernet0/0/0 is up, line protocol is up
  Link Local Address FE80::1, Interface ID 10
  Area 1, Process ID 1, Instance ID 0, Router ID 1.1.1.1
  Network Type BROADCAST, Cost: 1
  Transmit Delay is 1 sec, State DR, Priority 1
  Designated Router (ID) 1.1.1.1, local address FE80::1
  No backup designated router on this network
  Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
    Hello due in 00:00:01
  Graceful restart helper support enabled
  Index 1/1/1, flood queue length 0
  Next 0x0(0)/0x0(0)/0x0(0)
  Last flood scan length is 0, maximum is 0
  Last flood scan time is 0 msec, maximum is 0 msec
  Neighbor Count is 0, Adjacent neighbor count is 0
  Suppress hello for 0 neighbor(s)
```

OSPF routing tábla ellenőrzésére az alábbi parancsot használjuk. Minden hálózati prefix esetén látható:

- next-hop local-link címe
- next-hop interfész neve
- „administrative distance" és OSPF cost.

```
R1#show ipv6 route ospf
IPv6 Routing Table - default - 21 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       la - LISP alt, lr - LISP site-registrations, ld - LISP dyn-eid
       a - Application
O   2001:DB8:0:23::/64 [110/2]
     via FE80::3, GigabitEthernet0/0/3
O   2001:DB8:0:25::/64 [110/3]
     via FE80::3, GigabitEthernet0/0/3
OI  2001:DB8:2::2/128 [110/2]
     via FE80::3, GigabitEthernet0/0/3
OI  2001:DB8:2:1::/64 [110/3]
     via FE80::3, GigabitEthernet0/0/3
OI  2001:DB8:3::3/128 [110/1]
     via FE80::3, GigabitEthernet0/0/3
OI  2001:DB8:3:1::/64 [110/2]
     via FE80::3, GigabitEthernet0/0/3
<...rövidítve...>
```

Érdemes megjegyezni, hogy a routing táblába csak azok az OSPF bejegyezések kerülnek, amelyek nem találhatók meg kisebb „administrative distance" segítségével, azaz olyan routing protokollal, amelyik az üzemeltető által jobban preferált. Például azok a hálózatok, amelyek közvetlenül vannak csatlakoztatva a következő módon írathatóak ki:

```
R1#show ipv6 route connected
IPv6 Routing Table - default - 21 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       la - LISP alt, lr - LISP site-registrations, ld - LISP dyn-eid
       a - Application
C   2001:DB8:0:12::/64 [0/0]
     via GigabitEthernet0/0/2, directly connected
C   2001:DB8:0:13::/64 [0/0]
     via GigabitEthernet0/0/3, directly connected
C   2001:DB8:0:14::/64 [0/0]
     via GigabitEthernet0/0/1, directly connected
LC  2001:DB8:1::1/128 [0/0]
     via Loopback0, receive
C   2001:DB8:1:1::/64 [0/0]
     via GigabitEthernet0/0/0, directly connected
```

3.5.4. OSPFv3 biztonsági beállítások

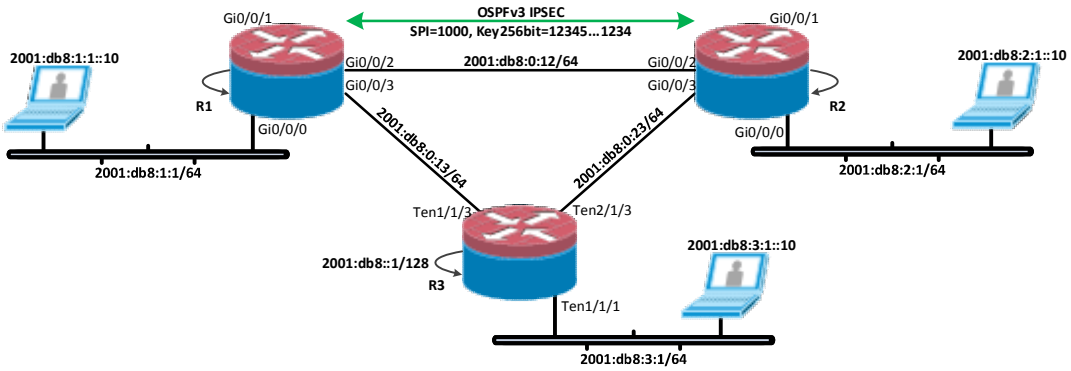
Annak érdekében, hogy az OSPFv3 csomagokat biztonságosan célba juttathassuk, anélkül, hogy valaki azokat módosítaná, vagy hamis adatokat küldene a hálózat felé, ajánlott bekapcsolni az OSPF authenticationt. Az OSPFv2-től eltérően ez nem egy egyszerű „jelszóval" lett megoldva, hanem a szabványosított IPsec protokoll segítségével. (Példánkban nem a korábban már említett Authentication trailert alkalmazzuk, hanem az IPSec-et.)

A biztonági beállítást megtehetjük az egész OSPFv3 areára, de a nagyobb biztonság érdekében, külön beállítható minden egyes interfészre is. A router a Security Policy Indexet (SPI) és a hozzá tartozó kulcsot használja a hash kulcsok beállítására és ellenőrzésére.

Az OSPF interfész biztonságának beállítása az alábbi parancs segítségével végezhető el:

ipv6 ospf encryption {**ipsec spi** *spi* **esp** {*encryption-algorithm* [[*key-encryption-type*] *key*] | **null**} *authentication-algorithm* [*key-encryption-type*] *key* | **null**}

A 20. ábra és a konfiguráció bemutatja az OSPFv3 authentication bekapcsolását a példahálózatunkon, az R1 és R2 router között, a backbone areában.



20. ábra: OSPFv3 IPsec

A konfiguráció beállítása után az interfész beállítása az R1 és R2 routereken a következő:

- Security Policy Index: 1000
- AES-CBC encryption 256 bites kulccsal, ami 64 hexadecimális számjegy hosszú.
- SHA-1 authentication, amihez egy 40 hexadecimális számjegy hosszú kulcs tartozik.

Természetesen más opciókat is lehet használni, mint például 3DES és MD5. A kulcsok a követhetőség érdekében lettek ilyen egyszerűre értékekre beállítva.

R1 konfigurácó:

```
interface GigabitEthernet0/0/2
 no ip address
 negotiation auto
 ipv6 address FE80::1 link-local
 ipv6 address 2001:DB8:0:12::1/64
 ipv6 ospf encryption ipsec spi 1000 esp aes-cbc 256
12345678901234567890123456789012345678901234567890123456789012345678901234 sha1
1234567890123456789012345678901234567890
 ospfv3 1 ipv6 area 0
```

R2 konfigurácó:

```
interface GigabitEthernet0/0/2
 no ip address
 negotiation auto
 ipv6 address FE80::2 link-local
 ipv6 address 2001:DB8:0:12::2/64
 ipv6 ospf encryption ipsec spi 1000 esp aes-cbc 256
12345678901234567890123456789012345678901234567890123456789012345678901234 sha1
1234567890123456789012345678901234567890
 ospfv3 1 ipv6 area 0
end
```

Az ellenőrzéshez általában elég csak meggyőződni arról, hogy az OSPF szomszédunk állapota FULL, de lehetőség van arra is, hogy magát az IPsec kapcsolatot az SPI index segítségével beazonosítsuk.

```
R1#show ipv6 ospf neighbor
            OSPFv3 Router with ID (1.1.1.1) (Process ID 1)

Neighbor ID     Pri   State           Dead Time   Interface ID    Interface
3.3.3.3           1   FULL/BDR        00:00:33    58              GigabitEthernet0/0/3
2.2.2.2           1   FULL/DR         00:00:33    12              GigabitEthernet0/0/2

R1#show crypto ipsec sa
interface: GigabitEthernet0/0/2
    Crypto map tag: GigabitEthernet0/0/2-OSPF-MAP, local addr FE80::1

   IPsecv6 policy name: OSPFv3-1000
   protected vrf: (none)
   local  ident (addr/mask/prot/port): (FE80::/10/89/0)
   remote ident (addr/mask/prot/port): (::/0/89/0)
   current_peer FF02::5 port 500
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 101, #pkts encrypt: 101, #pkts digest: 101
    #pkts decaps: 69, #pkts decrypt: 69, #pkts verify: 69
```

```
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0
    #pkts not decompressed: 0, #pkts decompress failed: 0
    #send errors 0, #recv errors 0

     local crypto endpt.: FE80::1,
     remote crypto endpt.: FF02::5 ← ebből látszik, hogy él a kapcsolat
     plaintext mtu 1462, path mtu 1500, ipv6 mtu 1500, ipv6 mtu idb GigabitEthernet0/0/2
     current outbound spi: 0x3E8(1000)
     PFS (Y/N): N, DH group: none

     inbound esp sas:
      spi: 0x3E8(1000)
        transform: esp-256-aes esp-sha-hmac ,
        in use settings ={Transport, }
        conn id: 2001, flow_id: HW:1, sibling_flags FFFFFFFF80000009, crypto map:
GigabitEthernet0/0/2-OSPF-MAP
        sa timing: remaining key lifetime (sec): 0
        Kilobyte Volume Rekey has been disabled
        IV size: 16 bytes
        replay detection support: N
        Status: ACTIVE(ACTIVE) ← ebből látszik, hogy él a kapcsolat

     outbound esp sas:
      spi: 0x3E8(1000)
        transform: esp-256-aes esp-sha-hmac ,
        in use settings ={Transport, }
        conn id: 2002, flow_id: HW:2, sibling_flags FFFFFFFF80000009, crypto map:
GigabitEthernet0/0/2-OSPF-MAP
        sa timing: remaining key lifetime (sec): 0
        Kilobyte Volume Rekey has been disabled
        IV size: 16 bytes
        replay detection support: N
        Status: ACTIVE(ACTIVE) ← ebből látszik, hogy él a kapcsolat
```

3.5.5. IPv6 MP-BGP

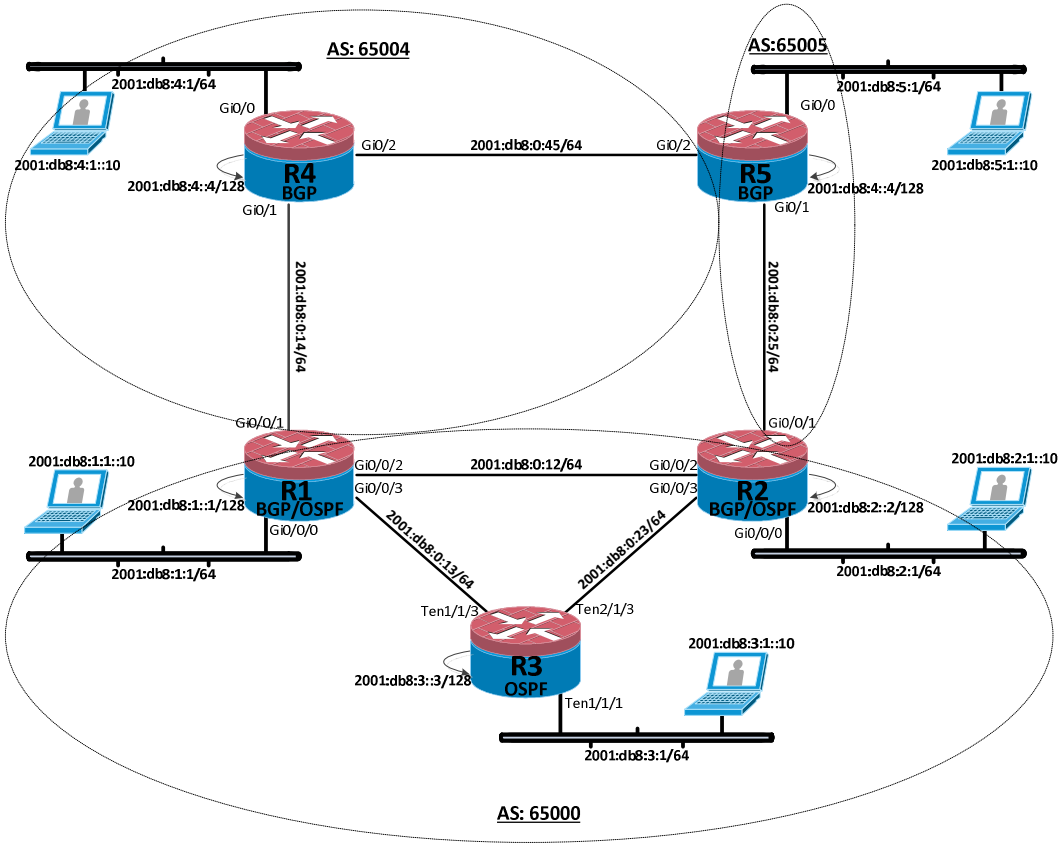
Az MP-BGP (BGPv4) protokoll konfigurálásához az idáig használt topologiához csatlakoztatjuk az R4 és R5 routereket, melyek az internetszolgáltatókat szimbolizálják. Az R1, R2 és R3 routerek az Autonomous System (AS) 65000 részei, míg az internet szolgáltatók: R4 AS65004 és R5 AS65005 (21. ábra).

Konfiguráció

Az MP-BGP implementációja egyaránt támogatja az IPv6 és IPv4 protokollt. Az MP-BGP alapszintű konfigurációja a következő lépésekből áll:

1. MP-BGP Autonomous System (AS) szám hozzárendelése a BGP processzhez.
2. A Router azonosító megadása. Ez Cisco eszközökön egy 32 bit hosszú IPv4 cím és kötelező, hogy elérehető (up státus) legyen. Ezért ezt a címet, általában az egyik loopback interfészhez szokták rendelni.
3. MP-BGP szomszéd(ok) definiálása.
4. IPv6 címcsalád definiálása és opcionálisan a saját hálózatok hirdetése és szűrése.

Az alábbi példa bemutatja az MP-BGP protokoll konfigurációját az R1, R2, R4 és az R5-es routereken a fenti diagram alapján. Segítségként a fenti diagramot használjuk a példához.



21. ábra: MP-BGP kialakítása CISCO eszközökkel

R1 konfigurácó:

```
router bgp 65000 ← BGP Autonomous System azonosítója
 bgp router-id 1.1.1.1 ← BGP router azonosítója
 neighbor 2001:DB8:0:12::2 remote-as 65000 ← BGP szomszédok definiálása
 neighbor 2001:DB8:0:14::4 remote-as 65004
!
 address-family ipv6
  network 2001:DB8:1::1/128 ← hirdetendő hálózat definiálása
  network 2001:DB8:1:1::/64
  network 2001:DB8:2::2/128
  network 2001:DB8:2:1::/64
  network 2001:DB8:3::3/128
  network 2001:DB8:3:1::/64
  neighbor 2001:DB8:0:12::2 activate ← BGP szomszéd engedélyezése
  neighbor 2001:DB8:0:14::4 activate
```

R2 konfigurácó:

```
router bgp 65000
 bgp router-id 2.2.2.2
 neighbor 2001:DB8:0:12::1 remote-as 65000
```

```
 neighbor 2001:DB8:0:25::5 remote-as 65005
 !
 address-family ipv6
  network 2001:DB8:1::1/128
  network 2001:DB8:1:1::/64
  network 2001:DB8:2::2/128
  network 2001:DB8:2:1::/64
  network 2001:DB8:3::3/128
  network 2001:DB8:3:1::/64
  neighbor 2001:DB8:0:12::1 activate
  neighbor 2001:DB8:0:25::5 activate
```

R4 konfigurácó:

```
router bgp 65004
 bgp router-id 4.4.4.4
 neighbor 2001:DB8:0:14::1 remote-as 65000
 neighbor 2001:DB8:0:45::5 remote-as 65005
 !
 address-family ipv6
  network 2001:DB8:0:14::/64
  network 2001:DB8:0:45::/64
  network 2001:DB8:4::4/128
  network 2001:DB8:4:1::/64
  neighbor 2001:DB8:0:14::1 activate
  neighbor 2001:DB8:0:45::5 activate
```

R5 konfigurácó:

```
router bgp 65005
 bgp    router-id 5.5.5.5
 neighbor 2001:DB8:0:25::2 remote-as 65000
 neighbor 2001:DB8:0:45::4 remote-as 65004
 !
 address-family ipv6
  network 2001:DB8:0:25::/64
  network 2001:DB8:5::5/128
  network 2001:DB8:5:1::/64
  neighbor 2001:DB8:0:25::2 activate
  neighbor 2001:DB8:0:45::4 activate
```

A hirdetendő hálózatokat és a BGP szomszédokat egyesével kell megadni, bár lehetőség van a hálózatok összesítésére (aggregálására) is rövidebb maszk megadásával.

Konfiguráció ellenőrzése

Az MP-BGP konfiguráció a `show bgp ipv6` parancs segítségével ellenőrizhető. Az alábbi példák bemutatják a fontosabb parancsokat és használatukat az R1 routerről:

BGP szomszédok ellenőrzése a `show bgp ipv6 unicast neighbors` parancs segítségével:

```
R1#show bgp ipv6 unicast summary
```

```
BGP router identifier 1.1.1.1, local AS number 65000
BGP table version is 89, main routing table version 89
13 network entries using 3536 bytes of memory
25 path entries using 3600 bytes of memory
12/7 BGP path/bestpath attribute entries using 2976 bytes of memory
4 BGP AS-PATH entries using 128 bytes of memory
0 BGP route-map cache entries using 0 bytes of memory
0 BGP filter-list cache entries using 0 bytes of memory
BGP using 10240 total bytes of memory
BGP activity 13/0 prefixes, 29/4 paths, scan interval 60 secs

Neighbor        V           AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down
State/PfxRcd
2001:DB8:0:12::2
                4        65000     371     374       89    0    0 04:58:50       12
2001:DB8:0:14::4
                4        65004      41      44       89    0    0 00:24:08       7
<...rövidítve...>
```

A BGP hálózati tábla ellenőrzése a `show bgp ipv6 unicast` parancs segítségével lehetséges. Fontos, hogy a tábla nem minden eleme kerül át a routing táblába, ami alapján a csomagokat a router majd továbbítja.

```
R1#show bgp ipv6 unicast
BGP table version is 113, local router ID is 1.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, m multipath, b backup-path, f RT-Filter,
              x best-external, a additional-path, c RIB-compressed,
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
 r>  2001:DB8:0:14::/64
                      2001:DB8:0:14::4
                                               0             0 65004 i
 *>i 2001:DB8:0:25::/64
                      2001:DB8:0:25::5
                                               0    100      0 65005 i
 *                    2001:DB8:0:14::4
                                                             0 65004 65005 i
 * i 2001:DB8:0:45::/64
                      2001:DB8:0:25::5
                                               0    100      0 65005 65004 i
 *>                   2001:DB8:0:14::4
                                               0             0 65004 i
 * i 2001:DB8:1::1/128
                      2001:DB8:0:12::2
                                               2    100      0 i
 *>                   ::                       0         32768 i
 * i 2001:DB8:1:1::/64
                      2001:DB8:0:12::2
                                               3    100      0 i
 *>                   ::                       0         32768 i
 *>  2001:DB8:2::2/128
                      FE80::3                  2         32768 i
 * i                  2001:DB8:0:12::2
                                               0    100      0 i
```

```
 *>  2001:DB8:2:1::/64
                       FE80::3                  3         32768 i
 * i                   2001:DB8:0:12::2
                                                0    100      0 i
 * i 2001:DB8:3::3/128
                       2001:DB8:0:12::2
                                                1    100      0 i
 *>                    FE80::3                  1         32768 i
 * i 2001:DB8:3:1::/64
                       2001:DB8:0:12::2
                                                2    100      0 i
 *>                    FE80::3                  2         32768 i
 * i 2001:DB8:4::4/128
                       2001:DB8:0:25::5
                                                0    100      0 65005 65004 i
 *>                    2001:DB8:0:14::4
                                                0             0 65004 i
 * i 2001:DB8:4:1::/64
                       2001:DB8:0:25::5
                                                0    100      0 65005 65004 i
 *>                    2001:DB8:0:14::4
                                                0             0 65004 i
 * i 2001:DB8:5::5/128
                       2001:DB8:0:25::5
                                                0    100      0 65005 i
 *>                    2001:DB8:0:14::4
                                                              0 65004 65005 i
 * i 2001:DB8:5:1::/64
                       2001:DB8:0:25::5
                                                0    100      0 65005 i
 *>                    2001:DB8:0:14::4
                                                              0 65004 65005 i
```

A routing tábla ellenőrzése a `show ipv6 route` parancs kiadásával lehetséges. Példaként nézzük meg, hogy mit látunk az R1, R4 és az R5 routereken:

Az R1 routeren látható az összes BGP router, amit R4 és R5 hirdet valamint az összes OSPF belső route amit R2 és R3 hirdet.

```
R1#sh ipv6 route
IPv6 Routing Table - default - 21 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       la - LISP alt, lr - LISP site-registrations, ld - LISP dyn-eid
       a - Application
C   2001:DB8:0:12::/64 [0/0]
     via GigabitEthernet0/0/2, directly connected
L   2001:DB8:0:12::1/128 [0/0]
     via GigabitEthernet0/0/2, receive
C   2001:DB8:0:13::/64 [0/0]
     via GigabitEthernet0/0/3, directly connected
L   2001:DB8:0:13::1/128 [0/0]
     via GigabitEthernet0/0/3, receive
C   2001:DB8:0:14::/64 [0/0]
```

```
     via GigabitEthernet0/0/1, directly connected
L   2001:DB8:0:14::1/128 [0/0]
     via GigabitEthernet0/0/1, receive
O   2001:DB8:0:23::/64 [110/2]
     via FE80::3, GigabitEthernet0/0/3
B   2001:DB8:0:25::/64 [200/0]
     via 2001:DB8:0:25::5
B   2001:DB8:0:45::/64 [20/0]
     via FE80::4, GigabitEthernet0/0/1
LC  2001:DB8:1::1/128 [0/0]
     via Loopback0, receive
C   2001:DB8:1:1::/64 [0/0]
     via GigabitEthernet0/0/0, directly connected
L   2001:DB8:1:1::1/128 [0/0]
     via GigabitEthernet0/0/0, receive
OI  2001:DB8:2::2/128 [110/2]
     via FE80::3, GigabitEthernet0/0/3
OI  2001:DB8:2:1::/64 [110/3]
     via FE80::3, GigabitEthernet0/0/3
OI  2001:DB8:3::3/128 [110/1]
     via FE80::3, GigabitEthernet0/0/3
OI  2001:DB8:3:1::/64 [110/2]
     via FE80::3, GigabitEthernet0/0/3
B   2001:DB8:4::4/128 [20/0]
     via FE80::4, GigabitEthernet0/0/1
B   2001:DB8:4:1::/64 [20/0]
     via FE80::4, GigabitEthernet0/0/1
B   2001:DB8:5::5/128 [20/0]
     via FE80::4, GigabitEthernet0/0/1
B   2001:DB8:5:1::/64 [20/0]
     via FE80::4, GigabitEthernet0/0/1
L   FF00::/8 [0/0]
     via Null0, receive
```

Az R4 routeren látható az összes BGP route, amit az R4 és R5 hirdet, valamint az összes belső route, amit a network parancsal hirdetünk az R1 és R2 routereken keresztül. Az OSPF R1, R2 és R3 routerek közti összeköttetésekhez tartozó 2001:DB8:0 route-ok azért nem láthatók itt, mert ezeket a hálózatokat nem hirdetjük az R1 vagy R2 routereken BGP protokoll segítségével.

```
R4#show ipv6 route
IPv6 Routing Table - default - 17 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       a - Application
C   2001:DB8:0:14::/64 [0/0]
     via GigabitEthernet0/1, directly connected
L   2001:DB8:0:14::4/128 [0/0]
     via GigabitEthernet0/1, receive
B   2001:DB8:0:25::/64 [20/0]
     via FE80::5, GigabitEthernet0/2
C   2001:DB8:0:45::/64 [0/0]
     via GigabitEthernet0/2, directly connected
L   2001:DB8:0:45::4/128 [0/0]
```

```
     via GigabitEthernet0/2, receive
B   2001:DB8:1::1/128 [20/0]
     via FE80::1, GigabitEthernet0/1
B   2001:DB8:1:1::/64 [20/0]
     via FE80::1, GigabitEthernet0/1
B   2001:DB8:2::2/128 [20/2]
     via FE80::1, GigabitEthernet0/1
B   2001:DB8:2:1::/64 [20/3]
     via FE80::1, GigabitEthernet0/1
B   2001:DB8:3::3/128 [20/1]
     via FE80::1, GigabitEthernet0/1
B   2001:DB8:3:1::/64 [20/2]
     via FE80::1, GigabitEthernet0/1
LC  2001:DB8:4::4/128 [0/0]
     via Loopback0, receive
C   2001:DB8:4:1::/64 [0/0]
     via GigabitEthernet0/0, directly connected
L   2001:DB8:4:1::4/128 [0/0]
     via GigabitEthernet0/0, receive
B   2001:DB8:5::5/128 [20/0]
     via FE80::5, GigabitEthernet0/2
B   2001:DB8:5:1::/64 [20/0]
     via FE80::5, GigabitEthernet0/2
L   FF00::/8 [0/0]
     via Null0, receive
```

Mivel az R3 routeren csak az OSPF protokollt futtatjuk, ezért csak az R1 és R2 routerek OSPF táblája kerül szinkronizálásra. Az R4 és R5 routerek által hirdetett hálózatok elérésére csak a default route segítségével lehetséges a jelenlegi konfiguráció mellett.

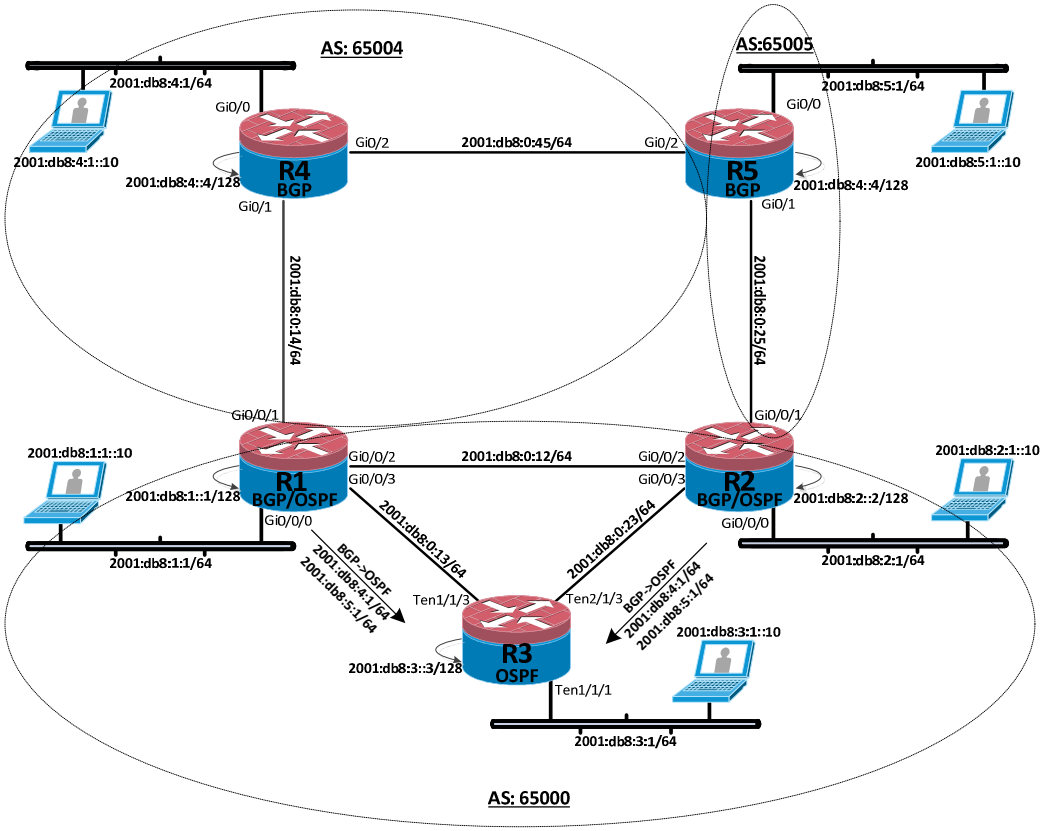
```
R3#sh ipv6 route
IPv6 Routing Table - default - 14 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, RL - RPL, O - OSPF Intra, OI - OSPF Inter
       OE1 - OSPF ext 1, OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1
       ON2 - OSPF NSSA ext 2, a - Application
O   2001:DB8:0:12::/64 [110/2]
     via FE80::1, TenGigabitEthernet1/1/3
     via FE80::2, TenGigabitEthernet2/1/3
C   2001:DB8:0:13::/64 [0/0]
     via TenGigabitEthernet1/1/3, directly connected
L   2001:DB8:0:13::2/128 [0/0]
     via TenGigabitEthernet1/1/3, receive
L   2001:DB8:0:13::3/128 [0/0]
     via TenGigabitEthernet1/1/3, receive
C   2001:DB8:0:23::/64 [0/0]
     via TenGigabitEthernet2/1/3, directly connected
L   2001:DB8:0:23::3/128 [0/0]
     via TenGigabitEthernet2/1/3, receive
OI  2001:DB8:1::1/128 [110/1]
     via FE80::1, TenGigabitEthernet1/1/3
OI  2001:DB8:1:1::/64 [110/2]
     via FE80::1, TenGigabitEthernet1/1/3
OI  2001:DB8:2::2/128 [110/1]
```

```
     via FE80::2, TenGigabitEthernet2/1/3
OI   2001:DB8:2:1::/64 [110/2]
     via FE80::2, TenGigabitEthernet2/1/3
LC   2001:DB8:3::3/128 [0/0]
     via Loopback0, receive
C    2001:DB8:3:1::/64 [0/0]
     via TenGigabitEthernet1/1/1, directly connected
L    2001:DB8:3:1::1/128 [0/0]
     via TenGigabitEthernet1/1/1, receive
L    FF00::/8 [0/0]
     via Null0, receive
```

MP-BGP és OSPFv3 hálózati hirdetések cseréje

Előfordulhat, hogy bizonyos partner hálózatok vagy hálózat specifikus útválasztás miatt némely BGP hálózatot szeretnénk a belső OSPF routing táblába hirdetni. A „redistribute" funkcióval néhány, vagy akár az összes BGP hálózatot át tudjuk vinni az OSPF táblába, külső hálózatként. Ezeknek az internet címeknek átvitele OSPF protokollba a következő célokat is szolgálhatják:

- Bizonyos partner hálózatot normál körülmények között egy szolgáltató felé szeretnék elérni politikai, késleltetés, sávszélesség vagy egyébb okok miatt.
- Egy belső (IGP) router az útválasztási döntését egy bizonyos hálózat elérhetőségétől teszi függővé.



22. ábra: Redisztribúció BGP és OSPF között

Természetesen a teljes BGP hálózati tábla átadása OSPF-be szinte soha nem praktikus, vagy lehetséges a BGP tábla hatalmas mérete (és az ebből következő memóriaigénye) miatt. A BGP hálózati címek átadása csak kontrollált módon, szűrve ajánlott (lásd az alábbi példát). A külső hálózatok többségét általában a „default route" használatával fogjuk elérni.

Nézzünk meg egy példát arra az esetre, ha az R4 és R5 BGP által hirdetett hálózatát az R1 valamint az R2 routeren keresztül az OSPF táblába visszük át (22. ábra).

R1 konfiguráció:

```
router ospfv3 1
 router-id 1.1.1.1
 !
 address-family ipv6 unicast
  passive-interface GigabitEthernet0/0/1
  passive-interface Loopback0
  redistribute bgp 65000 metric 1000 metric-type 1 route-map BGP-OSPF ←
BGP hálózat redisztribuciója a „BGP-OSPF" route map segitségével
 exit-address-family
!
router bgp 65000
 bgp router-id 1.1.1.1
 neighbor 2001:DB8:0:12::2 remote-as 65000
 neighbor 2001:DB8:0:14::4 remote-as 65004
 !
 address-family ipv6
  network 2001:DB8:1::1/128
  network 2001:DB8:1:1::/64
  network 2001:DB8:2::2/128
  network 2001:DB8:2:1::/64
  network 2001:DB8:3::3/128
  network 2001:DB8:3:1::/64
  neighbor 2001:DB8:0:12::2 activate
  neighbor 2001:DB8:0:14::4 activate
!
ipv6 prefix-list R4-R5-NET seq 5 permit 2001:DB8:4:1::/64 ← prefix-list a külső
hálózat engedélyezésére
ipv6 prefix-list R4-R5-NET seq 10 permit 2001:DB8:5:1::/64
route-map BGP-OSPF permit 10 ← route-map, amit a redisztribucióra használunk
 match ipv6 address prefix-list R4-R5-NET
```

R2 konfiguráció:

```
router ospfv3 1
 router-id 2.2.2.2
 !
 address-family ipv6 unicast
  passive-interface GigabitEthernet0/0/1
  passive-interface Loopback0
  redistribute bgp 65000 metric 1000 metric-type 1 route-map BGP-OSPF
 exit-address-family
!
```

```
router bgp 65000
 bgp router-id 2.2.2.2
 neighbor 2001:DB8:0:12::1 remote-as 65000
 neighbor 2001:DB8:0:25::5 remote-as 65005
 !
address-family ipv6
  network 2001:DB8:1::1/128
  network 2001:DB8:1:1::/64
  network 2001:DB8:2::2/128
  network 2001:DB8:2:1::/64
  network 2001:DB8:3::3/128
  network 2001:DB8:3:1::/64
  neighbor 2001:DB8:0:12::1 activate
  neighbor 2001:DB8:0:25::5 activate
!
ipv6 prefix-list R4-R5-NET seq 5 permit 2001:DB8:4:1::/64
ipv6 prefix-list R4-R5-NET seq 10 permit 2001:DB8:5:1::/64
route-map BGP-OSPF permit 10
 match ipv6 address prefix-list R4-R5-NET
```

Ha megvizsgáljuk az R3 router tábláját, már látható, hogy R4 és R5 lokális hálózata megjelent, mint external OSPF route.

```
R3# show ipv6 route
IPv6 Routing Table - default - 18 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user Static route
       B - BGP, R - RIP, H - NHRP, I1 - ISIS L1
       I2 - ISIS L2, IA - ISIS interarea, IS - ISIS summary, D - EIGRP
       EX - EIGRP external, ND - ND Default, NDp - ND Prefix, DCE - Destination
       NDr - Redirect, RL - RPL, O - OSPF Intra, OI - OSPF Inter
       OE1 - OSPF ext 1, OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1
       ON2 - OSPF NSSA ext 2, a - Application
O   2001:DB8:0:12::/64 [110/2]
     via FE80::2, TenGigabitEthernet2/1/3
     via FE80::1, TenGigabitEthernet1/1/3
C   2001:DB8:0:13::/64 [0/0]
     via TenGigabitEthernet1/1/3, directly connected
L   2001:DB8:0:13::2/128 [0/0]
     via TenGigabitEthernet1/1/3, receive
L   2001:DB8:0:13::3/128 [0/0]
     via TenGigabitEthernet1/1/3, receive
O   2001:DB8:0:14::/64 [110/2]
     via FE80::1, TenGigabitEthernet1/1/3
C   2001:DB8:0:23::/64 [0/0]
     via TenGigabitEthernet2/1/3, directly connected
L   2001:DB8:0:23::3/128 [0/0]
     via TenGigabitEthernet2/1/3, receive
O   2001:DB8:0:25::/64 [110/2]
     via FE80::2, TenGigabitEthernet2/1/3
OI  2001:DB8:1::1/128 [110/1]
     via FE80::1, TenGigabitEthernet1/1/3
OI  2001:DB8:1:1::/64 [110/2]
     via FE80::1, TenGigabitEthernet1/1/3
OI  2001:DB8:2::2/128 [110/1]
     via FE80::2, TenGigabitEthernet2/1/3
```

```
OI  2001:DB8:2:1::/64 [110/2]
     via FE80::2, TenGigabitEthernet2/1/3
LC  2001:DB8:3::3/128 [0/0]
     via Loopback0, receive
C   2001:DB8:3:1::/64 [0/0]
     via TenGigabitEthernet1/1/1, directly connected
L   2001:DB8:3:1::1/128 [0/0]
     via TenGigabitEthernet1/1/1, receive
OE1 2001:DB8:4:1::/64 [110/1001]
     via FE80::1, TenGigabitEthernet1/1/3
OE1 2001:DB8:5:1::/64 [110/1001]
     via FE80::2, TenGigabitEthernet2/1/3
L   FF00::/8 [0/0]
     via Null0, receive
```

3.6. MikroTik útválasztók konfigurálása

A MikroTik hálózati eszközök alkalmazása egyre népszerűbb hazánkban, így könyvünkből sem maradhatott ki az IPv6 konfigurálásának ismertetése RouterOS rendszerek esetében. A könyv készítésekor a rendszer 6.27-es verzióját vettük alapul, és a parancssori (CLI) működést ismertetjük.

3.6.1. Alapbeállítások

Az IPv6 protokoll használatához telepíteni (és engedélyezni) kell a szoftver verziónak megfelelő ipv6 csomagot. A csomag meglétét a `system package print` parancs segítségével ellenőrizhetjük. Az IPv6-os címek, melyek az egyes hálózati interfészekhez vannak rendelve, az `ipv6 address print` parancs segítségével kérdezhetőek le:

```
[admin@MikroTik] > ipv6 address print
Flags: X - disabled, I - invalid, D - dynamic, G - global, L - link-local
 #    ADDRESS                     FROM-POOL INTERFACE            ADVERTISE
 0 DL fe80::20c:42ff:fe3f:2e74/64           ether2               no
 1 DL fe80::20c:42ff:fe3f:2e73/64           ether1               no
 2 DL fe80::20c:42ff:fe3f:2e75/64           ether3               no
```

Minden olyan interface, mely aktív, tehát van hozzá csatlakoztatva hálózat, és nincs letiltva, rendelkezik link local címmel. IPv6-os cím hozzáadása az `ipv6 address add` paranccsal oldható meg:

```
[admin@MikroTik] > ipv6 address add interface=ether1 address=2001:DB8::1/64
```

Melynek eredménye:

```
[admin@MikroTik] > ipv6 address print
Flags: X - disabled, I - invalid, D - dynamic, G - global, L - link-local
 #    ADDRESS                     FROM-POOL INTERFACE            ADVERTISE
 0 DL fe80::20c:42ff:fe3f:2e74/64           ether2               no
 1 DL fe80::20c:42ff:fe3f:2e73/64           ether1               no
 2 DL fe80::20c:42ff:fe3f:2e75/64           ether3               no
 3  G 2001:db8::1/64                        ether1               yes
```

Beállíthatjuk módosított EUI-64 alkalmazásával létrehozott IPv6 cím használatát is az `eui-64=yes` paraméter megadásával, ez azonban útválasztók esetében nem célszerű.

Statikus route bejegyzés létrehozása az `ipv6 route add` paranccsal lehetséges. Példaként egy alapértelmezett átjáró hozzáadása:

```
[admin@MikroTik] > ipv6 route add dst-address=::/0 gateway=2001:db8::2
```

Míg a routing táblázat kiíratása:

```
[admin@MikroTik] > ipv6 route print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, o - ospf, b - bgp, U - unreachable
 #      DST-ADDRESS             GATEWAY                 DISTANCE
 0 A S  ::/0                    2001:db8::2                    1
 1 ADC  2001:db8::/64           ether1                         0
```

A RouterOS rendszerekben az IPv4 protokoll esetén alkalmazottal megegyezően, IPv6 protokoll esetén is a `ping` és a `tool traceroute` parancs használható a hálózati kapcsolatok ellenőrzésére. Fontos ugyanakkor, hogy nem hivatkozhatunk közvetlenül IPv6-os névre, csak IP-címekre. Például:

```
[admin@MikroTik] > ping ipv6.google.com
invalid value for argument address:
    invalid value of mac-address, mac address required
    invalid value for argument ipv6-address
    failure: dns name exists, but no appropriate record
```

De:

```
[admin@MikroTik] > ping 2a00:1450:4016:803::1000 count=4
  SEQ HOST                                     SIZE TTL TIME  STATUS
    0 2a00:1450:4016:803::1000                   56  49 26ms  echo reply
    1 2a00:1450:4016:803::1000                   56  49 26ms  echo reply
    2 2a00:1450:4016:803::1000                   56  49 26ms  echo reply
    3 2a00:1450:4016:803::1000                   56  49 26ms  echo reply
    sent=4 received=4 packet-loss=0% min-rtt=26ms avg-rtt=26ms max-rtt=26ms
```

A gyártó által javasolt megoldás a következő:

```
[admin@MikroTik] > ping [:resolve ipv6.google.com] count=4
  SEQ HOST                                     SIZE TTL TIME  STATUS
    0 2a00:1450:4016:803::1000                   56  49 27ms  echo reply
    1 2a00:1450:4016:803::1000                   56  49 26ms  echo reply
    2 2a00:1450:4016:803::1000                   56  49 26ms  echo reply
    3 2a00:1450:4016:803::1000                   56  49 26ms  echo reply
    sent=4 received=4 packet-loss=0% min-rtt=26ms avg-rtt=26ms max-rtt=27ms
```

3.6.2. Az ipv6 firewall

Az `ipv6 firewall` használata nagyon hasonló az `ip firewall` alkalmazásához, így példák segítségével csak az alapvető parancsokat ismertetjük.

Az `add` parancs segítségével egy meglévő lánc végéhez fűzhetünk újabb sort. Törölni a `remove` paranccsal lehet, míg a `print` parancs segítségével listázhatjuk ki a láncot.

Egy sor beszúrása a lánc végére, mely minden csomagot eldob, ami a 2001:db8::100 címről érkezik:

```
[admin@MikroTik] > ipv6 firewall filter add chain=input \
 src-address=2001:db8::100 action=drop
```

Kiírja a láncok tartalmát:

```
[admin@MikroTik] > ipv6 firewall filter print
Flags: X - disabled, I - invalid, D - dynamic
 0    chain=input action=drop src-address=2001:db8::100/128 log=no \
 log-prefix=""
```

Törli a megjelenített sort:

```
[admin@MikroTik] > ipv6 firewall filter remove numbers=0
```

3.6.3. Automatikus IPv6 címkiosztás

Router Advertisement (RA)

A MikroTik eszközökön a /ipv6 nd alatt állíthatjuk be a RA központi beállításait. Első lépésként irassuk ki a beállításokat:

```
[admin@MikroTik] > ipv6 nd print
Flags: X - disabled, I - invalid, * - default
 0  * interface=all ra-interval=3m20s-10m ra-delay=3s mtu=unspecified
      reachable-time=unspecified retransmit-interval=unspecified \
        ra-lifetime=30m
      hop-limit=unspecified advertise-mac-address=yes advertise-dns=yes
      managed-address-configuration=no other-configuration=no
```

Az interfészenként beállítást az IPv6 cím hozzáadásakor tehetjük meg. Például:

```
[admin@MikroTik] > ipv6 address add interface=ether1 \
  address=2001:db8::1/64 advertise=yes
```

Új cím hozzáadásakor a hirdetés alapértelmezetten engedélyezett (`advertise=yes`), így ezt nem szükséges megadni. Az RA-val hirdetett prefixek közé automatikusan felkerülnak a hozzáadott globális címek prefixei, de manuálisan is vihetünk fel hirdetendő prefixeket. A prefixek kíratása a következő módon lehetésges:

```
[admin@MikroTik] > ipv6 nd prefix print
Flags: X - disabled, I - invalid, D - dynamic
 0  D prefix=2001:db8::/64 interface=ether1 on-link=yes autonomous=yes
valid-lifetime=4w2d preferred-lifetime=1w
```

DHCPD

Első lépésben az IPv6 poolt kell létrehozni, melyből a prefixek delegálásra kerülnek:

```
/ipv6 pool add name=IPv6PDPool prefix=2001:db8::/56 prefix-length=60
```

A megadott konfigurációval a 2001:db8::/56 címtartományból kerülnek delegálásra /60 méretű prefixek. Hozzuk létre a DHCP szervert:

```
/ipv6 dhcp-server add name=dhcpserver1 interface=ether10 \
  address-pool=IPv6PDPool disabled=no
```

Érdekes megoldás, hogy egy interfésznek automatikusan biztosíthatunk egy megadott poolból IPv6 címet a következő parancs segítségével:

```
/ipv6 address add interface=ether1 add address=::1/64 from-pool=IPv6PDPool
```

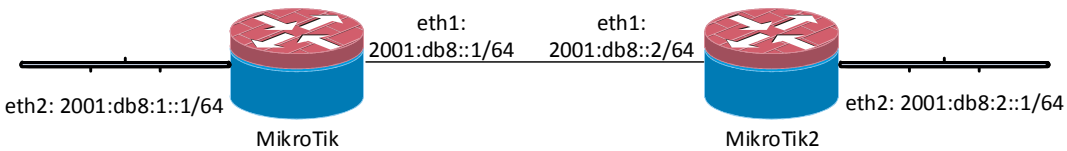
Ez lehet egy általunk beállított pool, de akár egy prefix delegációval kapott pool is, így a kapott prefixekből automatikusan rendelhetünk IPv6 címet a router megfelelő interfészéhez, majd onnan RA segítségével biztosíthatjuk a megfelelő prefixxel rendelkező címeket a kliensek részére.

3.6.4. OSPFv3

RouterOS rendszerekben az OSPFv3 beállítása a `routing ospf-v3 instance` parancs segítségével történhet. Első lépésként írassuk ki az alapbeállításokat:

```
[admin@MikroTik] > routing ospf-v3 instance print
Flags: X - disabled, * - default
 0  * name="default" router-id=0.0.0.0 distribute-default=never
redistribute-connected=no redistribute-static=no redistribute-rip=no
redistribute-bgp=no
      redistribute-other-ospf=no metric-default=1 metric-connected=20
metric-static=20 metric-rip=20 metric-bgp=auto metric-other-ospf=auto
```

Jól látszik, hogy egy IPv6 OSPF példány található az útválasztón, default néven, mely nem hirdeti tovább a többi routing protokollal kapott, valamint a statikus és közvetlenül csatlakozó hálózatokat. A következőkben állítsuk be az OSPFv3-at a 23. ábrán látható hálózathoz, majd ellenőrizzük működését. Az egyszerűség kedvéért csak backbone area van a hálózatban.



23. ábra: OSPF példahálózat MikroTik eszközökkel

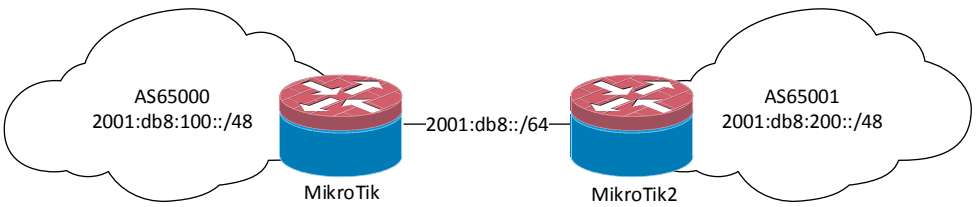
Ahogy már korábban említettük, az IPv4-nél alkalmazott OSPFv2 esetében `network` megadása szükséges az OSPF engedélyezéséhez, az OSPFv3 esetében az interfészeket kell konfigurálni. Az OSPFv3 működéséhez minden a hálózatban lévő útválasztónak egyedi azonosítóval kell rendelkeznie. (Ugyanúgy, mint az OSPFv2 esetében.) Ez OSPFv3 esetében is egy négy byte hosszú azonosító maradt, amit érdemes manuálisan beállítani. Első lépésben engedélyezzük a csatlakozó és a statikus hálózatok redisztribúcióját, állítsuk be a router-id-t, majd engedélyezzük a másik útválasztóhoz csatlakozó interfészen az OSPFv3-at:

```
[admin@MikroTik] > routing ospf-v3 instance set 0 \
 redistribute-static=as-type-1 redistribute-connected=as-type-1
[admin@MikroTik] > routing ospf-v3 interface add instance-id=0 \
interface=ether1 area=backbone
[admin@MikroTik2] > routing ospf-v3 instance set 0 \
 redistribute-static=as-type-1 redistribute-connected=as-type-1
[admin@MikroTik2] > routing ospf-v3 interface add instance-id=0 \
interface=ether1 area=backbone
```

Ellenőrizzük le a működést:

```
[admin@MikroTik] > routing ospf-v3 neighbor print
 0 instance=default router-id=2.2.2.2 address=fe80::20c:42ff:fe42:d0c9
   interface=ether1 priority=1 dr=2.2.2.2 backup-dr=1.1.1.1 state="Full"
   state-changes=6 ls-retransmits=0 ls-requests=0 db-summaries=0
   adjacency=3m39s
[admin@MikroTik] > ipv6 route print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, o - ospf, b - bgp, U - unreachable
 #      DST-ADDRESS              GATEWAY                   DISTANCE
 0 ADC  2001:db8::/64            ether1                           0
 1 ADC  2001:db8:1::/64          ether2                           0
 2 ADo  2001:db8:2::/64          fe80::20c:42ff:fe42:d...       110
[admin@MikroTik2] > routing ospf-v3 neighbor print
 0 instance=default router-id=1.1.1.1 address=fe80::20c:42ff:fe3f:2e73
   interface=ether1 priority=1 dr=2.2.2.2 backup-dr=1.1.1.1 state="Full"
   state-changes=5 ls-retransmits=0 ls-requests=0 db-summaries=0
   adjacency=2m16s
[admin@MikroTik2] > ipv6 route print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, o - ospf, b - bgp, U - unreachable
 #      DST-ADDRESS              GATEWAY                   DISTANCE
 0 ADC  2001:db8::/64            ether1                           0
 1 ADo  2001:db8:1::/64          fe80::20c:42ff:fe3f:2...       110
 2 ADC  2001:db8:2::/64          ether2                           0
```

3.6.5. IPv6 BGP



24. ábra: MP-BGP példahálózat MikroTik eszközökkel

Első lépésként írassuk ki a BGP alapbeállításait:

```
[admin@MikroTik] > routing bgp instance print
Flags: * - default, X - disabled
 0 *  name="default" as=65530 router-id=0.0.0.0 redistribute-connected=no
redistribute-static=no redistribute-rip=no
      redistribute-ospf=no redistribute-other-bgp=no out-filter="" client-
to-client-reflection=yes ignore-as-path-len=no
      routing-table=""
```

Állítsuk be az útválasztó saját AS számát:

```
[admin@MikroTik] > routing bgp instance set 0 as=65000

[admin@MikroTik2] > routing bgp instance set 0 as=65001
```

Állítsuk be a BGP-vel hirdetni kívánt hálózatot:

```
[admin@MikroTik] > routing bgp network add network=2001:db8:100::/48

[admin@MikroTik2] > routing bgp network add network=2001:db8:200::/48
```

Majd adjuk hozzá a BGP peer-t:

```
[admin@MikroTik] > routing bgp peer add name=MikroTik2-AS65001 \
 remote-address=2001:db8::2 remote-as=65001 update-source=ether1

[admin@MikroTik2] > routing bgp peer add name=MikroTik-AS65000 \
remote-address=2001:db8::1 remote-as=65000 update-source=ether1
```

Ellenőrizzük a működést:

```
[admin@MikroTik] > routing bgp peer print detail
Flags: X - disabled, E - established
 0 E name="MikroTik2-AS65001" instance=default remote-address=2001:db8::2
     remote-as=65001 tcp-md5-key="" nexthop-choice=default multihop=no
     route-reflect=no hold-time=3m ttl=255 in-filter="" out-filter=""
     address-families=ip,ipv6 update-source=ether1 default-originate=never
     remove-private-as=no as-override=no passive=no use-bfd=no
```

Az `E` betű jelzi, hogy létrejött a kapcsolat a másik útválasztóval. Többek közt leolvasható a távoli útválasztó IP címe, a távoli AS száma, az authentikáció során használt osztott kulcs és az is látszik, hogy nem használunk szűrőlistát. Készítsünk egy szűrőlistát, mely megakadályozza, hogy default route hirdetést fogadjunk el a szomszédtól, valamint az elfogadott prefixek esetében beállítja a weight értékét 200-ra:

```
[admin@MikroTik] > routing filter add action=discard \
 chain=bgpv6-AS65001-IN prefix=::/0 prefix-length=0 protocol=bgp
[admin@MikroTik] > routing filter add action=accept \
 chain=bgpv6-AS65001-IN protocol=bgp set-bgp-weight=200
```

Rendeljük hozzá a peer-hez a szűrőlistát:

```
[admin@MikroTik] > routing bgp peer set 0 in-filter=bgpv6-AS65001-IN
```

Ellenőrizzük a szűrőlista tartalmát:

```
[admin@MikroTik] > routing filter print
Flags: X - disabled
 0   chain=bgpv6-AS65001-IN prefix=::/0 prefix-length=0 protocol=bgp
invert-match=no action=discard set-bgp-prepend-path=""

 1   chain=bgpv6-AS65001-IN protocol=bgp invert-match=no action=accept set-
bgp-weight=200 set-bgp-prepend-path=""
```

Szűrőlistát eltávolítani a `routing filter remove` parancs segítségével tudunk. A művelet előtt ne felejtsük el a lista használatát megszüntetni az adott peer esetében sem!

3.6.6. IPv6 PPPOE szerver beállítása

A következőkben a PPPOE szerver beállítását ismertetjük annak ellenére, hogy hosszas próbálkozás ellenére sem sikerült megbízható működésre bírni a MikroTik implementációját IPv6 protokollal. Több szoftververzió kipróbálása után, valószínűsíthető, hogy valamilyen programhiba okozza a problémát. Mivel az internetszolgáltatók előszeretettel használják hitelesítésre a PPPOE protokollt, így a probléma megoldása érdekében jeleztük a hibát a gyártónak.

Elsőként hozzuk létre a PPPOE kliensek számára delegálásra kerülő címtartomány(oka)t tartalmazó IPv6 poolt a következő parancs segítségével:

```
/ipv6 pool add name=IPv6PDPool prefix=2001:db8::/56 prefix-length=60
```

A megadott konfigurációval a 2001:db8::/56 címtartományból kerülnek delegálásra /60 méretű prefixek, ami azt jelenti, hogy egy-egy kliens 16 darab /64-es méretű alhálózatot használhat, és összesen 16 klienst tud kiszolgálni a router. Állítsuk be az általunk használt PPP profilban a létrehozott pool nevét:

```
/ppp profile set 0 remote-ipv6-prefix-pool=IPv6PDPool
```

Hozzuk létre a pppoe szervert:

```
/interface pppoe-server server add disabled=no interface=ether1 disabled=no
```

Végül pedig hozzuk létre a PPPOE belpéshez használható felhasználót:

```
/ppp secret add name=User1 password=P@ssw0rd1
```

3.6.7. IPv6 PPP kliens beállítása

A PPPOE kliens beállítása lényegesen egyszerűbb, és problémamentesen működött. Első lépésként hozzuk létre a PPPOE klienst:

```
/interface pppoe-client add add-default-route=yes interface=ether1
password=P@ssw0rd1 user=User1 disabled=no
```

Ezután csak a DHCP kliens hozzáadása szükséges:

```
/ipv6 dhcp-client add interface=pppoe-out1 pool-name=ppp-test \
  pool-prefix-length=64
```

Ha szeretnénk tudatni a PPPOE szerverrel, hogy /62 méretű prefix delegálását kérjük, akkor a sor végére még írjuk oda, hogy: `prefix-hint=::/62`

A kiadott parancs eredményeként, ha a PPPOE szerver DHCP szervere delegál a kliens részére prefixet, akkor az megjelenik az IPv6 poolok között:

```
[admin@MikroTik] > ipv6 pool print
Flags: D - dynamic
 #   NAME        PREFIX                                  PREFIX-LENGTH EXPIRES-AFTER
 0 D ppp-test    2001:db8::10:0:0:0:0/60                            64 2d23h59m56s
```

Látható, hogy a szerver /60-as méretű prefixet delegál (ahogy a szerveren beállítottuk), melyet a kliensen futó DHCP szerver tovább delegálhat /64-es darabokban.

4. IPv6 áttérési technológiák összehasonlító elemzése

4.1. Általános áttekintés

4.1.1. Az IPv4-ről IPv6-ra való áttérés időbeli lezajlása

Az Internet történelmében egyszer sikerült a pillanatszerű protokollváltás: az ARPANET-en 1983. 01. 01-én volt az áttérés NCP-ről (Network Control Program) TCP/IP-re (RFC 801). Ma ez lehetetlen feladat. Több milliárd csomópont van, lehetetlen őket egyszerre „átkapcsolni". Sőt jelenleg rengeteg hardver és szoftver eleve alkalmatlan IPv6-ra.

Bár az átmenet régen megkezdődött, először nagyon lassan haladt, és csak az IPv4 címek kifogyása adott neki lendületet. A két protokoll tartósan egymás mellett fog élni a fentieken túl azért is, mert egyrészt bizonyos hardver és szoftver szállítók nem fogják megoldani az IPv6 kompatibilitást, másrészt a felhasználók ragaszkodnak a régi eszközeikhez. Tehát meg kell oldani az IPv4 és az IPv6 alapú rendszerek együttműködését!

4.1.2. Az IPv4 és IPv6 alapú rendszerek együttműködésének fontosabb esetei

A legtöbb alkalmazásunk kliens-szerver konfigurációban működik. Egyik szempont, hogy mely protokollokra képes a kliens és a szerver. A másik szempont, hogy a hálózat mely protokollokra képes a klienstől a szerverig terjedő úton. Ezen szempontok szerint vizsgálva a következő eseteket és megoldásokat tartjuk említésre érdemesnek:

Az egyszerű eset: dual stack használata

Ha a kliens és a szerver közül bármelyik is képes mindkét protokoll használatára (*dual stack*), akkor a „közös nyelv" használatával a kommunikáció megoldott – feltéve, hogy a köztük levő hálózat is támogatja azt. A probléma az, hogy már nincs elég IPv4 cím! (Kényszermegoldás: *Dual-Stack Lite*.)

IPv6 képes kliens IPv4-only környezetben és IPv6 szerver

A kliens képes IPv6-ra, de az internetszolgáltató (ISP) csak IPv4-címet ad a kliensnek, a szerver pedig kizárólag IPv6-ra képes. Egy régi és viszonylag jó megoldás: *6to4* használata. Ezt a megoldást mélyebben is bemutatjuk. A 6to4-hez hasonló megoldás még a *teredo* és a *6rd*, röviden ezekkel is foglalkozunk.

IPv6 kliens és IPv4 szerver

Csak IPv6-ra képes a kliens (már csak IPv6 cím jutott neki) és csak IPv4-re képes a szerver (régi, az IPv6-ot nem támogatja). Egy jó megoldás: *DNS64* szolgáltatás + *NAT64* átjáró használata. Mindkét részével mélyebben foglalkozunk.

IPv4 kliens és IPv6 szerver

Csak IPv4-re képes a kliens (régi hardver és/vagy szoftver) és csak IPv6-ra képes a szerver (ilyenek is vannak, és számuk várhatóan nőni fog). Egy megoldás lehetne (vagy inkább lehetett volna): *NAT46+DNS46*, de évek óta nem lett belőle szabvány, bár léteznek rá implementációk. Röviden ismertetjük a megoldások alapötletét.

IPv6 kliens és IPv6 szerver DE útközben egy szakaszon csak IPv4 van

Tipikus eset, az új IPv6 „szigeteket" össze kell kötni. A megoldás az IPv6 datagramok szállítása IPv4 fölötti „alagútban" (6in4 tunnel). Bemutatunk egy olyan megoldást is (MPT [6]), amely bármelyik IP verzió (4 vagy 6) felett bármelyik verziót képes átvinni.

IPv4 kliens és IPv4 szerver DE útközben egy szakaszon csak IPv6 van

Ma még nálunk nem igazán jellemző, de majd lehet. A megoldás az IPv4 datagramok szállítása IPv6 fölötti „alagútban" (4in6 tunnel). És a fent említett MPT is használható.

IPv4 kliensek privát IPv4 címmel és IPv4 szerver DE az ISP hálózatában csak IPv6 van

A klienseknek már nem jutott publikus IPv4 cím, de ahelyett, hogy IPv6-ot használnának, 4-es verziójú privát IP-címeket adnak nekik. Megoldás lehet a MAP protokoll, röviden tárgyaljuk.

4.2. Emlékeztető az IP-címek megosztásáról

Ennek a témakörnek (legalább részleges) ismeretét ugyan feltételezzük az Olvasóról, de a továbbiak megértéséhez szükséges néhány fontos részlet bemutatása és az egységes terminológia használata érdekében egy rövid összefoglalót adunk róla.

4.2.1. A címfordítást használó megoldás

A hálózati címfordítás (NAT/NAPT) működési elve

A TCP/IP protokollcsaládot eredetileg végponttól végpontig való kommunikációra tervezték. Az alkalmazások arra számítanak, hogy a címek és portszámok a hálózati átvitel során változatlanok.

Az IPv4-címek szűkössége miatt terjedt el az a megoldás, hogy egy hálózatban, ahol privát IP-címeket használnak, de szükség van külső kommunikációra is, a problémát címfordítással oldják meg.

Legyen a feladat először az, hogy a *privát IP-címmel rendelkező (kliens) gépek elérhessenek külső, publikus IP-címmel rendelkező (szerver) gépeket*. Ehhez rendelkezésünkre áll egy router, aminek van publikus IP-címe.

A megoldás alapötlete a következő:

- A privát IP-címmel rendelkező gépről a célcím alapján küldjük el a csomagot az Internet felé. (Ekkor a csomag elvileg ugyan odaérne, de a válasz nem találna vissza. Egyébként pedig tiltott a privát IP-címeknek a publikus környezetben történő használata.)
- A kimenő router cserélje ki a forrás privát IP-címét a saját publikus IP-címére. Így már a válasz visszaér a címcserét végrehajtó routerhez.
- A router továbbítsa a választ a megfelelő privát IP-című gépnek. – De hogyan?

Ahhoz, hogy a router a választ az eredeti feladónak vissza tudja küldeni, nyilván kell tartania, és egy válaszcsomag érkezésekor tudnia kell, hogy ki volt a küldője annak a csomagnak, amire a válasz érkezett. Ennek érdekében a nyilvántartáshoz az IP-címen túl mást is felhasznál. TCP és UDP esetén ezek a forrás portszámok. De a routernek nem elegendő ezeket megjegyeznie, hiszen a forrás portszámok csak *gépenként egyediek*, a forrás IP-címet viszont a sajátjára cseréli. Tehát a megoldás (az ún. *traditional NAT* esetén, de lásd majd később: *extended NAT*) a következő:

- A router a kimenő csomagokban a forrás IP-cím cseréjekor a forrás portszámokat is kicseréli a routeren egyedi portszámokra: az IP-címek és a célportszám mellett ezekkel már egyértelműen azonosítani tudja a kapcsolatokat. Kapcsolatonként nyilvántartja, hogy mit mire cserélt ki.
- A bejövő csomagoknál az IP-címen kívül a portszámot is vissza kell cserélnie. (Itt most az irány változása miatt a cél IP-cím és a célport az, amit átír.)

Megjegyzés: terminológia nem egységes, de eredetileg a NAT csak az IP-címek cseréjét jelentette, ezt hívják ma *basic NAT*-nak, vagy *one-to-one NAT*-nak. A fenti megoldás precíz neve a *NAPT* (Network Address and Port Translation). Nevezik *many-to-one NAT*-nak is.

A NAT célja és működése szempontjából a fent ismertetett megoldást *Source NAT*-nak (SNAT) hívjuk akkor, ha a router publikus IP-címe fix, és *Masquerade*-nek, ha DHCP-vel kapta az interfésze

az IP-címet (ilyenkor nem egy fixen megadható IP-címre, hanem az interfész aktuális IP-címére kell a forrás IP-címet kicserélni).

A másik irányú feladat az, hogy *privát IP-címmel rendelkező gépeket elérhetővé tegyünk az Internet felől.* Erre a megoldás a *Destination NAT* (DNAT) vagy más néven *port forwarding*, ahol a router az adott portjára érkező csomagokat egy meghatározott privát IP-című gépnek továbbítja úgy, hogy a célcímet kicseréli a csomagban. Például a 80-as portra érkező csomagokat a 10.1.1.2 webszerver, a 25-ösre érkezőket pedig a 10.1.1.3 SMTP szerver felé továbbítja.

ICMP esetén nincs portszám, de segíthet az, hogy egy hibaüzenetben benne van az azt kiváltó TCP vagy UDP adategység első 64 bitje a portszámokkal. Amennyiben nem hibaüzenetről van szó, akkor is van megoldás: az ICMP üzenet valamilyen azonosító jellegű mezőjét használják fel.

A NAT-ról bővebben az RFC 3022-ben olvashatunk, az IP, TCP, UDP, ICMP fejrészek mezőinek módosításával a 4.1. rész, az ellenőrző összeg hatékony újraszámításával a 4.2. rész foglalkozik. Lényegében arról van szó, hogy a régi portszámot "kivonjuk", az újat pedig "hozzáadjuk" és nem kell a számítást a teljes adategységre elvégezni.

A NAPT kritikája

A megoldás súlyos problémája, hogy sérti az IP-címek és porszámok végponttól végpontig történő változatlanságának elvét, ami számos következménnyel jár. Vizsgáljunk meg közülük néhányat:

- Koncepcionálisan az összes olyan alkalmazási rétegbeli protokollal probléma lehet, ahol az alkalmazási rétegbeli PDU-ban megjelenik valamilyen kommunikációazonosító (pl. IP-cím, portszám). Például egy FTP kliens aktív módban a vezérlő kapcsolaton keresztül megadja a szervernek, hogy mely portján várja, hogy a szerver felépítse az adatkapcsolatot. Privát IP címmel várhatja – hacsak nem segít valaki: *protocol helper*. (Az adott esetben persze tökéletes megoldás az FTP passzív módjának alkalmazása, amikor a vezérlő kapcsolaton keresztül a szerver küldi el a kliensnek az IP-cím + portszám párost, ahova a kliens gond nélkül fel tudja építeni az adatkapcsolatot. Ez a megoldás a kliens oldali tűzfal problémáját is megoldja, ami kifele engedni fogja a kapcsolat felépítését, míg befele nem engedné.) Hasonlóképpen egy IP telefon rendszernél is gond lesz az IP-címek átküldésével (arra is van megoldás, létezik például SIP NAT helper is).
- A port forwarding ugyan alkalmas arra, hogy egyetlen publikus IP-cím használatával több szervert tegyünk elérhetővé, de ettől még a privát IP-címmel rendelkező gépek használhatósága nem teljes értékű, hiszen elérhetőségük nem automatikus, a NAPT eszközön beállításra van szükség, sőt egy adott portot csak egy eszköz fele lehet továbbítani. (Tehát ha két privát IP-címmel rendelkező, azonos publikus IP-cím mögé helyezett előfizető szeretne azonos porton szolgáltatni – például mindkettő web szervert szeretne üzemeltetni –, az már problémát okoz.)
- A közös publikus IP-címen való osztozás további problémája, hogy a kommunikációban részt vevő felek nem tudják egymást kölcsönösen azonosítani. Például egy web szerver nem a NAPT mögötti kliensek IP-címét látja (privát IP címük egyébként teljesen haszontalan is lenne), hanem az NAPT eszköz külső interfészének publikus IP-címét. Visszaélések megállapítása (lásd következő pont) vagy például szavazások (egy IP-címről naponta egy szavazat) esetén ez komoly gondot okoz.
- Internetszolgáltatók esetén fontos jogszabályi előírás, hogy képeseknek kell lenniük utólag adatot szolgáltatni arról, hogy adott időpontban egy adott IP-címet mely előfizetőjük használt. Többletköltség árán műszakilag természetesen megoldható, hogy az IP-címeket a portszámokkal együtt naplózzák, de még ekkor is problémát okozhat, ha olyan megkeresést kapnak, hogy csak időpont és IP-cím alapján kell azonosítaniuk az előfizetőt (mert a támadást vagy visszaélést bejelentő fél nem naplózta a kliens portszámát, csak IP-címét).

A fenti problémák majd NAT64-nél is előjönnek, ott az alkalmazások kompatibilitására részletesen ki fogunk térni.

NAPT mélyebben: hagyományos és kiterjesztett típusok

A fent bemutatott NAPT megoldás előnye az egyszerűség, aminek ára a portszámok pazarló használata. Már 2007-ben javasoltak olyan megoldást, ami a portszámokkal sokkal takarékosabban bánik [7]. Nézzünk bele most egy kicsit mélyebben mindkét megoldás működésébe!

Két számítógép között egyszerre több kommunikáció is folyamatban lehet, akár TCP akár UDP használatával. Bár az UDP kapcsolatmentes protokoll, nevezzük most ezeket mégis *kapcsolat*nak (connection vagy session) mindkét esetben. Egy kapcsolatot a következő 5 szám azonosít egyértelműen: forrás IP-cím, forrás portszám, cél IP-cím, cél portszám, protokoll (TCP vagy UDP).

Egy *hagyományos* (traditional) típusú NAPT implementáció a *kapcsolatok követésé*re (connection tracking) csak a 7. táblázat szerinti azonosítókat használja, tehát kihagyja a (privát IP címmel rendelkező kliens szempontjából) cél IP-címet és cél portszámot. Minden forrás IP-cím + forrás port páros esetén egyedire cseréli a kimenő csomagban a forrás portszámot. Mivel a portszámok 16 bitesek, és erre a célra csak az 1024-től 65535-ig terjedő tartományt használja, ezért összesen 63k darab TCP és ugyanennyi UDP kapcsolatot képes kezelni a kimenő interfészére felhúzott minden egyes publikus IP-cím mellett. Ez jelentős korlátozás, ezért erre a problémára már 2007-ben javasolták a következő, ma elterjedten alkalmazott megoldást.

Egy *kiterjesztett* (extended) típusú NAPT implementáció a kapcsolatok követésére a 8. táblázat szerinti azonosítókat használja, tehát a cél IP-címet és cél portszámot is. Csak akkor cseréli ki a csomagban a forrás portszámot, ha az feltétlenül szükséges, azaz a csere nélkül nem lenne a kapcsolat egyértelműen azonosítható a visszaérkező csomagban szereplő értékek alapján. A 8. táblázatban több olyan esetet is bemutatunk, amikor a forrás portszámot nem kell cserélni, a csere csak az utolsó sorban volt szükséges. Ezzel a megoldással lényegesen sikerült kibővíteni a kimenő interfészre felhúzott publikus IP-címenként kezelhető kapcsolatok számát.

Source IP Address	Source Port Number	External IP Address	Temp. Port Number	Transport Protocol
10.1.2.2	5001	192.0.2.1	10001	TCP
10.1.2.3	5001	192.0.2.1	10002	TCP
10.1.3.5	5002	192.0.2.1.	10003	TCP

7. táblázat: Hagyományos NAPT kapcsolat követési táblázat (translation table) [8]

Source IP Address	Source Port Number	External IP Address	Temp. Port Number	Destination IP Address	Dest. Port Number	Transport Protocol
10.1.2.2	5001	192.0.2.1	5001	198.51.100.2	80	TCP
10.1.2.3	5001	192.0.2.1	5001	203.0.113.3	80	TCP
10.1.3.5	5001	192.0.2.1.	5001	198.51.100.2	443	TCP
10.1.3.6	5001	192.0.2.1.	10001	198.51.100.2	80	TCP

8. táblázat: Kiterjesztett NAPT kapcsolat követési táblázat (translation table) [8]

Megjegyezzük azonban, hogy a kezelhető kapcsolatok száma így sem végtelen, ezzel később részletesen foglalkozunk.

A cél IP-cím és cél portszám bevonásának természetesen ára van, a kapcsolattáblában lényegesen nagyobb méretű kulcs alapján kell keresni! (A gyakorlatban hash függvény segítségével oldják meg a gyors keresést.)

Az extended megoldást használja például a Linux *Netfilter* keretrendszere [9] is, amit a felhasználói interfészének neve után *iptables*nek is szoktak nevezni.

4.2.2. Az Address plus Port (A+P) megoldás

Az A+P működési elve

Ez a megoldás is az IPv4 címek szűkösségének problémáját hivatott enyhíteni. Alapötlete, hogy az útválasztásba vonjuk be az IP-címeken túl a portokat is. Egy szolgáltató ugyanazt az IP-címet több előfizetőnek is kiosztja, és ezzel együtt az azonos IP-címmel rendelkező előfizetőknek diszjunkt és megfelelő méretű portszám tartományokat is kioszt. Ezenkívül mindegyik előfizetőnél olyan eszközt helyez el, ami az adott előfizető forrás portszám használatát a neki kiosztott portszám tartományra korlátozza. A szolgáltató ezután saját hálózatában az útválasztási algoritmust úgy módosítja, hogy az A+P megoldásba bevont címek esetén az adott címre érkező datagramok a cél portszám alapján a megfelelő előfizetőhöz érkezzenek meg. Nézzük, meg, hogyan működik ez a megoldás!

Az előfizető kliense természetesen nem foglalkozik azzal, hogy A+P mögött van, tehát tetszőleges forrás portot választ. A szolgáltató által kihelyezett eszköz egy NAT44 segítségével átírja a forrás portot az előfizető számára kiosztott tartományból valamelyik szabad portra. A datagram gond nélkül eljut a címzetthez, majd a válasz visszaér szolgáltatóhoz. A szolgáltató speciális útválasztása a cél portszám alapján az azonos IP-című előfizetők közül a megfelelőhöz továbbítja a datagramot, majd a kihelyezett NAT44 eszköz átírja a portszámot és így a datagram eljut az illetékes alkalmazáshoz.

A megoldásról bővebben a (kísérletinek nevezett, 2011-ben kiadott) RFC 6346-ban olvashatunk.

Az A+P kritikája

A megváltoztatott útválasztást támogató eszköz még költséghatékonyan elhelyezhető a szolgáltatónál, de a minden előfizetőhöz kihelyezendő eszköz már érdemben drágítja a szolgáltatást. Az előfizetőknek kiosztandó diszjunkt portszám tartományok követelménye még súlyosabb probléma, ugyanis ha túl kicsire választják, akkor az korlátozza az előfizetőt[9], így csökkenti a szolgáltatás értékét, ha pedig túl nagyra választják, az viszont tönkreteszi a megoldás hatékonyságát. Például előfizetőnként 1024 portot számítva 64 előfizető[10] osztozhat egy publikus IP-címen. Ezért a megoldás jóval kevésbé hatékony, mint a NAPT, ahol nem kell maximumra méretezve fixen kiosztani az egyes előfizetőknek a portszám tartományokat, hanem a traditional esetben is statisztikai multiplexelés van, az extended hatékonysága pedig még annál is sokkal jobb. Az A+P előnye viszont a speciális útválasztást végző eszköz állapotmentessége: vele szemben egy extended NAPT eszköznek igen nagyszámú kapcsolat kezelésére kell képesnek lennie!

A NAPT-nál említett említett további problémák természetesen itt is fennállnak.

4.3. A DNS64 + NAT64 megoldás

4.3.1. A megoldás elvi működése

Az IPv4 címek kifogyása miatt az IP 4-es és 6-os verziójának együttműködésében várhatóan az lesz az első tipikus megoldandó probléma, hogy az internetszolgáltatók csak IPv6 címet tudnak

[9] Lásd később az egyes alkalmazások portszám fogyasztását. Illetve egy előfizető (és családtagjai) egyidejűleg több eszköz használatára is igényt tarthatnak.

[10] Vagy 63, ha a 0-1023 tartományt nem osztjuk ki.

adni az új ügyfeleiknek, és a *csak IPv6 címmel rendelkező* (IPv6-only) klienseknek el kell érniük a *csak IPv4 címmel rendelkező* (IPv4-only) szervereket is.

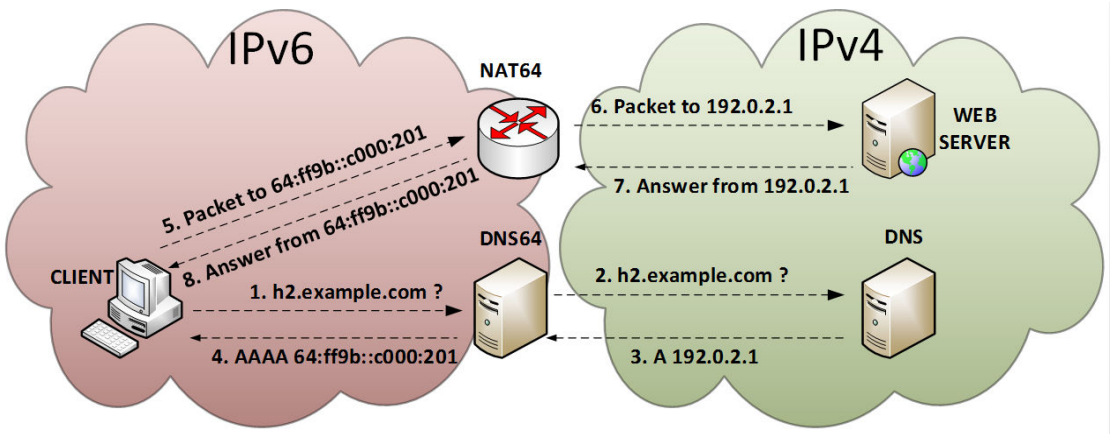
A DNS64+NAT64 megoldás használhatóságának egyik szükséges feltétele, hogy a csatlakozni kívánó fél (kliens) a DNS rendszert használja az elérni kívánt fél (szerver) IP-címének kiderítésére, és a kliensben névkiszolgálóként egy DNS64 szerver legyen beállítva.

A *DNS64* szerver (RFC 6147) egy *caching-only* névkiszolgálóhoz hasonlóan *recursive query*kre válaszol. Ennek során, egy adott *szimbolikus név*hez (domain name):

- ha van IPv6 cím, akkor továbbítja a válaszában a kliensnek
- ha nincs IPv6 cím, de IPv4 cím van, akkor az IPv4 cím alapján generál egy speciális IPv6 címet, és azt adja vissza a kliensnek.

A speciális IPv6 cím egy *IPv4 címet beágyazó IPv6 cím* (IPv4-Embedded IPv6 Address) ami a legegyszerűbb esetben, az ún. *NAT64 Well-Known Prefix* használata esetén, a 64:ff9b::/96 prefix + az utolsó 32 biten a szimbolikus névhez tartozó IPv4 cím.

A megoldás működéséhez továbbá szükséges, hogy az útválasztási táblázatok szerint a NAT64 Well-Known Prefix (mint hálózat) felé az út egy *NAT64* átjárón (RFC 6146) keresztül vezessen (anycast címzés használható).



25. ábra: DNS64+NAT64 megoldás elvi működése [10]

Az elvi megoldás működését a 25. ábra példáján mutatjuk be. Kövessük végig a példát. Az IPv6-only kliens csatlakozni szeretne az IPv4-only **`h2.example.com`** nevű webszerverhez, ennek érdekében:

1. A kliens lekéri a szerver IPv6 címét a szimbolikus neve alapján.
2. A DNS64 szerver a DNS rendszertől megkérdezi a **`h2.example.com`** névhez tartozó IP-címet.[11]
3. Válaszként csak egy IPv4-címet (192.0.2.1) kap („A" record).
4. A DNS64 szerver a benne beállított NAT64 Well-Known prefix (64:ff9b::/96) használatával előállítja a szükséges IPv4 címet beágyazó IPv6 címet (64:ff9b::c000:201) és ezt elküldi a kliensnek („AAAA" record).

[11] A valóságban a DNS64 szerver két külön kérést küld: először egy „AAAA" rekord kérést, és amennyiben erre üres választ kap, akkor egy „A" rekord kérést [11]. Az RFC 6417 5.1.8. pontja azt is megengedi, hogy a két kérdést párhuzamosan küldje el (a második küldése előtt nem várja meg az első válaszát).

5. A kliens TCP SYN szegmenst tartalmazó IPv6 csomagot küld a kapott IPv4 címet beágyazó IPv6 címre (64:ff9b::c000:201), amely a routing beállítása miatt a NAT64 átjáróhoz érkezik meg.
6. A NAT64 átjáró az IPv6 csomag alapján egy IPv4 csomagot készít; benne a célcím a speciális IPv6 célcím utolsó 32 bitje (192.0.2.1), a forráscím pedig a NAT64 átjáró IPv4 címe lesz. Közben a NAT64 átjáró eltárolja a kapcsolattáblájába a szükséges információkat, és ha korszerű az implementáció, akkor a forrás portszámot csak szükség esetén cseréli. Az átjáró az elkészült IPv4 csomagot elküldi a címzettnek, ami az IPv4-only szerver.
7. Az IPv4-only szerver megkapja a TCP SYN szegmenst tartalmazó IPv4 csomagot és a megszokott módon válaszol (SYN+ACK). Mivel a kapott IPv4 csomagban a forráscím a NAT64 átjáró IPv4 címe volt, a válasz címzettje is a NAT64 átjáró IPv4 interfésze lesz, a forráscím pedig a saját IPv4 címe. A válasz megérkezik az NAT64 átjáróhoz.
8. A válasz alapján a NAT64 átjáró elkészíti a neki megfelelő IPv6 csomagot, melybe forráscímként ugyanaz a speciális IPv6 cím kerül, amit a DNS64 szerver generált, célcímként az IPv6-only kliens IPv6 címe kerül; mindez a kapcsolattábla alapján történik. Az IPv6 csomagot a NAT64 átjáró elküldi az IPv6-only kliensnek.

Amikor az IPv6-only kliens megkapja a csomagot, úgy folytatja a kommunikációt, hogy közben mit sem sejt arról, hogy a szervernek IPv4 címe van. A klienshez hasonlóan a szerver sem észlel semmi különöset, hiszen látszólag a NAT64 átjáró IPv4 interfészével kommunikál. Ennek persze az a következménye, hogy a szerver az IPv6-only kliens valódi IPv6 címe helyett a NAT64 átjáró IPv4 címét fogja naplózni. Ez egyben a megoldás egyik gyengesége is, de természetesen nem csupán a NAT64-é, hanem a privát IP-címek használatakor alkalmazott NAT-é is, mint azt már említettük. (Számos esetben okoz súlyos problémát, például naplózásnál, IP-cím alapján végzett szavazásnál, vagy kitiltásnál, stb.)

Megjegyzés: A NAT64-től való megkülönböztetésül a NAT-ot NAT44-nek is hívják, amikor mind a lecserélt, mind az új IP-cím verziószáma 4-es.

A fentiekben bemutatott megoldás az RFC 6052 szerinti 64:ff9b::/96 előre lefoglalt, ún. NAT64 Well-Known Prefixet használja. A prefix használatának számos korlátja van, lásd RFC 6052 3. rész. A NAT64 átjáró megvalósításakor a gyakorlatban sok esetben az egyes IPv6 hálózatokból szoktak erre a célra lefoglalni egy részt, ezt hálózat-specifikus prefixnek (Network-Specific Prefix) nevezik.

4.3.2. Az IPv4-címeket beágyazó IPv6-címekről

Az *IPv4-címeket beágyazó IPv6-cím*eket (IPv4-Embedded IPv6 Address) még két további névvel illetik a felhasználásuk céljától függően, bár szerkezetük és előállításuk azonos.

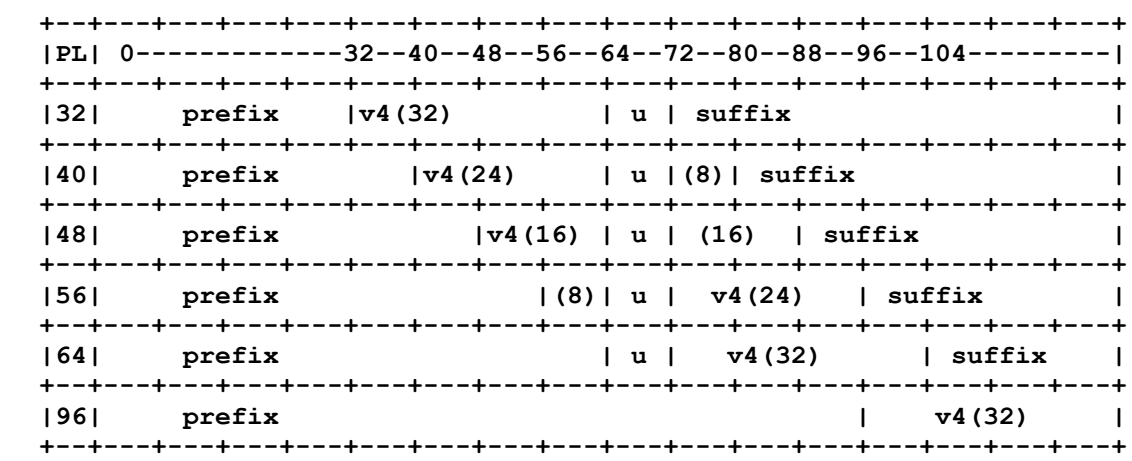
- Azokat az IPv6 címeket, amiket arra használunk, hogy IPv4 állomásokat képviseljenek IPv6 hálózatokban, úgy nevezzük, hogy *IPv4-ből konverált IPv6 cím*ek (IPv4-Converted IPv6 Address). A fenti példában pontosan erről volt szó.
- Az *IPv4-re lefordítható IPv6 cím* (IPv4-Translatable IPv6 Address) megnevezést pedig akkor használjuk, ha a cím egy IPv6 állomáshoz tartozik, és a fordítás célja, hogy a csak IPv4-re képes eszközök is el tudják érni az IPv6 állomást. (Ezzel az esettel most nem foglalkozunk.)

Az RFC 6052 definiálja, hogy hogyan kell az IPv4-címeket beágyazó IPv6-címeket képezni. Hálózat-specifikus prefixnél a hálózat adminisztrátora dönti el, hogy a rendelkezésére álló címtartományból mekkora tartományt szeretne erre a célra szánni. Ennek megválasztása során tekintettel kell lennie az alábbi szabályokra is:

- A prefix mérete szigorúan csak 32, 40, 48, 56, 64 vagy 96 lehet.
- Az IPv6 címben a 64-71. sorszámú biteknek 0 értékűeknek kell lenniük.
- Az IPv4 cím 32 bitjét a választott méretű prefix után írjuk, de a fenti követelmény kielégítése érdekében a szigorúan 0 értékű bitek helyét „átugorjuk”.

- A cím végét szükség esetén ugyancsak 0 értékű bitekkel töltjük ki.

A címek lehetséges formátuma:



Ha például egy /48-as címtartományunk van, akkor a hálózat-specifikus prefix mérete /56, /64 vagy /96 lehet. A választás szempontjairól az RFC 6052 3.3. és 3.4. részében olvashatunk.

4.3.3. A megoldás élettartama

IPv6 áttérési megoldásról lévén szó, az nyilvánvaló, hogy akkor már nem lesz szükség rá, amikor minden eszköz képes lesz az IPv6-ra. Azonban ehhez várhatóan még akár egy vagy több évtizedre is szükség lehet. A megvalósításhoz használt eszközök forgalma kezdetben várhatóan növekedni fog a csak IPv6-ra képes kliensek számának növekedésével, aztán várhatóan csökkenésnek indul, amint egyre nagyobb arányban képesek lesznek a szerverek IPv6-ra. És a forgalom hosszú idő alatt válik elenyészővé. Olyan technológiának ítéljük meg, amit érdemes mind megismerni, mind kutatni, mert legalább egy évtizedig szükség lesz rá.

4.4. A MAP megoldás és változatai

A kiindulási probléma hasonló, mint a DNS64+NAT64 megoldásnál: IPv4 címek kifogyása miatt az internetszolgáltatók már nem tudnak 4-es verziójú publikus IP-címet adni a klienseknek, a szolgáltató hálózatában csak IPv6 működik, de a klienseknek el kell érniük a csak IPv4 címmel rendelkező szervereket. A megközelítés viszont más: a kliensek nem IPv6 címet kapnak, hanem privát IPv4 címet. A MAP változatától függően a szolgáltató IPv6 hálózatában vagy *beágyazással* (encapsulation, MAP-E) vagy *állapotmentes címfordítással* (translation, MAP-T) kerül átvitelre a datagram. A szolgáltató IPv6 hálózata egy Border Relay eszközzel kapcsolódik a publikus IPv4 hálózathoz.

Bár ez is egy lehetséges megoldás a publikus IPv4 címek kifogyásának a problémájára, de ez a megközelítés inkább konzerválja a helyzetet: a klienseket az IPv4 világban tartja. Ezzel szemben a NAT64+DNS64 megoldás előremutató, a kliensek alapvetően az IPv6 világba kerülnek, de lehetőségük nyílik a csak IPv4 címmel rendelkező szerverek elérésére is.

A részletek tárgyalása nélkül megjegyezzük, hogy a megoldás rokonságban van az A+P módszerrel, de ügyes trükkel a portszámot az IPv6 címben tárolja, így az IPv6 hálózatban szabályos, IPv6 cím alapú útválasztást végezhet. Az A+P-vel való rokonságot tükrözi a két változatának teljes neve

is: *Mapping of Address and Port with Encapsulation* (MAP-E, RFC 7597), illetve *Mapping of Address and Port using Translation* (MAP-T, RFC 7599).

Mivel a megoldást nem tartjuk előremutatónak, bővebben nem foglalkozunk vele.

4.5. A DNS46+NAT46 megoldás

A megoldandó probléma éppen a fordítottja, mint a DNS64+NAT64 esetén: itt a csak IPv4 címmel rendelkező kliensek számára szeretnénk biztosítani a csak IPv6 címmel rendelkező szerverek elérését. Sajnos esetünkben a fordított probléma megoldása nem használható egy az egyben. Míg a teljes IPv4 címtartomány könnyen leképezhető az IPv6 címtartomány egy kis részére (egy IPv6 címben bőségesen elfér egy teljes IPv4 cím), a fordítottja nem igaz. Ezért dinamikus összerendelést (mapping) hoznak létre a két címtartomány egyes elemei között. Mivel a kliens kizárólag IPv4 címeket képes kezelni, a DNS46 megoldás az elérni kívánt szerver szimbolikus neve alapján kiderített IPv6 címhez egy IPv4 címet rendel, és azt adja vissza a kliensnek. A kliens az így kapott IPv4 címet célcímként felhasználva kapcsolódik a szerverhez. Úgy van beállítva a routing, hogy a DNS46 szerver által visszaadott *IPv6 címeket reprezentáló IPv4 címek* tartománya felé az átjáró egy NAT46 eszköz, ami ismeri a DNS64 szerver aktuális összerendeléseit (mapping), és elvégzi a címek cseréjét, valamint a protokoll konverziót. Ezt a megoldást leíró RFC sajnos nem került elfogadásra, az utolsó draft már 2010-ben lejárt [12]. Bár a megoldást nem szabványosították, mégis több implementációja létezik. Szabad szoftverként például létezik modul az OpenWRT-hez [13], és egyes nagy hálózati eszközgyártók termékei is támogatják, például a Cisco ASA 5510 tűzfal [14] vagy a Brocade ServerIron ADX [15].

Mivel a megoldás nem szabványos, mélyebben nem foglalkozunk vele, viszont kíváncsian várjuk, hogy az IETF milyen szabványos megoldást kínál a problémára.

4.6. A 6to4 megoldás

Ezt a megoldást akkor használjuk, ha egy IPv6 képes eszköz IPv4-only környezetben van, és IPv6 protokollal egy másik IPv6-os eszközt szeretne elérni (akár az is lehet IPv4-only környezetben).

A 6to4 megoldás (RFC 3056) egy „automatikus" tunnel, ami az IPv6 datagramokat IPv4 datagramokba csomagolja be (és természetesen ki is). A 6in4-hez hasonlóan a 41-es protokollazonosítót használja (lásd később).

4.6.1. A 6to4 címzése

Ahhoz, hogy egy eszköz IPv6 protokollt tudjon használni, szüksége van IPv6 címre. Mivel a 6to4 megoldást natív IPv6 Internet elérés hiányában használjuk, az IPv6 címet az IPv4 címből állítjuk elő, és a natív IPv6 címektől való megkölönböztetés érdekében *6to4 IPv6 cím*nek hívjuk.

A 6to4 címzéshez a 2002::/16 prefixet foglalták le. A 6to4 IPv6 címek képzése az alábbi módon történik.

- Hálózatcím: 2002::/16 prefix + publikus IPv4 cím 32 bitje + 16 bit subnet ID
    - Egyetlen host (lásd később) esetén a subnet ID egy generált véletlenszám.
    - Ha a 6to4 mechanizmust router használja, akkor akár több IPv6 hálózat is lehet mögötte, ekkor hasznos a subnet ID.
- Gépcím: A szabványos módosított EUI-64 azonosító

Ilyen módon a 6to4 megoldás minden publikus IPv4 címhez egy 2002::/16 kezdetű, /48 méretű IPv6 címtartományt rendel. Mindegyik IPv4 cím „mögött" elérhető lehet egy ilyen méretű IPv6 hálózat.

4.6.2. A 6to4 megoldás működése

A kommunikáció során a 6to4 IPv6 címmel rendelkező állomások datagramjainak IPv4 datagramokba való be- és kicsomagolását végző *6to4 psedo-interfész* (6to4 pseudo-interface) elhelyezése szempontjából kétféle konfiguráció lehetséges:

- lehet egyetlen host, amin az IPv6 kliens fut, és a becsomagolást is a host végzi – a 6to4 pseudo-interface a hoston van (*6to4 host*)
- lehet több IPv6-os gép egy IPv6 hálózaton, aminek a routere végzi a becsomagolást – a *6to4 (border) router* rendelkezik 6to4 pseudo-interface-szel.

A működés során ez a két eset nem különbözik lényegesen, az viszont igen (és ezért külön tárgyalást igényel), hogy az IPv4 környezetben levő IPv6 eszköz – nevezzük a továbbiakban *6to4 IPv6 állomás*nak – egy az IPv6 Internetre kapcsolódó *natív IPv6 állomás*sal vagy egy másik 6to4 IPv6 állomással kommunikál-e.

Tekintsük először egy 6to4 IPv6 állomás és egy natív IPv6 állomás kommunikációját. Ebben az esetben az IPv4 hálózat és a natív IPv6 Internet között egy további eszközre van szükség: ez a *6to4 relay*. A 6to4 relay feladata a 6to4 IPv6 állomástól érkező, IPv4 datagramba ágyazott IPv6 datagram kicsomagolása és továbbítása az IPv6 Interneten át a natív IPv6 állomás felé. A másik irányú kommunikáció során szintén a 6to4 relay feladata a natív IPv6 állomástól származó datagram IPv4 datagramba való becsomagolása és az IPv4 hálózaton keresztül való továbbítása. Ilyen eszközből több is lehet, az IPv4 irányából a legközelebbi a 192.88.99.1 anycast címen érhető el.

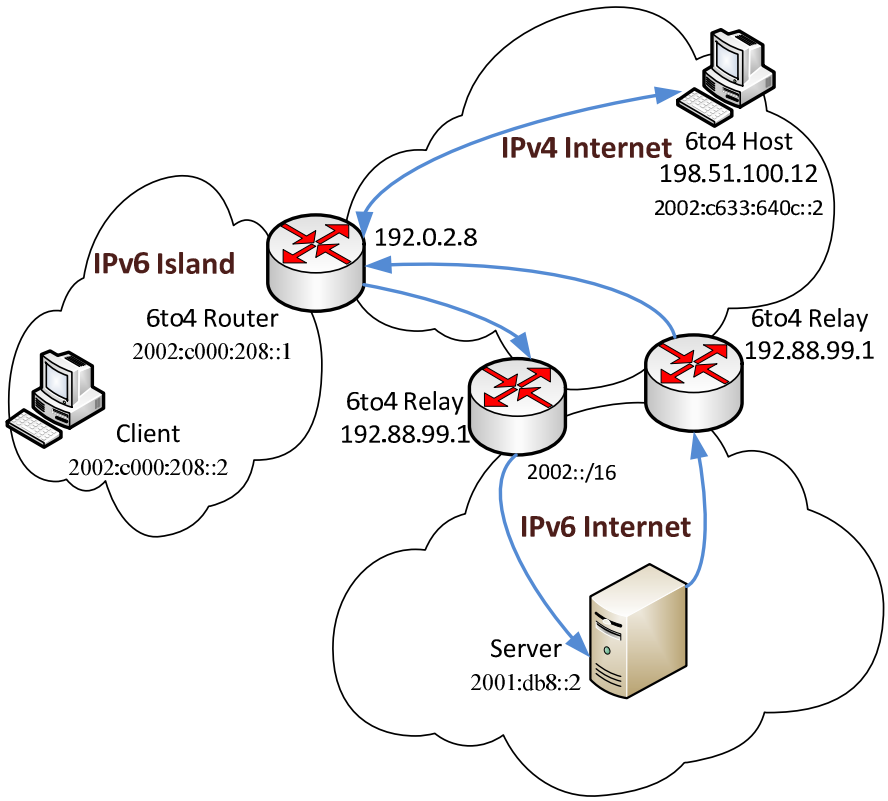
A 6to4 megoldás működését a 4. ábrán látható hálózat példáján mutatjuk be, ahol egy 6to4 IPv6 címmel rendelkező kliens kommunikál egy natív IPv6 szerverrel. A kommunikáció lépései:

- A kliens a szerver felé IPv6 csomagot küld (forráscím: 2002:c000:208::2, célcím: 2001:db8::2).
- A becsomagolás elvégzője (itt most a 2002:c000:208::1 címen elért 6to4 router, de lehetne maga a host is) az IPv6 csomagot IPv4 csomagba ágyazza, és az IPv4 segítségével elküldi egy 6to4 relaynek (forráscím: 192.0.2.8, célcím: 192.88.99.1).
- A beágyazást elvégző 6to4 routerhez az IPv4 hálózat metrikája szerint legközelebbi (192.88.991 anycast címre hallgató) 6to4 relay kapja meg a csomagot.
- A 6to4 relay kicsomagolja az IPv4-be ágyazott IPv6 csomagot, majd továbbítja a natív IPv6 hálózatba (forráscím: 2002:c000:208::2, célcím: 2001:db8::2).
- A natív IPv6 hálózatban a csomag megérkezik a címzetthez.
- A címzett válaszol (forráscím: 2001:db8::2, célcím: 2002:c000:208::2).
- A válasz a natív IPv6 állomáshoz az IPv6 Internet metrikája szerint legközelebbi, az IPv6 hálózatba 2002::/16 prefixet hirdető 6to4 relayhez érkezik meg. Ez lehet az előbbivel azonos vagy attól eltérő 6to4 relay (jelen példánkban eltérő). Erre a kommunikáló feleknek nincs befolyása!
- Visszafele a 6to4 relay az IPv6 csomagot IPv4-be csomagolva küldi a kliensnek, pontosabban a korábban a kliens IPv6 csomagját IPv4-be csomagoló eszköznek, ami példánkban a 192.0.2.8 IPv4 címre hallgató 6to4 router (forráscím: 192.88.99.1, célcím: 192.0.2.8). Megjegyzés: A 192.0.2.8 célcímet a 6to4 relay a 2002:c000:208::2 IPv6 célcímből tudta megállapítani.
- A válasz megérkezik a 192.0.2.8 IP-című 6to4 routerhez.
- A 6to4 router kicsomagolja az IPv4 csomagból az IPv6 csomagot és elküldi a kliensnek (forráscím: 2001:db8::2, célcím: 2002:c000:208::2).
- Végül a kliens megkapja a szerver válaszát.

A fenti példában a 6to4 állomás volt a kliens a natív IPv6 állomás pedig a szerver, de a fordított esetnek sincs akadálya, a kommunikáció a natív IPv6 irányából is kezdeményezhető.

A megoldás működéséhez szükséges feltétel, hogy a becsomagolást végző eszköznek (6to4 router vagy 6to4 host) legyen publikus IPv4-címe, különben nem tudna érvényes 6to4 IPv6 címet nyújtani a mögötte található eszközöknek. Ha a becsomagolást végző eszköznek nincs publikus IPv4 címe, akkor a 6to4 helyett *Teredo*t lehet használni (RFC 4380).

A másik eset, amikor két 6to4 állomás kommunikál egymással (két IPv6 szigetet kötünk össze a 6to4 segítségével az IPv4 Internet felhasználásával). Ebben az esetben a becsomagolást végző eszköz a datagram cél IPv6-címéből (2002::/16 prefix) tudja meg, hogy nem egy 6to4 relaynek, hanem a megfelelő IPv4 című pseudo-interfésszel rendelkező eszköznek (6to4 host vagy 6to4 router) kell IPv4 szinten címeznie és küldenie a csomagot. A 26. ábra példája segítségével egy ilyen esetet is nyomon követhetünk. Kommunikáljon most az előbbi példában szereplő kliens az ábra jobb felső sarkában található 6to4 állomással:



26. ábra: A 6to4 megoldás lehetőségei [16] (módosítva)

- A kliens a 6to4 állomás felé IPv6 csomagot küld (forráscím: 2002:c000:208::2, célcím: 2002:c633:640c::2).
- A becsomagolás elvégzője (a 2002:c000:208::1 című 6to4 router) az IPv6 csomagot IPv4 csomagba ágyazza. Ennek során az IPv6 csomagban szereplő célcím 2002::/16 prefixe alapján tudja, hogy a datagramot nem egy 6to4 relaynek kell címeznie, hanem a cél IPv4 cím a cél IPv6 címből az első 16 bitet követő 32 bit, azaz 198.51.100.12 lesz (forráscím: 192.0.2.8, célcím: 198.51.100.12).
- A csomag megérkezik a 198.51.100.12 IP-című 6to4 hosthoz.
- A 6to4 host pseudo interfésze elvégzi a kicsomagolást és továbbítja az IPv6 datagramot az IPv6 interfésznek (forráscím: 2002:c000:208::2, célcím: 2002:c633:640c::2).

- Az IPv6 címen elérhető alkalmazás válaszol (forráscím: 2002:c633:640c::2, célcím: 2002:c000:208::2).
- A választ a 6to4 host pseudo interfésze IPv4 csomagba ágyazza. A cél IPv4 címet szintén a cél IPv6 címből állapítja meg (forráscím: 198.51.100.12, célcím: 192.0.2.8).
- A válasz megérkezik a 192.0.2.8 IP-című 6to4 routerhez.
- A 6to4 router kicsomagolja az IPv4 csomagból az IPv6 csomagot és elküldi a kliensnek (forráscím: 2002:c633:640c::2, célcím: 2002:c000:208::2).
- Végül a kliens megkapja a 6to4 host válaszát.

Figyeljük meg, hogy ebben az esetben a kommunikációban mindkét irányban ugyanazok a 6to4 interfészek vettek részt. (Ez nem véletlen, hiszen a kommunikáló eszközök 6to4 IPv6 címében az IPv4 címek egyértelműen rögzítettek voltak.)

4.6.3. A 6to4 megoldás élettartama

A megoldás már régóta üzemel. A továbbiakban már csak viszonylag rövid ideig van/lesz rá szükség; addig, amíg lesz olyan terület, ahol az internetszolgáltatók nem nyújtanak IPv6 elérést. Bár az Amerikai Egyesült Államokban némelyek már (2011 óta) temetni szeretnék a megoldást[12], Magyarországon Budapest kivételével még nagyon is valóságos probléma, hogy valaki nem tud IPv6 internet előfizetést vásárolni.

4.6.4. A 6to4 megoldás problémái

Szolgáltatásminőség

A 6to4 megoldás által nyújtott szolgáltatás minőségére súlyos következményekkel jár az, hogy amennyiben egy 6to4 IPv6-ot használó állomás egy a natív IPv6-ot használó állomással kommunikál, akkor – amint láttuk – a két irányban nem feltételnül azonos útvonalon (és 6to4 relayen) halad a kommunikáció, és ezen ráadásul a 6to4 állomás internetszolgáltatója sem tud segíteni. Miért? Vizsgáljuk meg ezt a kérdést! Amennyiben a 6to4 állomás internetszolgálatója egyáltalán nem foglalkozik azzal, hogy ügyfeleli részére IPv6 elérést biztosítson, akkor a forgalom mindkét irányban valamilyen más szervezet által üzemeltetett 6to4 relayen halad keresztül. (A 6to4 router pedig az ügyfélnél található.) Ha az internetszolgáltató rendelkezik natív IPv6 Internet eléréssel, akkor üzemeltethet egy saját 6to4 relayt, amit az ügyfeleinek a 192.88.99.1 anycast címen meghirdetve elérheti, hogy azok kimenő 6to4 IPv6 forgalma rajta, mint számukra legközelebbi 6to4 relayen keresztül menjen át az IPv6 Internetre. Arra azonban nincs befolyása, hogy a vissszafele irányuló forgalom melyik 6to4 relayen menjen keresztül; egy adott ügyfél esetén pontosan azon a relayen fog átmenni, amely az IPv6 Interneten a legközebb van ahhoz a natív IPv6 állomáshoz, amellyel az ügyfél számítógépe éppen kommunikál. Ha az adott relay túlterhelt, akkor az negatívan befolyásolja a felhasználó által tapasztalt szolgáltatásminőséget. A probléma orvosolható a 6to4 megoldás egy „továbbfejlesztett" változata a 6rd (RFC 5969) használatával. (Ami azonban NEM helyettesíti a 6to4 megoldást!)

[12] Az erre irányuló draft RFC nyomon követhető: https://tools.ietf.org/html/draft-ietf-v6ops-6to4-to-historic. A 6-os verzióban még a teljes 6to4 megoldásról (RFC 3056) szó volt, 8-as verziótól már egyre inkább csak az RFC 3068 szerinti 192.88.99.0/24 anycast prefixről. Ezt végül 2015. májusában elfogadták, mint RFC 7526.

Biztonság

A 6to4 megoldás biztonsági kérdéseivel külön RFC foglalkozik (RFC 3964). A 6to4 megoldás természetéből adódóan érzékeny a szolgáltatásmegtagadás (DoS: Denial of Service) és a címhamisítás (address spoofing) típusú támadásokra. A legtöbb biztonsági problémát a 6to4 következő két jellemzője okozza:

1. Egy 6to4 routernek el kell fogadnia és ki kell bontania bármely más 6to4 routertől (6to4 hostot is beleértve) vagy relaytől származó forgalmat.
2. Egy 6to4 relaynek el kell fogadnia bármely natív IPv6 node forgalmát.

Az RFC-ben meghatároztak olyan szabályokat, hogy egy 6to4 routernek, illetve relaynek milyen ellenőrzéseket kellene végrehajtania. Például egy 6to4 router (a 6to4 hostot is beleértve) ne engedjen meg olyan forgalmat, aminek például:

1. IPv4 címe privát, broadcast vagy valamilyen lefoglalt tartományba tartozik
2. IPv4 forráscíme nem egyezik meg azzal, ami a 6to4 prefixében van
3. IPv6 címe nem globális IPv6 cím (hanem például link-lokális, stb.)
4. 6to4 prefixe nem egyezik meg a sajátjával.

Hasonlóan 6to4 relayre is vannak ilyenek. Például a fentiek közül az első három arra is vonatkozik. Kifejezetten 6to4 relayre vonatkozó tiltás, hogy ne fogadjon el 6to4 routertől olyan forgalmat, aminek az IPv6 célcímében 6to4 prefix szerepel (és nem natív IPv6 cím).

Sajnos ezeknek a szabályoknak a betartását a 6to4 implementációk nem mindig ellenőrzik (kényszerítik ki).

Az RFC számos támadási lehetőséget leír, amelyeket nem részletezünk, csak megjegyezzük, hogy a 6to4 sajnos sok lehetőséget ad a támadásra, ezért aki használni szeretné, annak mérlegelnie kell a kockázatokat. Az OpenBSD operációs rendszerben például biztonsági megfontolásból nem implementálták a 6to4 megoldást: http://www.securityfocus.com/columnists/459.

4.6.5. A 6to4 „továbbfejlesztései”

Teredo

A 6to4 megoldás használatát korlátozza, hogy használatához a 6to4 pseudo-interface számára publikus IPv4 cím szükséges, de ma rengeteg felhasználó privát IP-című gépet és valamilyen NAT megoldást használ. A *Teredo*t (RFC 4380) kifejezetten NAT-on keresztüli működésre tervezték. Ezt úgy éri el, hogy a Teredo kliensek a jól ismert (kliensbe beépített) címeken (3544-es UDP port) elérhető Teredo szerverektől megszerzett információ alapján IPv4 fölött UDP használatával építenek ki alagutat a megfelelő *Teredo relay* felé, amelynek natív IPv6 kapcsolata van. A Teredo a 2001::/32 prefixű IP címeket használja, ahol a következő 32 biten a Teredo szerver IPv4 címe található, majd a kommunikáció egyéb paraméterei, így a NAT típusa, és speciális kódolással a NAT eszköz publikus IPv4 címe, valamint az azon használt UDP portszám is.

Mélyebben nem foglalkozunk vele, mivel egyrészt csak *végső megoldás*nak (last resort) szánják arra az esetre, ha sem natív IPv6, sem 6to4 nem használható, másrészt pedig mostanában tervezik lekapcsolni a Teredo relayeket[13].

6rd

Szintén a 6to4 negatív tulajdonságainak kiküszöbölésére született a 6rd (RFC 5969) megoldás. A 6rd fő előnyének szánták, hogy csak az adott szolgáltató hálózatán belül és kontrollja alatt működik, és itt szállítja IPv4 fölött az előfizetők IPv6 forgalmát. Ennek megfelelően nem a 2002::/16 prefixet,

[13] A javaslat a következő dokumentum 8. oldalán: http://www.ietf.org/proceedings/88/slides/slides-88-v6ops-0.pdf

hanem a szolgáltató saját IPv6 prefixét használja. Ez azért előnyös, mert míg a 6to4 esetén kiszámíthatatlan, hogy melyik 6to4 relayen keresztül megy át egy csomag a natív IPv6 hálózatból az IPv4 hálózatba, itt a szolgáltató prefixe ezt egyértelműen meghatározza. Mindez garantált elérhetőséget és szolgáltatás minőséget biztosít az előfizetőknek. Amennyiben tehát egy szolgáltató szeretne a segítségével „majdnem igazi" IPv6 internet előfizetést nyújtani, akkor arra a célra kiválóan alkalmazható. Viszont a 6rd-t csak a szolgáltató tudja a hálózatában bevezetni, a szolgáltató nélkül a megoldás használhatatlan, tehát a 6rd NEM képes a 6to4 megoldást helyettesíteni.

4.7. A 6in4 megoldás

Ezt a megoldást akkor használjuk, ha IPv6 szigetek csak IPv4 hálózaton keresztül tudnak egymással kommunikálni (IPv6-over-IPv4, lásd: RFC 4213).

Ekkor az IPv6 csomagokat IPv4 csomagokba ágyazva visszük át az IPv4 hálózaton. Az IPv4-ben a 41-es protokollazonosítót használjuk az IPv6 csomagok azonosítására. (Amint az IP fölött a Protocol mezőben a TCP protokollt a 6, az UDP-t a 17, az ICMP-t pedig az 1 érték azonosítja.) A be- és kicsomagolást az IPv6 szigetek határán levő átjárók végzik.

A megoldás nagyon hasonlít a 6to4 technikára, azzal a különbséggel, hogy az alkalmazott tunnelek kizárólag manuális módon hozhatóak létre. Ennek a módszernek legnagyobb előnye a biztonság, hiszen csak előre beállított tunneleket alkalmaz, amelyek konfigurálását mindkét végponton el kell végezni. Ez ugyanakkor hátránya is, hiszen együttműködést és munkát igényel mindkét féltől. Problémát okoz még, ha nincs mindkét oldalnak állandó publikus IPv4 címe, hiszen ekkor minden címváltozáskor módosítani kell a tunnel másik végpontján lévő eszköz beállításait. A módszer ennek ellenére elterjedt; a tunnel brókerek is ezt a megoldást alkalmazzák, és általában valamilyen kiegészítő szoftver segítségével oldják meg a beállítások (tunnel végpont címek) automatikus módosítását.

4.8. Az MPT tunnel

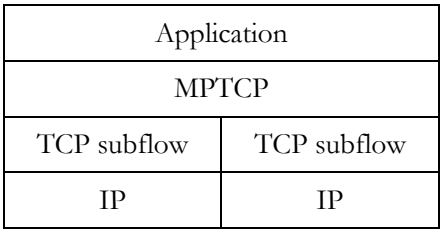
Az MPT [6] *hálózati szintű többutas kommunikációs könyvtár*at (network layer multipath communication library) a Debreceni Egyetemen fejlesztették ki Dr. Almási Béla vezetésével. Segítségével bármely verziójú IP (4 vagy 6) fölött bármely verziójú IP alagút (tunnel) létrehozható. Képességei ennél lényegesen bővebbek, segítségével lehetőség van több út átviteli kapacitásának összegzésére [17]-[18] vagy vezeték nélküli hálózatokban roaming megoldására [19], akár eltérő hálózati technológiák estén is [20]. Mi most, mint az eltérő IP verziószám problémájának megoldására használható alagútképző megoldással foglalkozunk vele.

Az MPT hálózati szintű többutas kommunikációs könyvtár alapvető újdonságát a sokkal közismertebb Multipath TCP-vel (RFC 6824) való összehasonlítás segítségével mutatjuk be.

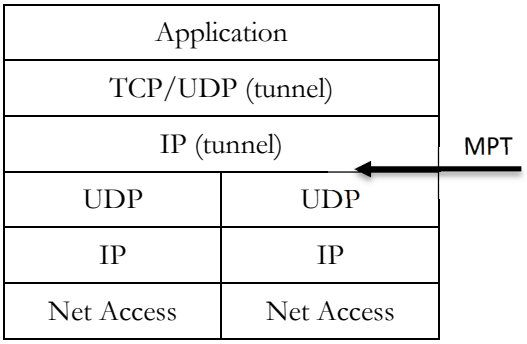
Az MPTCP több TCP kapcsolatot (TCP subflow) használ potenciálisan eltérő utak felett (27. ábra), ami egy jó megoldás az eltérő átviteli utak kapacitásának összegzésére. A TCP által nyújtott megbízható bájtfolyam típusú átvitel tökéletes megoldás a hálózati alkalmazások széles osztálya számára, mint például web böngészés, elektronikus levelek küldése vagy letöltése, fájlok átvitele. Viszont kifejezetten nem kívánatos egy másik osztály számára, mint például az IP-telefónia, a videokonferencia vagy más olyan valós idejű alkalmazások, amelyek jobban tolerálják a (kismértékű) csomagvesztést, mint a nagy késeltetéseket vagy késleltetés-ingadozásokat, amelyeket a TCP újraküldési mechanizmusa okoz.

Az MPT hálózati szintű többutas kommunikációs könyvtár UDP/IP-t használ minden link szintű kapcsolat felett, és ezek felett IP alagutat hoz létre. A két szint IP verziója egymástól függetlenül megválasztható. Az IP alagút fölött akár TCP, akár UDP használható (28. ábra). Ezért az újraküldés kihagyható, ha nincs rá szükség. Ez az architektúra az MPT-t az MPTCP-nél általánosabbá teszi,

más felhasználási lehetőségekkel. Ezenkívül az MPT-t kiválóan alkalmassá teszi az IPv6-ra való áttérés során (többféle) alagútképző megoldásként való használatra az a tény, hogy a két szinten egymástól függetlenül megválasztható az IP verziója.



27. ábra: Az MPTCP protokollverem (protocol stack) architektúrája (RFC 6824)



28. ábra: Az MPT könyvtár által megvalósított architektúra [17]

Az MPT legújabb verziójában protokollváltás történt. A szerzők követik a GRE-in-UDP RFC draft [21] specifikációjában leírtakat. A koncepcionális működés ugyanaz maradt, de egy GRE fejrész került az IP tunnel és UDP fejrész közé, ahogy azt a GRE-in-UDP megköveteli [22].

4.9. További ajánlott források

Az IPv6 áttéréssel számos publikáció foglalkozik. Ajánlunk közülük néhányat. Az IPv4/IPv6 együttélésről, átjárásról, migrációról és trendekről szóló könyvfejezet: [23]. Hasonló téma, de a 3G UMTS hálózatokra, kifejezetten mobil környezetre fókuszálva: [24]. Az IPv6 anycast áttekintését tartalmazza, és egy speciális használati lehetőségéről ír a mobilitás-kezelés területén: [25]. Az IPv6 áttéréssel kapcsolatos trendek, statisztikák vizsgálatával foglalkozik: [26].

Az IPv4 címek megosztására használható algoritmusokat osztályozza [27], melynek során értinti azt fontos szempont is, hogy az egyes megoldások elősegítik-e az IPv6 mielőbbi bevezetését.

5. IPv6 áttérési technológiák vizsgálata

5.1. Általános áttekintés

5.1.1. Vizsgálati szempontok

Az alkalmazandó IPv6 áttérési technológiákat általában az egyes szervezetek (pl. internetszolgáltatók) hálózati rendszergazdái választják ki a megoldandó feladatokhoz. Némely probléma megoldására több technológia is létezik, ilyenkor dönteni kell az adott célra legmegfelelőbb technológiáról. Ezen túl a bemutatott áttérési technológiák többségéhez számos implementáció létezik, melyek közül szintén választani kell. Ezekben a döntésekben szeretnénk gyakorlati segítséget nyújtani a témában végzett kutatási eredményeink bemutatásával.

Ha egy technológiának vagy annak egy implementációjának az éles szolgatói környezetben való üzemszerű alkalmazhatóságát vizsgáljuk, akkor a vizsgálatnak legalább a következő négy szempontra kell kiterjednie:

- A megoldás kompatibilis-e az alkalmazásokkal, befolyásolja-e azok működését, használhatóságát?
- A megoldás stabilan viselkedik-e, nem fog-e összeomlani?
- A megoldás gazdaságos-e? Milyen a teljesítménye / erőforrás fogyasztása más megoldásokhoz képest?
- A megoldás biztonságos-e? Milyen sebezhetőségei vannak, milyen kockázatokat hordoz az alkalmazása?

Ezek a kérdések mind az egyes technológiák, mind azok implementációinak szintjén felmerülnek. Messze nem tudunk rájuk a teljesség igényével választ adni. Kutatásaink során igyekeztünk elsősorban azokat a technológiákat megvizsgálni, amelyekre várhatóan hamarosan szükség lesz. Ezek között is erősen válogatnunk kellett az erőforrásaink (anyagi, humán, idő) korlátos volta miatt.

Kutatásainkat jelenleg is folytatjuk, eredményeinket folyamatosan publikáljuk. Cikkeink általában már a bírálati fázisban elérhetők itt: http://www.hit.bme.hu/~lencse/publications/.

5.1.2. Kiválasztási szempontok

Csak *szabad szoftver* (free software) [28] (más szóhasználatban *nyílt forrású* (open source) [29]) implementációkkal foglalkoztunk. Erre több okunk is volt [16]:

- Bizonyos cégek licencei (például: Cisco [30] és Juniper [31]) nem engedélyezik *teljesítőképesség-vizsgálat*ok (benchmarking) eredményeinek publikálását.
- Szabad szoftvereket bárki, bármilyen célra használhat, így eredményeink a lehető legszélesebb kör számára hasznosak.
- A szabad szoftverek egyben ingyenesek is, ami számunka is fontos szempont volt.

A továbbiakban problémakörönként bemutatjuk azokat a szempontokat, amelyek a vizsgálandó technológiák és implementációk kiválasztásában vezettek bennünket.

IPv6 kliens esete az IPv4 szerverrel

A publikus IPv4 címtartomány kimerülése miatt az első megoldandó gyakorlati probléma az, hogy a klienseknek már nem jut IPv4 cím, ezért kénytelenek az IPv6 protokollt használni, de a szerverek nagy része még mindig csak az IPv4 protokoll segítségével érhető el. Erre a problémára számos megoldást találtak ki, melyek közül a legfontosabbnak az alábbiakat találtuk [32]:

- A NAT-PT/NAPT-PT (RFC 2766) megoldást 2000-ben még szabványnak javasolták (proposed standard státuszú volt az RFC), de számos ok miatt 2007-ben végképp elvetették (RFC 4966).
- *Alkalmazásszintű átjáró* (ALG: Application Level Gateway, lásd: RFC 2663) vagy más néven proxy használata működőképes alternatíva lehet, de nagyon költséges, mert minden alkalmazáshoz külön ki kell fejleszteni, és üzemeltetni kell az IPv6 és IPv4 hálózatok határán.
- A legáltalánosabb és flexibilisebb megoldásnak az DNS64 szerver és NAT64 átjáró alkalmazását tartjuk.

Ezért a DNS64 és NAT64 megoldások vizsgálatával részletesen foglalkoztunk.

IPv6-ra képes kliens IPv4 környezetben

Magyarországon ma még gyakran előforduló probléma, hogy szeretne valaki IPv6-ot használni, de az internetszolgáltatója még nem képes azt neki nyújtani. Erre a problémára a 6to4 és a Teredo megoldások léteznek. Közülük a 6to4 vizsgálatával foglalkoztunk.

Adott IP verziójú szigetek összekötése a másik verzió felett

Előbb az IPv6 szigetek IPv4 fölötti összekötése, majd az IPv4 szigetek IPv6 fölötti összekötése lesz várhatóan a gyakoribb feladat. Ezek mindegyikére alkalmas az MPT könyvtár. Ennek a teljesítőképességét is megvizsgáltuk.

5.1.3. Vizsgálati szempontok technológiánként

NAT64

A NAT64 technológiánál az eddig elvégzett vizsgálataink az alábbiakra terjedtek ki:

- A *technológia* mely alkalmazásokkal kompatibilis és melyekkel nem?
- A *technológiának* milyen a portszám fogyasztása az egyes alkalmazásoknál?
- Az *egyes implementációknak* milyen a stabilitása és teljesítménye?

Összesen két implementációt vizsgáltunk, mert amikor a méréseket végeztük, akkor azt a két szabad szoftver implementációt találtuk vizsgálatra érdemesnek. Más (részben később megjelent) implementációk vizsgálatát is tervezzük a jövőben.

DNS64

Az egyes DNS64 implementációkat elsősorban stabilitás és teljesítőképesség szempontjából vizsgáltuk háromféle operációs render (Linux, OpenBSD és FreeBSD) alatt. Az egyik implementáció hibája miatt érintettünk biztonsági kérdéseket is.

6to4

Az egyes 6to4 implementációknál is teljesítőképességet és stabilitást vizsgáltunk. Itt szintén vizsgáltuk az operációs rendszerek hatását is, bár itt az operációs rendszer gyakran erősen behatárolta a használható implementációkat.

MPT könyvtár

Az MPT könyvtár teljesítményét abból a szempontból vizsgáltuk, hogy milyen hatékonyan képes több átviteli út kapacitását összegezni. Mind a hordozó, mind a tunnel protokoll szerepében mindkét IP verzió működését teszteltük (összesen 4 forgatókönyv).

5.2. A NAT64 technológia kompatibilitása

A témával kapcsolatos kutatási eredményeinket [33]-ben közöltük, most ezeket foglaljuk össze. Először áttekintjük az alkalmazások kompatibilitásával kapcsolatos publikációk eredményeit, majd bemutatjuk az általunk kiválasztott NAT64 implementációkat, a tesztelendő alkalmazásokat, a tesztelési módszert, végül közöljük és értelmezzük az eredményeket.

5.2.1. Más kutatók eredményei

Škoberne és Ciglarič [34] 18 hálózati alkalmazás NAT64 kompatibilitást tesztelte elméletben és gyakorlatban virtualizált környezetben az Ecdysis [35] NAT64 implementáció segítségével. Az eredményt három kategóriába sorolták: jó, feltételes és rossz (well translated, conditionally translated, poorly translated). Megállapították, hogy az átlagos internet-felhasználók által naponta használt alkalmazások többségénél (konkrétan: HTTP, HTTPS, IMAP, NTP, POP3, RDP, CIFS, SMTP, SSH, TELNET) az eredmény a jó kategóriába esik. Az FTP és BitTorrent protokollokat a feltételes, a Skype, MSNP, SIP, OpenVPN, IPsec, PPTP protokollokat pedig a rossz kategóriába sorolták. Arra a következetésre jutottak, hogy az otthoni felhasználók szempontjából a VoIP és az üzenetküldő (Instant Messaging) alkalmazások inkompatibilitása rontja a felhasználói élményt. Vállalati környezetben viszont a legtöbb üzleti alkalmazás megfelelően fog működni.

Bajpai és szerzőtársai [36] egyes hálózati alkalmazások különféle implementációit tesztelték a Dual-Stack Lite és a NAT64 megoldásokkal való kompatibilitás szempontjából. A tesztelt web böngészők: Safari 5, Google Chrome 10, Firefox 4, Opera 11, amelyekkel a következő vizsgálatokat végezték el: WebMail (Gmail TLSv1 használatával), média letöltés (YouTube Flash és HTML5 használatával), Google Maps, HTTP letöltés, Web Chat (Gmail, Yahoo, Freenode IRC). Levelezésre az Apple Mail 4-es verzióját használták és a következő protokollokat tesztelték: IMAP (Gmail és Microsoft Exchange), POP3 (Gmail) és SMTP (Gmail és Microsoft Exchange). Üzenetküldés (Instant Messaging) és IP fölötti telefonálás vizsgálatára a következőket tesztelték: iChat, Skype, SIP. További tesztelt protokollok: SSH, FTP, IRC, Git, Mercurial, OpenVPN, BitTorrent. Azt találták, hogy közülük a Skype, az OpenVPN, a BitTorrent és a SIP volt a NAT64-gyel inkompatibilis (a Dual Stack Lite esetén pedig egyáltalán nem észleltek problémát). Az inkompatibilitás okát is megvizsgálták. A Skype kliens egyáltalán nem támogatja az IPv6 protokollt. A Transmission nevű BitTorrent kliens sikeresen letöltötte a torrent fájlt a trackerről, de nem tudott csatlakozni olyan peerhez, ami csak IPv6 címmel rendelkező gépen futott. A cikk az OpenVPN-ről csak annyit állapít meg, hogy nem tudott IPv6 fölött „VPN csomagokat" átvinni, konkrét részleteket nem közöl. Részletesen leírják viszont a Linphone nevű SIP klienssel végzett vizsgálatot, ami IPv6 támogatással rendelkezik. Ebből annyit tartunk fontosnak megemlíteni, hogy a SIP jelzésátvitel még sikeresen lezajlott, de az RTP a médiafolyam nem indult el. Ennek okaként azt adták meg, hogy az SDP rekordokban a végpontokat IPv4 címek azonosítják. Az FTP-vel kapcsolatban nem jeleztek problémát, de sajnos azt nem közölték, hogy aktív vagy passzív módot használtak-e a vizsgálat során.

Megjegyezzük, hogy az OpenVPN esetén mindkét cikk inkompatibilitást állapított meg.

Egy másik cikkben [37] a Skype 5.0 verzióját NAT64 kompatibilisnek találták, amit mi a korábbi cikkek és saját tapasztalataink alapján egyaránt megkérdőjelezünk, ezért további eredményeiket nem közöljük. Az említett konferenciacikk kiadója (IARIA) szerepel az ún. *parazita* (predator) kiadók és folyóiratok nemzetközi listáján [38].

Végezetül megemlítjük, hogy az RFC 6586 pedig olyan gyakorlati tapasztalatokról számol be, amikor néhány felhasználó egy tisztán IPv6 hálózatot használt, és az IPv4 Internetet NAT64-en keresztül érték el. Számos érdekes eredményt közöl, például 12 üzenetküldő alkalmazás közül 6 működött 6 pedig nem, 12 internetes játék közül pedig csak egyetlen egy működött, ami web alapú volt. Fő következtetése, hogy a tisztán IPv6 alapú hálózat használható, de vannak problémák.

5.2.2. A vizsgálatainkhoz használt NAT64 implementációk

A NAT64 technológiának különféle alkalmazásokkal való kompatibilitásának vizsgálatához két NAT64 implementációt használtunk: ezek a TAYGA+iptables és az OpenBSD PF.

TAYGA

A TAYGA [39] egy állapotmentes szabad szoftver NAT64 implementáció, amit GPLv2 licensz alatt adtak ki. A készítői üzemszerűen használható NAT64 implementációnak szánták olyan esetekre, amikor egy hardveres NAT64 eszköz használata túlzás lenne [39]. Mivel állapotmentes, ezért csak 1-1 hozzárendelésre (one-to-one NAT) képes IPv6 és IPv4 címek között. Ezért egy állapottartó NAT44 megoldással együtt használják, ami Linux alatt az iptables. A TAYGA a forrás IPv6 címeket egy megfelelően választott privát IPv4 tartomány különböző címeire képezi le (és természetesen végrehatja a protokoll konverziót IPv6-ról IPv4-re), majd a privát IPv4 címeket az iptables SNAT-olja a kimenő interfész publikus IPv4 címére. A másik irányban az iptables végrehajtja a (most már) cél IP címben a publikus IPv4 címről a megfelelő privát IP-címre való cserét, a TAYGA pedig az általa ismert 1-1 hozzárendelés segítségével átírja az IPv4 csomagot IPv6 csomagra. A megoldás működéséhez a TAYGA konfigurálásakor a kliensek számának (és lehetséges egyidejű aktivitásának) megfelelő méretű privát IPv4 tartományt kell beállítani.

Packet Filter

A PF (Packet Filter) [40] az OpenBSD operációs rendszer beépített tűzfala és csomagmanipulációs eszköze, és amiként az OpenBSD, a PF is BSD licensz alatti szabad szoftver. A PF támogatja a NAT-ot, a terhelés kiegyensúlyozást (load balancing), a naplózást, valamint egyidejűleg állapottartó és állapotmentes csomagszűrőként is használható.

A NAT64-et az 5.1 verzió óta támogatja, és *címcsalád fordítás*nak (address familiy translation) nevezi. NAT64 implementációja az Ecdysis projektből származik [41]. Állapottartó módban több IPv6 címet képes egyetlen IPv4 címre fordítani (many-to-one NAT), amihez állapottáblát használ. A TAYGA-val szemben ez nagy előny, mert így nem kell minden egyes csomagot két programnak feldolgoznia.

5.2.3. A megvizsgált alkalmazások

Véleményünk szerint a leggyakrabban használt (kihagyhatatlan) hálózati alkalmazások a következők: HTTP, HTTPS, SMTP, POP3, IMAP4. Rajtuk kívül gyakran használatosak még: Telnet, SSH, FTP, OpenVPN, RDP, Syslog, különféle P2P és SIP. A megvizsgált protokollokat és legfontosabb jellemzőiket a 9. táblázatban foglaltuk össze.

5.2.4. Vizsgálatok és eredmények

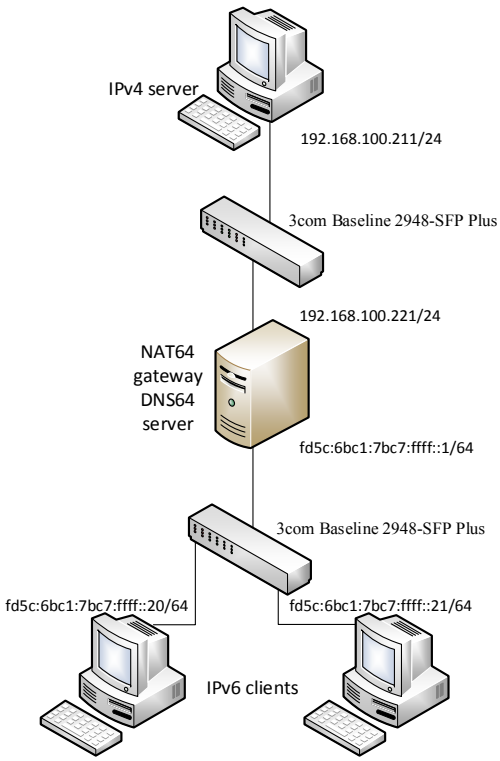
Mivel az egyes alkalmazások működése eltérő, részben eltérő struktúrájú hálózatokat kellett használnunk a vizsgálatukhoz. A továbbiakban kitérünk majd a különbségekre, most a teszthálózat általános felépítését ismertetjük. A vizsgálatokat a Széchenyi István Egyetem Távközlési Tanszékének *Távközlés-informatika oktató és kutató kaboratórium*ában végeztük. A vizsgálatokhoz elkülönített tesztthálózatot készítettünk, de ezt esetenként összekapcsoltuk a laboratórium hálózatával.

Hálózati eszközként egy *3Com Baseline 2948-SFP switch*et használtunk VLAN-okkal. (A következő ábrákon a két-két switch szimbólum ugyanazon switch eltérő VLAN-jait jelenti.) IPv6 címtartományként az fd5c:6bc1:7bc7:ffff::/64 Unique Local IPv6 Unicast Address (ULA) tartományt választottuk. Összesen legfeljebb 5 számítógépet használtunk különféle célokra. Az egyes alkalmazások vizsgálatához használt mérési összeállításokat az alábbiakban mutatjuk be. A vizsgálatok során használt részletes beállítások, az egyes szoftverek verziószámai, valamint az egyes mérések pontos

kivitelezése megtalálhatók az eredeti cikkünkben [33]. A teszteket mindkét NAT64 implementációval elvégeztük, és azonos eredményeket kaptunk. A teszteket mindig legalább két klienssel végeztük annak érdekében, hogy elkerüljük a hamis pozitív eredményt olyan esetben, amikor az első kliens sikeresen csatlakozik/működik, de a továbbiak már nem.

alkalmazás protokoll	szállítási protokoll	port
HTTP	TCP	80
HTTPS	TCP	443
SMTP	TCP	25
POP3	TCP	110
IMAP4	TCP	143
Telnet	TCP	23
SSH	TCP	22
FTP	TCP	20,21
OpenVPN	UDP/TCP	1194
RDP	TCP	3389
Syslog	UDP	514
P2P (BitTorrent)	TCP/UDP	6881-6999
SIP	TCP/UDP	5060
RTP/RTCP SIP-hez	UDP	1024-65535

9. táblázat: A megvizsgált protokollok és jellemzőik [33]



29. ábra: HTTP, HTTPS, Telnet, SSH, FTP, Syslog vizsgálatához használt teszthálózat [33]

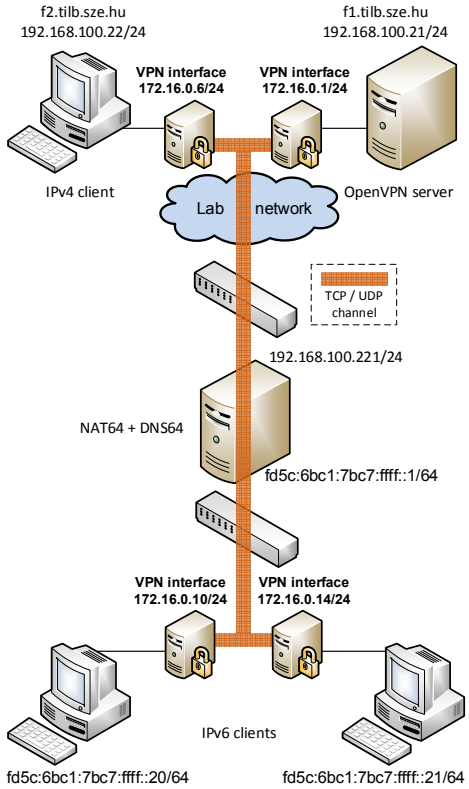
HTTP, HTTPS, Telnet, SSH, FTP, Syslog

A NAT64 technológiának a HTTP, HTTPS, Telnet, SSH, FTP, Syslog protokollokkal való kompatibilitásának teszteléséhez ugyanazt a hálózatot használtuk, a hálózat topológiája a 29. ábrán látható.

Az FTP kivételével mindegyik protokoll gond nélkül együttműködött a NAT64-gyel.

Az FTP működése során két TCP kapcsolatot használ. A *vezérlő kapcsolat* (control connection) a kliensnek a szerverre történő bejelentkezésekor épül fel, és mindvégig fennáll. Külön *adat kapcsolat* (data connection) épül fel minden egyes fájl, illetve könyvtár lista átvitele esetén. Az FTP-t *aktív mód*ban (active mode) és *passzív mód*ban (passive mode) is lehet használni. Amikor aktív módban adatkapcsolatra van szükség, akkor a kliens a vezérlő kapcsolaton keresztül (a PORT paranccsal) elküldi az IP-címét és azt a portszámot, amelyen várja a szerver kapcsolódását. A megadott IP-cím és portszám felhasználásával a szerver kezdeményezi az adat kapcsolat felépítését a kliens felé. Passzív módban viszont a kliens kéri a szervertől az IP-cím + portszám párost (a PASV parancs segítségével), melyre válaszul a szerver elküldi az IP-címét és azt a portszámot, ahol várja a kliens csatlakozását. Az adat kapcsolat felépítését a kliens kezdeményezi a szerver felé.

Ha a kliens NAT mögött van, akkor csak a passzív mód használható, aktív módban ugyanis a szerver képtelen lesz a kliens által küldött (tipikusan privát) IP-című géphez kapcsolódni. A probléma megoldására létezik ún. FTP *protokoll segítő* (protocol helper), ami figyeli a vezérlő kapcsolat forgalmát és szükség esetén végrehajtja az IP-cím és portszám fordítást. Mivel ilyet nem használtunk, aktív módban sikertelen volt a fájlok és könyvtár listák átvitele, passzív módban viszont kiválóan működött, ami megfelelt az előzetes várakozásainknak.



30. ábra: OpenVPN vizsgálatához használt teszthálózat [33]

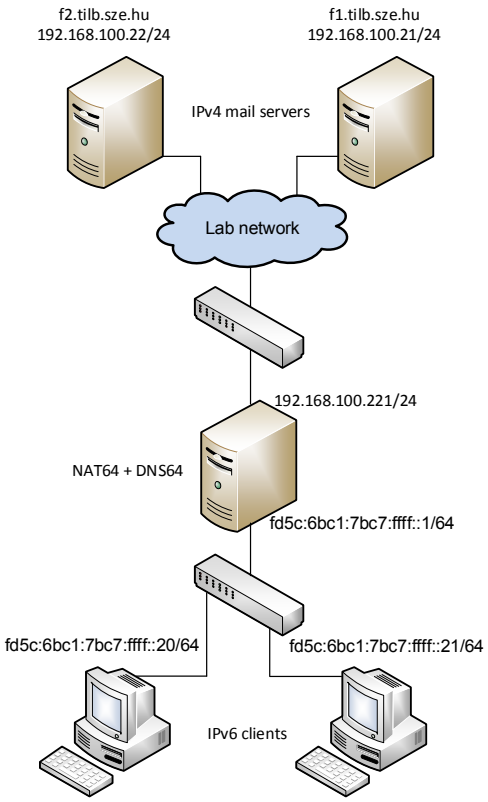
OpenVPN

Az OpenVPN vizsgálatához egy másik teszthálózatot építettünk, amely a laboratórium hálózatához is kapcsolódott. A vizsgálatainkhoz két darab IPv6-os klienst és egy darab IPv4-es OpenVPN klienst, valamit egy IPv4-es OpenVPN szervert használtunk a 30. ábra szerinti összeállításban.

Miután a szerver és az összes kliens gépen elindítottuk az OpenVPN-t, ICMP-vel (ping) vizsgálva bármely két kliens között sikeres volt a kommunikáció. Ennek magyarázatát, és az irodalomban publikált eredményektől való eltérés okát abban látjuk, hogy a használt Linux Debian Wheezy operációs rendszerben található OpenVPN *nem hivatalos fejlesztői folt*ot (unofficial developer patch) tartalmaz az IPv6 támogatására a 2.3.0 verziótól.

E-mail: SMTP, POP3, IMAP4

A levelező protokollok vizsgálatához is a labor hálózatához kapcsolódó teszthálózatot építettünk (31. ábra). Az életszerűség érdekében két IPv4-es szervert és két IPv6-os klienst használtunk. A névfeloldást a laboratórium DNS szervere végezte.



31. ábra: SMTP, POP3, IMAP4 és RDP vizsgálatához használt teszthálózat [33]

Mind a levelek SMTP-vel való elküldése, mind azok POP3 illetve IMAP4 protokollal való letöltése sikeres volt.

RDP

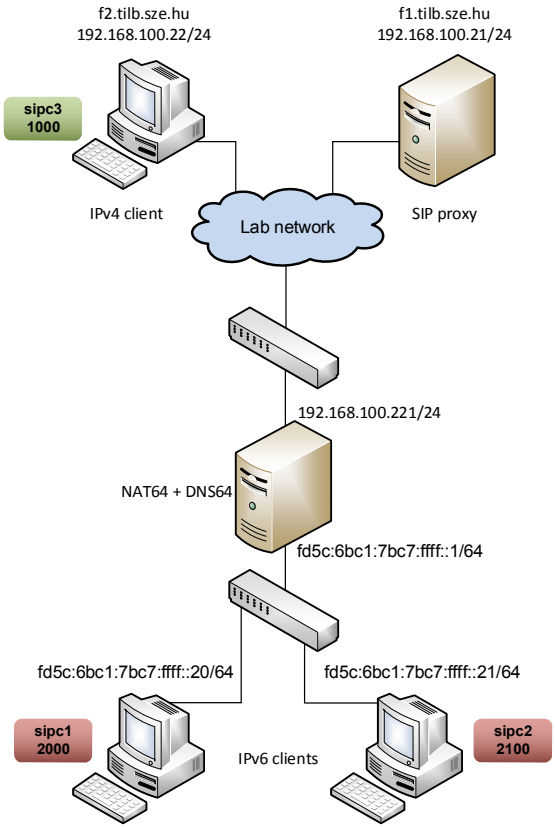
Az RDP vizsgálatát is a 30. ábrán látható hálózatot segítségével végeztük, de csak az egyik szervert használtuk. Itt is mindkét kliens sikeresen csatlakozott a szerverhez.

BitTorrent és Skype

A P2P alkalmazások családjából vett ezen két reprezentáns vizsgálatához a NAT64 átjáró külső interfészének IPv4 Internet kapcsolatát a laboratóriumi hálózaton keresztül biztosítottuk. Mindkét teszt sikertelen volt. BitTorrent esetén nem indult el a fájlok letöltése, a Skype-nál pedig már az authentikáció is sikertelen volt.

SIP

A SIP vizsgálatát a 32. ábrán látható hálózat segítségével végeztük. Csak IPv4 címen SIP proxyként a Trixbox CE 2.8.0.4 verzióját használtuk. Az egyik IPv6-os és az IPv4-es gépre Debian Linux alatt Linphone 3.5.2, a másik IPv6-os gépre Windows alá Linphone 3.6.1 SIP képes IP telefon szoftvert (softphone) telepítettünk. Azt tapasztaltuk, hogy a hívások mind IPv4-től IPv6 felé, mind az IPv6 felől IPv4 felé sikeresen felépültek, de beszélgetni nem lehetett, mert az RTP csatorna nem épült fel. Bár ellenőrző kísérleteinkben NAT44 esetén a „NAT mögött" opció biztosította a megfelelő működést, NAT64 esetén azonban ez sem oldotta meg a problémát [42].



32. ábra: SIP vizsgálatához használt teszthálózat [33]

Eredmények összefoglalása és összehasonlítása

Eredményeinket a 10. táblázatban foglaltuk össze. Az általunk megvizsgált NAT64 implementáció minden protokoll esetén azonosan viselkedett. A szakirodalomban közölt eredményekhez képest az OpenVPN esetén tapasztaltunk lényeges eltérést, mely nálunk kifogástalanul működött.

Alkalmazás protokoll	PF	Tayga	Ecdsys	
			[34]	[36]
HTTP	I	I	I	I
HTTPS	I	I	I	I
SMTP	I	I	I	I
POP3	I	I	I	I
IMAP4	I	I	I	I
Telnet	I	I	I	√
SSH	I	I	I	I
FTP passzív mód	I	I	?	?
FTP aktív mód	N	N	?	?
OpenVPN	I	I	N	N
RDP	I	I	I	√
Syslog	I	I	√	√
Skype	N	N	N	N
BitTorrent	N	N	feltételes	N
SIP	N	N	N	N

10. táblázat: NAT64 implementációk alkalmazásokkal való kompatibilitása [33]

Az irodalomból származó értékeknél megjegyezzük, hogy egyik cikkben sem említették az FTP működési módját, ezért az összefoglaló táblázatba nem tudtunk érdemi eredményt írni, bár mindkettőben vizsgálták: [34] szerint feltételes, [36] szerint pedig működőképes az FTP. Hangsúlyozzuk, hogy a méréseket mi 2013-ban végeztük a [33] cikkünkben megadott verziószámú szoftverekkel (további részletek Hajas Tamás szakdolgozatában [42]), az aktuális állapot változhat, ha további protokollsegédek válnak elérhetővé.

5.3. A NAT64 technológia portszám fogyasztása

5.3.1. Más kutatók eredményei: ami van, és ami kellene

A téma fontossága ellenére kifejezetten a NAT64 portszám fogyasztásával foglalkozó megbízható tudományos publikációt nem találtunk. Néhány olyannal találkoztunk, amelyek hálózati alkalmazások portszám fogyasztását vizsgálják.

Egyaránt hálózati alkalmazások portszám fogyasztásának *felső korlátját* vizsgálja [43] és [44], mert az Address plus Port (A+P) megoldás (RFC 6346) esetén arra van szükség. Ez a megoldás *állapotmentes cím plusz port megfeleltetést* (Stateless A+P Mapping, SMAP) használ, és ennek a támogatására a 64k méretű portszám tartományt kisebb, fix méretű tartományokra osztja fel. Ezek méretének meghatározásához az alkalmazások portszám fogyasztásának egy jó felső korlátjára van szükség. Mindkét cikk azt javasolja, hogy ez a korlát néhányszor 100 port körül legyen. Ezenkívül [43] kiválóan illusztrálja is a Google Map segítségével, hogy mi történik, ha nem áll rendelkezésre elegendő számú port: a portok számát 15-re korlátozva a térkép kb. felén fehér téglalapok maradtak, mert a szoftver párhuzamosan több TCP kapcsolatot használ a minél gyorsabb letöltés érdekében, és amikor a portszámok elfogytak, sikertelen volt az újabb TCP kapcsolatok felépítése.

A Dual Stack Light szempontjából vizsgálódnak, és a web böngészés portszám fogyasztásának a felső korlátját keresik [45] szerzői is. Azt szimulálják, hogy a TCP protocol stack hogyan reagál a portszámok elfogyására: arra a megállapításra jutnak, hogy ennek következménye egy 3 másodperces fennakadás (az idő telik, de nincs válasz). Végül aktív kliensenként 500 portot javasolnak.

Mindezekkel ellentétben a *hagyományos* (traditional) típusú NAT vagy NAT64 esetén nincs szükség a portszámtartomány előzetes felosztására, mert ezek (pl. az A+P-vel ellentétben) egy közös tartományból dinamikusan allokálnak portot az egyes alkalmazások szükségletei szerint. Ezért ezek méretezéséhez valamiféle *átlagos portszám fogyasztás* jellegű információra van szükség. Ez nem feltétlenül jellemezhető egyetlen számmal, szükség lehet valamilyen időfüggvényre.

Ami a *kiterjesztett* (extended) típusú NAT vagy NAT64 eszközöket illeti, azok a portszámokat újra fel tudják használni a különböző IP-című szerverek felé, ezért ebben az esetben elegendő a *legnépszerűbb és legnagyobb portszám igényű* cél IP-címekkel (szerverekkel) foglalkozni (részleteket ld. később).

Találtunk továbbá egy olyan cikket is, melyben az idő függvényében vizsgálták különféle hálózati alkalmazások portszám fogyasztását, de ez sajnos ugyanaz a cikk, aminek eredményeit korábban nem ismertettük, mert nem tartottuk megbízhatónak [37]. A benne használt vizsgálati módszer (és a bírálat) alaposságát továbbá az is megkérdőjelezi, hogy összesen két website tesztelése után bizonyítottnak tekintik, hogy a Firefox portszám fogyasztása magasabb, mint az Internet Exploreré. Ezért eredményeik ismertetését most sem vállaljuk fel.

Ugyan nem tudományos cikk, hanem draft RFC, de fontos eredményt közöl: [46]. A 2.1. pontban foglalkozik a NAT64 portszám fogyasztásával: a China Mobile szolgáltató azt figyelte meg, hogy a NAT64 szolgáltatás felhasználónként *átlagosan* kb. fele annyit portot igényel, mint a NAT44, mivel a legnépszerűbb web szerverek jelentős része elérhető IPv6-on keresztül, és ebben az esetben nincs szükség NAT64-re. És ez az arány tovább fog javulni az IPv6 elterjedésével. De sajnos az arányon kívül konkrét számértékeket nem közölnek.

5.3.2. Web böngészés portszám fogyasztásának vizsgálata

Ebben a részben azon kutatásaink [47] eredményét mutatjuk be, amelyek célja az volt, hogy megvizsgáljuk, hogy web böngészéskor milyen lehet egy (hagyományos típusú) NAT64 eszközön a *portszám fogyasztás nagyságrendje*, azaz a böngészők hány párhuzamos TCP kapcsolatot hoznak létre, és ez *milyen paraméterektől függhet*.

A mérési módszer

A mérésekhez a korábban említett TAYGA+iptables és OpenBSD PF NAT64 implementációkat használtuk. Minden tesztet elvégeztünk mindkét összeállítással, de az eredményekben nem találtunk érdemi eltérést.

A méréshez olyan weboldalakat választottunk, amelyek csak IPv4 címmel rendelkeztek, tehát elérésükhöz szükség volt NAT64-re. A választott weboldalakat a 11. táblázatban soroltuk fel.

Weboldal
www.amazon.com
www.ask.com
www.baidu.com
www.ebay.com
www.linkedin.com
www.live.com
www.sohu.com
www.taobao.com
www.twitter.com
www.yandex.ru

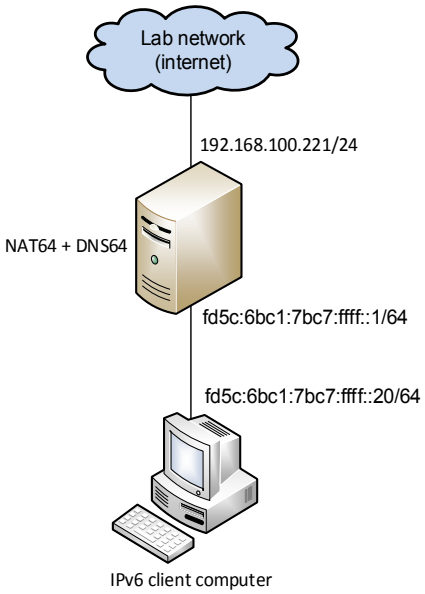
11. táblázat: A vizsgálatokhoz használt weboldalak

A méréshez különböző operációs rendszer és web böngésző kombinációkat választottuk annak érdekében, hogy meg tudjuk vizsgálni, milyen paraméterektől függ a portszám fogyasztás. A vizsgáltatokhoz használt kombinációk az alábbiak voltak:

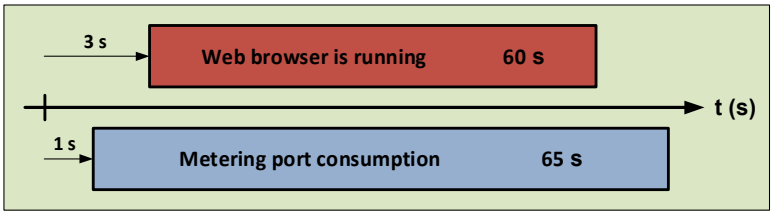
- Windows 7 Enterprise:
  - Mozilla Firefox 21.0
  - Internet Explorer 10.0.9200.16576
  - Google Chrome 27.0.1453.110m
  - Opera 12.15
- Debian Wheezy 7.1 Linux + KDE 4.8.4:
  - Iceweasel 17.0.6
  - Konqueror 4.8.4
  - Google Chrome 28.0.1500.45
- Ubuntu 12.04 LTS Linux:
  - Mozilla Firefox 21.0

A mérési összeállítás a 33. ábrán látható. A különféle böngészőket mindig az IPv6 kliens számítógépen futtattuk, és a portszám fogyasztást a NAT64 átjárón mértük.

A méréseket scriptek segítségével automatizáltuk, beleértve a távoli scriptek megfelelő időzítéssel való elindítását és leállítását is. Az időzítéseket a 34. ábrán mutatjuk be. Egy kísérlet 65 másodpercig



33. ábra: A HTTP protokoll portszám fogyasztásának mérése NAT64 átjárón [47]



34. ábra: A mérések időzítése [47]

tartott. Az első mérést 1 másodperc késleltetéssel indítottuk, és a használt portok számát másodpercenként naplóztuk. A böngészőt a portszám mérésnél még két másodperccel később indítottuk, 60 másodpercig futtattuk és még a böngésző leállításán túl két másodpercig ment a portszám mérés. Ezeket a védő intervallumokat azért használatuk, hogy a távoli script indításból és leállításából adódó szinkronizációs problémákat elkerüljük, hiszen az egyes folyamatok indítása és leállítása nem nulla idő alatt történt meg. A méréshez használt scripteket most mellőzzük, azok megtalálhatók [47] cikkünkben. Minden mérést 11-szer végeztük el.

Eredmények

Az eredmények első áttekintését a 12. táblázat segítségével tehetjük meg. A benne szereplő számértékek az adott URL megnyitásának kezdetétől a vizsgálati idő végéig mért portszám fogyasztás maximumát adják meg. Az első megfigyeléseink a következők:

- Igen nagy az eltérés (az általában) legkevesebb portot fogyasztó twitter.com (9-16 port, a böngészőtől függően) és a legtöbbet fogyasztó sohu.com (122-198) web oldal portszám fogyasztása között.
- Míg egyes oldalak portszám fogyasztása erősen függ a böngészőktől (például a baidu.com esetén csak 8 Windows/Explorerrel és 27 Ubuntu/Forefox-szal, ami több mint háromszoros eltérés) addig más oldalaknál ez az eltérés sokkal kisebb (például az ask.com estén mindig 24 és 33 között van).
- Egyes URL-eknél ugyanannak a böngészőnek a portszám fogyasztása számottevően eltérhet különböző operációs rendszerek alatt (például az ebay.com esetén a Chrome Windows alatt csak 30, Debian alatt 48 portot használt).

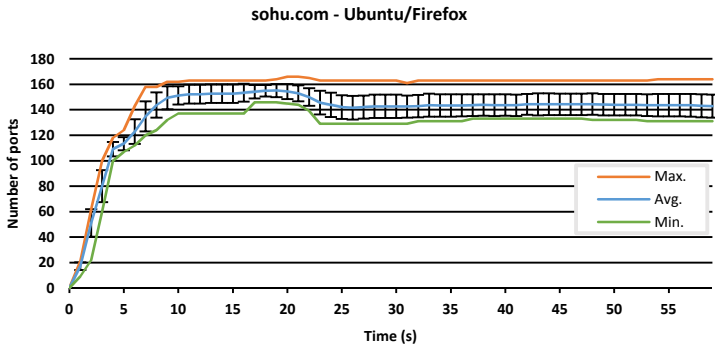
	amazon.com	**ask.com**	**baidu.com**	**ebay.com**	**linkedin.com**	**live.com**	**sohu.com**	**taobao.com**	**twitter.com**	**yandex.ru**
Windows/Firefox	74	28	25	42	18	26	198	135	11	26
Windows/Opera	58	24	16	32	14	37	140	73	13	20
Windows/Explorer	67	24	8	37	15	21	123	77	14	19
Windows/Chrome	48	25	10	30	14	23	122	67	11	18
Debian/Chrome	52	32	13	48	23	39	122	72	16	23
Debian/Iceweasel	55	31	25	43	19	25	186	131	12	20
Debian/Konqueror	35	26	11	39	12	21	122	135	9	17
Ubuntu/Firefox	52	33	27	40	20	25	166	117	11	21

12. táblázat: Egyes kliens és URL kombinációk maximális portszám fogyasztása

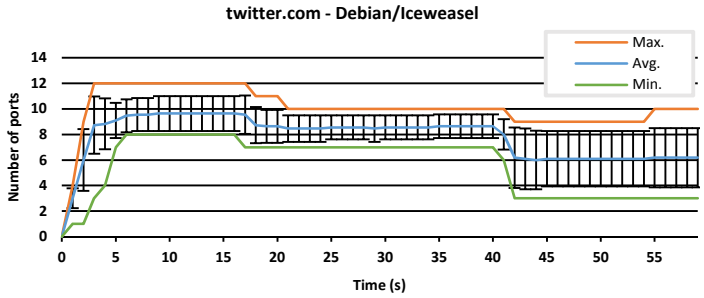
A fenti megfigyeléseken túl le kell szögeznünk, hogy ezek a számok maximális értékek, és ha ezeket használnánk a mért portszám fogyasztás tényleges eloszlása helyett, akkor az esetleg túlságosan konzervatív becsléshez vezetne (az eloszlás jellegétől függően). Ezért egyrészt mélyebbre kell ásnunk, másrészt viszont igencsak kívánatos volna az egyes eloszlásokat egyetlen számmal helyettesíteni, ugyanis olyan sok figyelembe veendő paraméterünk van, hogy a teljes eloszlás figyelembe vétele megnehezítheti lényeg megragadását.

Vajon helyettesíthetjük-e az eloszlást az átlaggal? (Az átlag jól tükrözi-e az eloszlást?) Szándékosan két erősen eltérő esetet választottunk, ahol grafikusan ábrázoljuk az egyes kísérletekben mért eredmények adott időpillanatban vett minimumát, maximumát és átlagát, sőt *hibasávok* (error bars) formájában feltüntetjük az [átlag – szórás, átlag + szórás] intervallumokat is. A 35. ábra alapján úgy tűnik,

hogy a szórás (tipikus értéke 8 körül van) viszonylag kicsi az átlaghoz képest (tipikus értéke 140 körül van), így az átlag jól tükrözi az eloszlást. Az 36. ábrán viszont egy ellenpéldát láthatunk: itt az áltag összemérhető a szórással (különösen 40 és 60 másodperc között). Mindezek ismeretében, mégis csak az átlagos portszám fogyasztás értéket tüntetjük fel az idő függvényében, amikor a 8 megvizsgált operációs rendszer + böngésző kombináció viselkedését összehasonlítjuk, ugyanis 8 vonal több, mint elég egy ábrán (a további vonalak teljesen emészthetetlenné tennék azt).



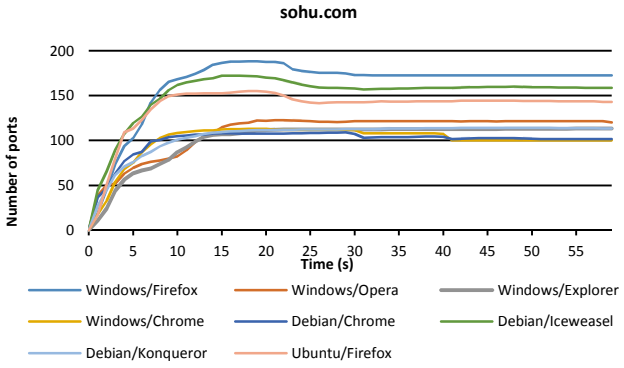
35. ábra: A sohu.com portszám fogyasztása Ubuntu/Firefox böngészővel [47]



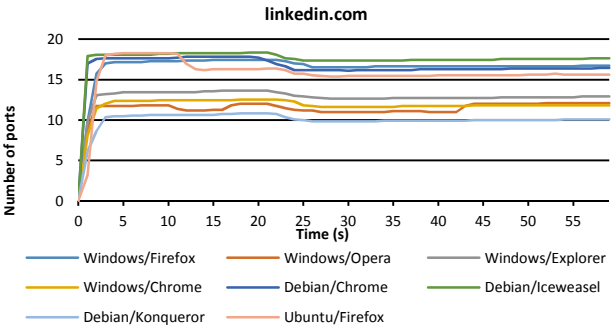
36. ábra: A twitter.com portszám fogyasztása Debian/Iceweasel böngészővel [47]

Az összes mérési eredmény megtalálható Hajas Tamás szakdolgozatának [42] CD mellékletén. Ezek vizsgálata alapján elmondható, hogy a két ábrán bemutatott helyzet reprezentatívan tükrözi a mérés stabilitását. Ugyanis a 35. ábrán nagy mértékű (100-as nagyságrendű) portszám fogyasztás mellett a relatív szórás alacsony. A 36. ábrán pedig ugyan nagy a relatív szórás, de a portszám fogyasztás értéke alacsony (nagyságrendileg 10 körüli), tehát egy NAT64 átjáró méretezése szempontjából a mérési bizonytalanság nem számottevő.

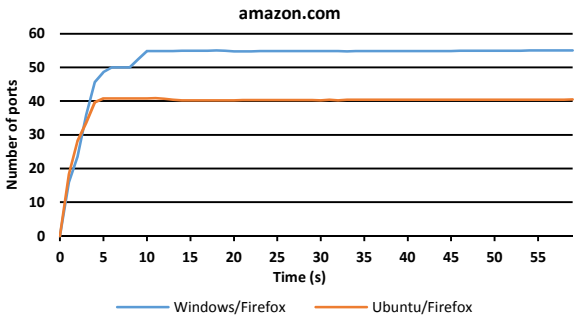
Hogyan befolyásolja az operációs rendszer + böngésző választás az egyes oldalak porszám fogyasztását? A 37. ábrán a tesztelt operációs rendszer + böngésző kombinációk portszám fogyasztását mutatjuk be a sohu.com oldal esetén. Az egyes kombinációk portszám fogyasztása közötti lényeges különbség igen jól látható: például az átlagérték maximumát (188,18) a Widows/Firefox kombináció adja 18 másodpercnél, amikor is a Debian/Crome átlagos értéke csak 110 körül van, ami lényeges eltérés. A 38. ábrán pedig a linkedin.com esetén vizsgáljuk meg ugyanezt a kérdést. Ennek az oldalnak a portszám fogyasztása lényegesen kisebb, de az eltérés itt is lényeges: itt a Debian/Iceweasel adja a maximumot (18,36) 20 másodpercnél, amikor a Debian/Konqueror értéke csak 10,8. Mindenesetre a legnagyobb portszám fogyasztás egyik esetben sem haladta meg a legkisebb érték kétszeresét



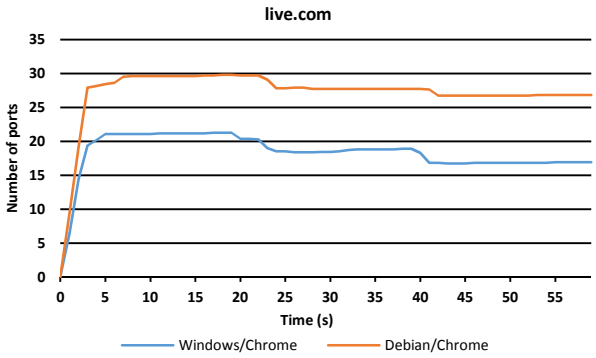
37. ábra: A sohu.com átlagos portszám fogyasztása különböző böngészőkkel [47]



38. ábra: A linkedin.com átlagos portszám fogyasztása különböző böngészőkkel [47]



39. ábra: Az amazon.com átlagos portszám fogyasztása azonos böngészővel különböző operációs rendszereken [47]



40. ábra: A live.com átlagos portszám fogyasztása azonos böngészővel különböző operációs rendszereken [47]

Adott böngésző mellett hogyan befolyásolja az operációs rendszer a portszám fogyasztást? Két olyan böngészőnk volt (Firefox és Chrome) amelyet két platformon is teszteltünk. Bár a tesztelt oldalak többségénél az eredmények igen hasonlóak voltak, néhány esetben jelentős eltérést tapasztaltunk. Például a 39. ábrán a Firefox eredményeit láthatjuk az amazon.com oldal esetén. Míg Windows esetén az átlagos portszám fogyasztás eléri az 55-öt, addig Ubuntunál ez alig haladja meg a 40-et. A 40. ábrán pedig a Chrome eredményeit láthatjuk. Itt Windows esetén eléri a 30-at, míg Debian esetén 22 alatt marad. (A legnagyobb érték itt sem érte el a legkisebb érték másfélszeresét.)

Mindezek alapján megállapítjuk, hogy a web böngészés portszám fogyasztása néhányszor 10-től néhány 100-ig terjedhet, értéke erősen függ a meglátogatott weboldaltól, de függ a böngészőtől is (kétszeresnél kisebb eltérést tapasztaltunk), sőt még adott böngésző esetén az operációs rendszertől is függhet (a tapasztalt eltérés másfélszeres alatti volt).

Megjegyezzük továbbá, hogy a fenti vizsgálatok eredményét jelentősen befolyásolja az a körülmény, hogy PC-s környezetben történtek. Okostelefonok esetén a szükséges portszámok mennyisége ettől eltérő (akár alacsonyabb is) lehet. Garai Gábor szakdolgozatában [48] arra a következtetésre jutott, hogy egy kis képernyős eszköz (Symbian alapú okostelefon) portszám fogyasztása jelentősen alacsonyabb egy nagy képernyős eszköz (Linuxot vagy Windowst futtató PC) portszám fogyasztásnál.

Mindezek alapján arra a következtetésre jutottunk, hogy a NAT64 átjárók méretezéséhez további vizsgálatokra van szükség.

5.3.3. FTP és további alkalmazások portszám fogyasztásának vizsgálata

Ezzel a kérdéssel is a [47] cikkünkben foglalkoztunk, most az ott elért eredményeket foglaljuk össze.

FTP portszám fogyasztásának mérése

A méréshez használt hálózat topológiáját a 41. ábrán mutatjuk be. Az ábra tetején látható, csak IPv4 címmel rendelkező gép volt az FTP szerver, a különböző FTP klienseket pedig az ábra alján látható csak IPv6 címmel rendelkező gépen futtattuk.

A HTTP vizsgálatához hasonlóan itt is különböző operációs rendszer és FTP kliens kombinációkat használtunk. Ezek a következők voltak:

- Windows 7 Enterprise:

- FileZilla 3.7.3 (beállítva: 1 majd 10 párhuzamosan letölthető fájl)
- Total Commander 8.01

- Debian Wheezy 7.1 Linux:
  - parancssori ftp kliens 0.17-27
  - Midnight Commander 4.8.3-10

A következő számú és méretű fájlt töltöttük le:

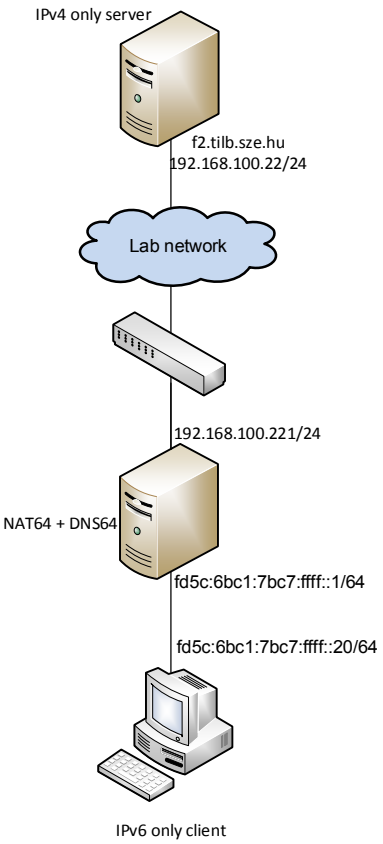
- 1x 100 MB
- 30x 1 MB
- 100x 1 MB
- 1x 500 MB

A portszám használat méréséhez a HTTP vizsgálatokhoz használt mérő scriptek módosított változatait használtuk.

FTP eredmények

Eredményeinket a 13. táblázat tartalmazza. Ezekkel kapcsolatban megállapítottuk, hogy:

- A felhasznált portok száma közel arányos az átvitt fájlok számával, de mindig valamennyivel nagyobb annál.



41. ábra: Az FTP portszám fogyasztásának méréséhez használt hálózat [47]

- A Midnight Commander különféle könyvtár listákat tölt le, ez a magyarázata annak, hogy már egyetlen fájl esetén is jelentős számú portot használ, és több fájl esetén is van egy kb. konstans többlete a többi alkalmazáshoz képest.
- A Filezilla a 10 párhuzamosan letölthető fájl beállítás esetén 9-cel több portot használt, mint amikor 1-et állítottunk be.

File méret (MB)	100	30 x 1	100 x 1	500
Debian/FTP	2	32	102	2
Debian/MC	10	39	109	10
Windows/TC	3	32	102	3
Windows/Filezilla 1 párhuzamos	4	33	103	4
Windows/Filezilla 10 párhuzamos	-	42	112	-

13. táblázat: A különféle FTP kliensek portszám használata [47]

5.3.4. További alkalmazások portszám fogyasztása

Teszteltük még a korábban NAT64-gyel kompatibilisnek talált következő protokollokat is: SSH, SCP, OpenVPN, SMTP, POP3, IMAP4, RDP, SYSLOG. (Az SCP-t ugyanazokkal a fájlokkal vizsgáltuk, mint az FTP-t). Ezek mindegyike csak 1-1 portot használt [47].

5.3.5. A globális web forgalom portszám fogyasztásának becslése

A módszer célja, alapötlete és korlátai

Web böngészés átlagos portszám fogyasztásának *közelítő, nagyságrendi jellegű becslésére* dolgoztunk ki módszert [8] cikkünkben. A módszer mind hagyományos, mind kiterjesztett NAT, illetve NAT64 eszközök méretezéséhez hasznos lehet, ezekhez hasonló módon, de két külön mennyiséget határoz meg.

A módszer alapötlete, hogy letöltéssel megvizsgáljuk a legnépszerűbb weboldalak portszám fogyasztását, és ezekből képezünk súlyozott átlagot vagy súlyozott maximumot a népszerűségüknek megfelelő súlyokkal (lásd később).

A módszernek számos korlátja van, például az, hogy a weboldalak kezdőoldalának és további oldalainak (ahova esetleg csak felhasználói név és jelszó megadása után lehet eljutni) a portszám fogyasztása jelentősen eltérhet. Továbbá elhanyagolásokat (a portszám fogyasztásnak a böngészőtől és az operációs rendszertől való függése) és közelítést (látogatások számát a látogatók számával) alkalmaztunk, ezenkívül a statisztikai adatok is hiányosak voltak. Mindezek ellenére megadtunk egy módszert, ami nagyságrendi becslést képes nyújtani, és konkrét időfüggvényeket is meghatároztunk a hagyományos, illetve a kiterjesztett NAT/NAT64 eszközökhöz. Ezek önmagukban még nem elegendők ilyen eszközök méretezésére, mert szükség van még a felhasználói viselkedés figyelembe vételére is. Például mennyi ideig időznek a felhasználók egy-egy oldalon, milyen gyakorisággal kattintanak (click), milyen valószínűséggel maradnak az adott oldalon, illetve hagyják el azt. A felhasználói viselkedésnek jól kidolgozott irodalma van, újabb és régi cikkek egyaránt találhatók benne, például: [49]-[51], ezzel mi nem foglalkoztunk.

Mit mérjünk?

Ha egy szolgáltató NAT44 vagy NAT64 eszközt szeretne méretezni, akkor tudnia kell, hogy a felhasználóinak a forgalma vajon milyen mennyiségű portszámot fog igényelni. Egy felhasználó portszám igénye szolgáltatónként több okból is eltérő lehet, hiszen országonként mások lehetnek a felhasználói szokások (más jellegű oldalakat látogatnak) és jelentős különbség lehet a mobil és a

vezetékes felhasználók portszám fogyasztása között (akár az eltérő tartalom, akár az eltérő képernyőméret miatt). Korábban feltártuk továbbá, hogy egy adott oldal letöltése során eltérés van a különböző böngészők portszám fogyasztása közt, sőt még azonos típusú böngészők között is, ha azok eltérő operációs rendszerek alatt futnak. Egy általános célú vizsgálatnál azonban vigyázni kell arra, nehogy a túl sok paraméter figyelembe vétele miatt a módszer túlságosan bonyolulttá, időigényessé, és így a gyakorlatban használhatatlanná váljon. Ezért következő döntéseket hoztuk.

- A portszám fogyasztásnak a böngészőtől és az operációs rendszertől való függését teljesen elhanyagoljuk. Ugyanis egyik oldalról a korábbi méréseink szerint a megvizsgált URL-ek esetében a legkisebb és a legnagyobb portszám fogyasztás között legfeljebb egy 2-es szorzó volt a különbség, ami nagyságrendi becslésnél (1 vagy 500 port per kliens) még megengedhető. A másik oldalról viszont a szóba jöhető böngészők és operációs rendszerek mindegyikén elvégezni a méréseket, majd azután az eredményeket azok piaci részével (ami nem pontosan ismert, ráadásul időben változó) súlyozva átlagolni az eredményeket; túlságosan nagy munkát igényelne.
- A területi, illetve a mobil/vezetékes jellegből fakadó eltéréseket úgy kezeljük, hogy egy általános módszert definiálunk, és a méréseket valamilyen konkrét adatokkal végezzük el. Szükség esetén a méréseket specifikus input adatokkal elvégezve az eredmények adott esetre nézve pontosíthatók.

Input adatok kiválasztása

Lehetséges bemeneti adatként megvizsgáltunk különböző statisztikákat. A legismertebb az *Alexa* által készített *globális 500-as lista* [52], de van ennél bővebb, illetve országonkénti vagy tartalom kategóriánkénti statisztikájuk is. Ezek a listák azonban sajnos csak *rangsor*ok (rank list), nem adják meg az egyes site-oknak a teljes web forgalomból való részesedését (amire szükség lenne az oldalakon mért porszám fogyasztási adatok súlyozott összegzéséhez). Egy másik oldalon [53] vannak ugyan súlyok, de ott egyrészt csak 35 web site-ot sorolnak fel, másrészt az adatok túl régiek (2012. éviek), ezért használhatatlanok. Egy harmadik oldalon [54] található *Nielsen top 100* lista kiváló lenne, mert az *egyedi látogatók számán* (unique audience) kívül az *összes letöltések számát* (total visits) is megadja, de sajnos a kora miatt (2010. évi) szintén használhatatlan. Végül a *Quantcast* [55] 100-as listáját választottuk. Ez sem teljesen jó, ugyanis egyrészt a 12 *rejtett profil* (hidden profile) miatt csak 88 elemét tudtuk használni, másrészt csak a *havi látogatók* (monthly people) számát közli, amivel csak közelíteni tudtuk a *havi letöltések* számát (amire a web forgalomban való súlyként szükségünk van). De legalább aktuális. (A lista országonként készült, az USA-t választottuk.)

A mérési módszer

Korábban 60 másodperces időintervallumban mértünk [47], és azt tapasztaltuk, hogy bár az oldalak ennyi idő alatt letöltődtek, mégsem csökkent nullára a portszám fogyasztás a NAT64 átjárón. Ennek két oka is lehetett: vagy tényleg éltek még a TCP kapcsolatok, vagy pedig azok ugyan már lezárultak, de a NAT64 eszközben még nem telt le a time-out, és ezért voltak még jelen az állapottáblában. Mivel különböző NAT/NAT64 eszközökben eltérő lehet a (default) time-out érték, úgy döntöttünk, hogy azt az intervallumot mérjük, amíg a TCP kapcsolat ténylegesen fennáll, és szükség esetén az egyes eszközök konkrét time-out értéke később figyelembe vehető. Az az idő, amíg a TCP kapcsolat fennáll, csak a klienstől és a szervertől függ, az esetleges NAT/NAT64 eszköztől nem, ezért a mérésekhez nem volt szükség ilyen eszközre.

A számításokhoz használt mérések előtt végeztünk egy mérés sorozatot a szükséges mérési időintervallum meghatározására, hogy az egyrészt kellően hosszú legyen, másrészt ne vesztegessünk el feleslegesen sok időt.

A mérés paraméterei

A méréshez egy átlagos otthoni számítógépet használtunk a következő paraméterekkel: AMD Athlon 64 X2 Dual Core 4200+ 2200 MHz CPU, 2 GB DDR2 667 MHz RAM, 320 GB HDD, Internet hozzáférés egy Linksys E3000 routerrel kábelneten keresztül 60 Mbps letöltési és 6 Mbps feltöltési sebesség mellett. A számítógépen Debian Linux 7.6 operációs rendszer és KDE volt.

Iceweasel (Firefox klón) 24.6.0 böngészőt használtunk, amit úgy állítottunk be, hogy üres kezdőlappal induljon (tehát ne próbáljon valamilyen oldalt betölteni vagy a legutóbb meglátogatott oldalt helyreállítani). A cache ki volt kapcsolva (0 MB beállításával), de ennek ellenére azt tapasztaltuk, hogy nőtt a `~/.cache/mozilla/firefox` könyvtár helyfoglalása, ezért minden egyes méréskor letöröltük annak tartalmát közvetlenül az oldal megnyitása előtt. (Később megvizsgáltuk a cachelelés hatását a portszám fogyasztásra, és azt tapasztaltuk, hogy nem csökkenti jelentősen a portszám fogyasztást [8].)

A Debian *nem szabad szoftver tárolójából* (non-free repository) flash playert telepítettük, hogy a böngésző be tudja tölteni az oldal esetleges flash tartalmát is. A *felugró ablakokat* (pop-up windows) blokkolva hagytuk, mert az volt a böngésző alap beállítása. Minden már beállítást is változatlanul hagytunk. A portszámmal kapcsolatos beállításokat ellenőriztük az `about:config` URL megnyitásával:

```
network.http.max-connections 256
network.http.max-persistent-connections-per-proxy 32
network.http.max-persistent-connections-per-server 6
```

Közülük az első a legfontosabb számunkra, mert ez korlátozhatja az egyszerre megnyitott TCP kapcsolatok számát, így a portszám fogyasztás értékét is.

A `netstat` Linux parancsot használtuk a nyitott TCP kapcsolatok monitorozására.

Amint már említettük, az élő TCP kapcsolatok száma érdekelt bennünket, ezért saját Linux kernelt fordítottunk (kernel verzió: 3.12.6, gcc verzió: 4.7.2-5), hogy a TCP protocol stack time-out értékét csökkenteni tudjuk. Az `include/net/tcp.h` fájban definiált TCP_TIMEWAIT_LEN paraméter értékét 60 másodpercről 1 másodpercre állítottuk.

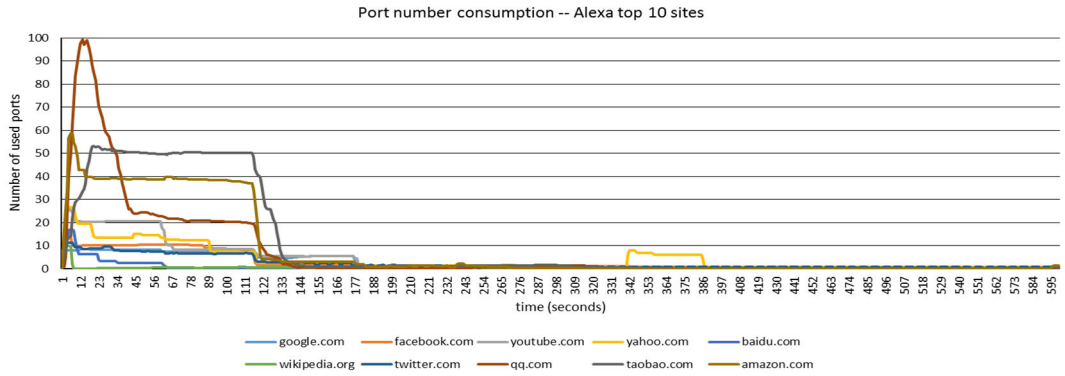
A mérési intervallum meghatározásához használt kísérlethez az 1 millió elemű Alexa [52] top site listát töltöttük le, a fájl dátuma 2014. július 18. volt. (Az egyes web site-ok súlytényezőinek becslésével kapcsolatos további paramétereket később adunk meg.) A méréshez használt scriptek és működésük leírása megtalálható [8] cikkünkben.

A mérési időintervallum meghatározása

A mérési időintervallum meghatározásához az Alexa [52] listában található első 10 URL-t használtuk fel. Néhány előzetes kísérlet után 600 másodpercre állítottuk be ezeknek a méréseknek az időtartamát. Minden oldalt 11-szer töltöttünk le, hogy kellően megbízható eredményeket kapjuk. A 11 mérés eredményét az alábbiak szerint átlagoltuk:

$$Ports(URL,t) = \frac{1}{11}\sum_{M=1}^{11} Ports(URL,t,M) \tag{1}$$

Az eredményeket a 42. ábra mutatja. Jól látható, hogy egy perc túlságosan rövid lenne mérési időintervallumnak, de két perc környékén a portszám fogyasztás erősen lecsökken, és 3 perc után (egy kivétellel) elenyészővé válik. Ezért mérési időintervallum hosszát 200 másodpercere választottuk.



42. ábra: Az Alexa lista első 10 elemének portszám fogyasztása az idő függvényében [8]

A súlytényezők meghatározása

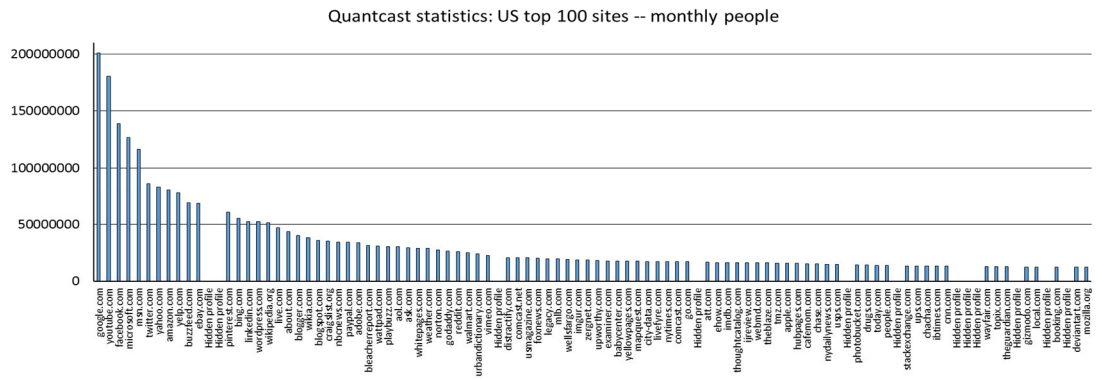
Mivel a forgalmi arányt is tartalmazó források közül egyedül a Quantcast lista [55] aktuális, ezért azt választottuk. Jobb becslés híján az egyes web oldalak látogatási számának arányát a látogatók számának arányával közelítettük. A 88 nyilvános profilú oldal adatait használtuk. A listát 2014. július 22-én töltöttük le. A súlytényezők alakulását a 43. ábrán mutatjuk be. Az oszlopok burkológörbéje az exponenciális eloszlásra emlékeztet. A súlytényezőket az alábbiak szerint határoztuk meg:

$$Weight(i) = \frac{Visitors(i)}{\sum_{j=1}^{88} Visitors(j)} \tag{2}$$

Ahol a weboldalakat 1-től 88-ig indexeljük (kihagyva a rejtett profilú oldalakat).

A portszám fogyasztás jellemzésére használt mennyiségek

Ami a hagyományos NAT eszközöket illeti, azok a web böngészés átlagos portszám fogyasztására érzékenyek, ezért számukra a 88 nyilvános profilú weboldal portszám fogyasztásának súlyozott összegét kell megadnunk (ami lényegében átlagot jelent, mivel a súlytényezők összege 1):



43. ábra: Az USA 100 legnépszerűbb weboldalának látogatószáma – a [55] forrásból 2014. 07. 22-én letöltött adatok alapján [8]

$$Ports(t) = \sum_{i=1}^{88} Weight(i) Ports(i,t) \quad (3)$$

Ahol *Ports(i,t)* az i-edik weboldal általunk mért portszám fogyasztását jelöli az idő függvényében, *Ports(t)* pedig a becsült portszám fogyasztási profilt (lásd majd lent), amivel egy tetszőleges web böngészés portszám fogyasztásának átlagos alakulását és időbeli lefutását közelítjük.

Ami az extended típusú NAT eszközöket illeti, azokra a portok újra felhasználása miatt csak a kiugróan magas népszerűségű ÉS portszám igényű szerverek (pontosabban cél IP-címek) jelenthetnek veszélyt. Így azok szempontjából a web böngészés portszám felhasználása az alábbi mennyiséggel jellemezhető:

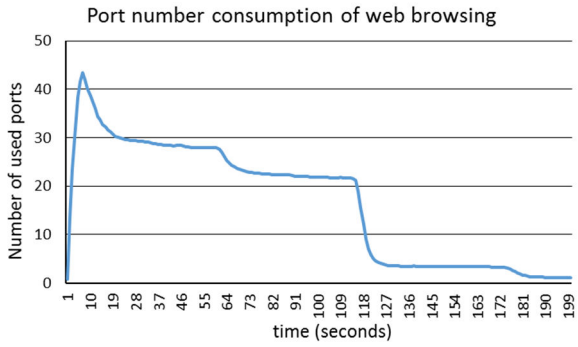
$$Ports(t) = \underset{i=1}{\overset{88}{MAX}} (Weight(i) Ports(i,t)) \quad (4)$$

Mérési eredmények és diszkussziójuk

A Quantcast lista [55] 88 nyilvános profilú weboldalát 11-szer letöltöttük, és 200 másodpercen keresztül mértük azok portszám fogyasztását. Az egyes weboldalak adott időpontokban mért porszám fogyasztását (1) szerint átlagoltuk, hogy kellően megbízható eredményeket kapjunk.

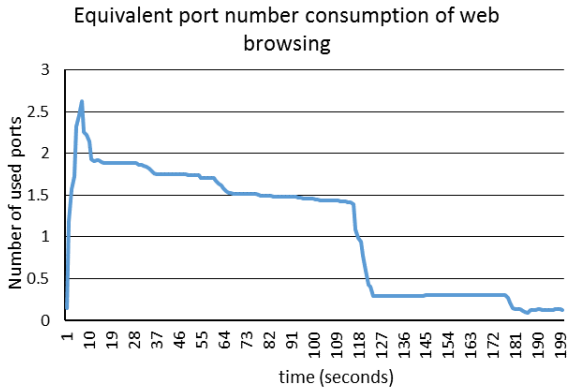
A (3) szerint kiszámított súlyozott összeget a 44. ábrán mutatjuk be. Amint az várható volt, a görbe kezdetben meredeken emelkedik, és a 2 perc körüli esése is határozott. Két kisebb esés figyelhető meg 1 és 3 percnél. Az ábra alapján úgy tűnik, hogy a NAT számára átlagosan 50 port bőven elég – ami jelentősen kisebb, mint az irodalomban közölt 500-as maximum érték –, de azt gondoljuk, hogy további mérésekre van szükség, mielőtt ilyen ökölszabályt megfogalmaznánk.

A (4) szerint kiszámított súlyozott maximumot pedig a 45. ábrán mutatjuk be. Az ekvivalens portszám fogyasztás értéke mindig kisebb, mint 3. Ennek oka, hogy kicsik a súlytényezők (a (google.com-mal 0.066873-nál kezdődnek, és a Quantcast listában a legutolsó helyen található mozilla.org értéke már csak 0.004063). Mivel itt súlyozott maximumról van szó, elvileg ezt még befolyásolhatná egy későbbi igen kiugró érték, de ennek valószínűsége a súlyok csökkenése miatt csekély. A [8] cikkben megfontolást közlünk még az esetlegesen azonos IP-címet használó virtuális web szerverekre (name based virtual hosting) nézve is.



44. ábra: Becsült portszám fogyasztási profil a ***hagyományos*** *típusú NAT eszközökhöz [8]*

Hangsúlyozzuk, hogy az eredményeink *nagyságrendi becslések*, és szeretnénk más kutatókat arra bátorítani, hogy ismételjék meg a méréseinket más böngészőkkel és eltérő, lehetőleg jobb népszerűségi listákkal. A [8] cikkünkben további megfontolásokat is közlünk a cache-elés, a tesztelésre használt szerver fizikai elhelyezése és egyéb tényezők befolyásoló hatására nézve.



*45. ábra: Becsült portszám fogyasztási profil a **kiterjesztett** típusú NAPT eszközökhöz [8]*

5.4. NAT64 implementációk stabilitása és teljesítménye

A témával kapcsolatos kutatási eredményeinket két cikkben közöltük. Az elsőben [56] (egy DNS64 implementáció mellett) csak a Linux alatti TAYGA [39] NAT64 implementációval foglalkoztunk. A másodikban [57] a TAYGA mellett az OpenBSD PF [40] által megvalósított NAT64 implementációt is megvizsgáltuk. Ez utóbbi cikkünk alapján mutatjuk be a két NAT64 implementáció stabilitását és teljesítményét. Vizsgálataink elvégzése után találtunk rá a Linux alatti Jool [58] *állapottartartó* (stateful) NAT64 implementációra, amelynek a teljesítőképességét és stabilitását a későbbiekben szintén tervezzük megvizsgálni.

Először röviden áttekintjük a korábbi kutatások eredményeit, majd bemutatjuk az általunk kidolgozott tesztelési módszert, utána közöljük és értelmezzük a mérési eredményeinket.

5.4.1. Korábbi tudományos eredmények

A TAYGA NAT64 implementáció teljesítményét (és implicit módon a TOTD DNS64 implementációét) hasonlítja össze a NAT44 teljesítményével [59]. Az Ecdysis [35] NAT64 implementáció teljesítményét (amelynek saját DNS64 implementációja van) hasonlítja össze a szerzők saját HTTP ALG (alkalmazásszintű átjáró) implementációjának teljesítményével [60]. Ugyancsak az Ecdysis NAT64 implementáció teljesítményét hasonlítja össze a NAT-PT és egy HTTP ALG teljesítményével [61]. Ezeknek a cikkeknek közös jellemzője, hogy egy adott NAT64 implementáció és egy adott DNS64 implementáció együttes teljesítményével foglalkoznak. Egyrészt ez természetes, hiszen mind NAT64 átjáróra, mind DNS64 szolgáltatásra szükség van ahhoz, hogy a csak IPv6 címmel rendelkező kliensek kommunikálni tudjanak a csak IPv4 címmel rendelkező szerverekkel, másrészt viszont ez bizonyos értelemben egy olyan „árukapcsolás”, amely eltakarhatja egy adott NAT64 vagy DNS64 implementáció önálló teljesítményét. Ugyanis bár a működéshez mindkettőre szükség van, egy nagy hálózatban ez általában két független szolgáltatás, amelyet két külön szerver nyújt: a NAT64-et egy router, a DNS64-et pedig egy DNS szerver. Így a két szolgáltatásra a legjobb implementációkat egymástól függetlenül lehet, sőt így célszerű kiválasztani.

A TAYGA NAT64 implementáció és a BIND DNS64 implementáció teljesítményét és stabilitását külön-külön teszteltük [56] cikkünkben. Bár ezek mindegyikét stabilnak és megfelelően gyorsnak

találtuk, mivel akkor mindkét célra csak 1-1 implementációt vizsgáltunk meg, további NAT64 és DNS implementációk teljesítőképességének és stabilitásának vizsgálatára volt szükség.

A DNS64 és NAT64 témában való kutatásokról jó összefoglaló található az [62] konferenciacikkben, ahol azt is demonstrálták, hogy a DNS64+NAT64 a gyakorlatban is használható megoldás egy ISP számára. Azonban a különböző DNS64 és NAT64 implementációk nagy terhelés alatti stabilitásának vizsgálatával és teljesítőképességük összehasonlításával nem foglalkoztak.

5.4.2. NAT64 átjárók teljesítményének és stabilitásának vizsgálata

A mérési eljárás

A NAT64 átjárók teljesítőképességének és stabilitásának vizsgálatára használt módszert lényegében már [56] cikkünkben kidolgoztuk, és a továbbiakban is azt használtuk. A módszer alapötlete a következő:

- A mérésekben NAT64 átjáróként viszonylag kis teljesítményű számítógépet használunk, amelyben egyedül a hálózati kártyák teljesítménye nagy. Ennek az összeállításnak az a célja, hogy már kisszámú kliens gép segítségével jelentős túlterhelést tudjunk elérni.
- Minden egyes kliens gép nagyszámú kliens viselkedését szimulálja úgy, hogy különféle célok felé indít kéréseket.
- A terhelés mértékét a kliens gépek számával szabályozzuk.
- Az összes kliens kérésére egyetlen egy nagy teljesítményű szerver válaszol.

Mivel a konkrét mérési összeállítások eltérőek voltak, a vizsgálat részleteit [57] alapján mutatjuk be. A NAT64 átjárók teljesítőképességének vizsgálatára alkalmazott teszthálózat a 46. ábrán látható. Ennek központi eleme a NAT64 átjáró (egy Intel Pentium III számítógép az ábra középső részén). Az ábra alsó részén található, csak IPv6 címmel rendelkező 8db Dell Precision 490 munkaállomás játssza a nagyszámú kliens szerepét úgy, hogy az *i*-edik (*i*=1..8) kliens gép a 10.*i*.0.0/16 IPv4 címtartománybeli szerverek elérését imitálja. Az IPv4 címeket beágyazó IPv6 címeket DNS64 szerver nélkül, „manuálisan" állította elő a teszteléshez használt bash script úgy, hogy a 2001:738:2c01:8001:ffff:ffff::/96 hálózat specifikus prefixhez hozzáírta a megfelelő IPv4-cím 32 bitjét. A NAT64 átjáróban a 10.0.0.0/8 hálózat felé a next hop router címét 193.225.151.70-re állítottuk, és ezt a címet az ábra felső részén látható Dell Precision 490 munkaállomásra húztuk fel, amely minden a 10.0.0.0/8 tartománybeli IP-című szerver helyett válaszolt (ehhez iptables segítségével a bejövő csomagokat a gép a saját IP-címére irányította át).

Ami a gépek konfigurációját illeti, a NAT64 átjáró egy 800MHz-es Intel Pentium III processzoros számítógép volt 256MB memóriával és két darab 3Com 3c940 Gigabit Ethernet hálózati kártyával. A Dell Precision 490 munkaállomásokban pedig 2 db kétmagos Intel Xeon 5130 2GHz CPU üzemelt. (A gépek pontos konfigurációja megtalálható: [57].) Ami az operációs rendszert illeti, minden gépre Debian Squeeze 6.0.3 GNU/Linux operációs rendszer telepítettük, beleértve a NAT64 átjárót is, amikor Linux alatt használtuk. Amikor pedig OpenBSD alatt, akkor annak verziója 5.1 volt.

Az egyes gépeken az IP-címeket a 46. ábrának megfelelően állítottuk be, további részletek megtalálhatók a [57] cikkünkben.

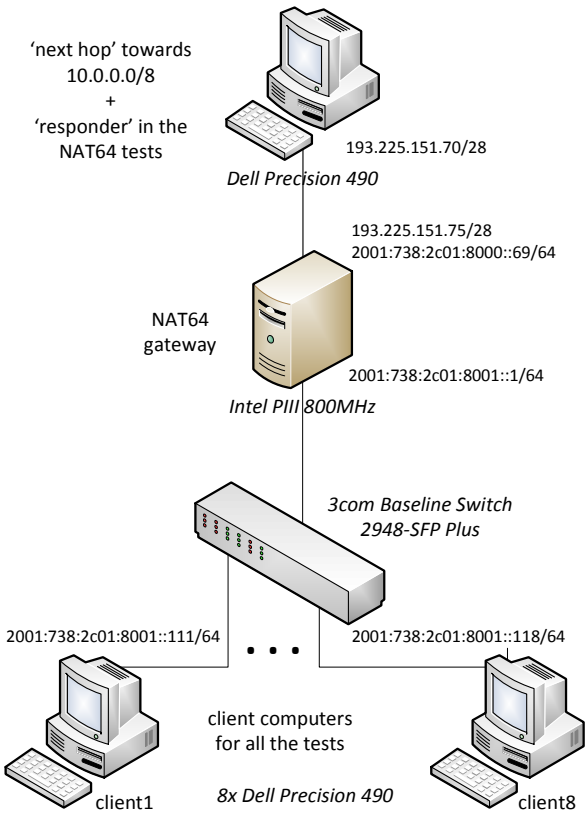
A méréseket az alábbi script segítségével végeztük, amelyet a kliensek hajtottak végre:

```
#!/bin/bash
i=`cat /etc/hostname | grep -o .$`
for b in {0..255}
do
  rm -r $b
  mkdir $b
  for c in {0..63..4}
  do
```

```
    ping6 -c11 -i0 -q 2001:738:2c01:8001:ffff:ffff:10.$i.$b.$c \
      >> $b/nat64p-10-$i-$b-$c &
    ping6 -c11 -i0 -q 2001:738:2c01:8001:ffff:ffff:10.$i.$b.$((c+1)) \
      >> $b/nat64p-10-$i-$b-$((c+1)) &
    ping6 -c11 -i0 -q 2001:738:2c01:8001:ffff:ffff:10.$i.$b.$((c+2)) \
      >> $b/nat64p-10-$i-$b-$((c+2)) &
    ping6 -c11 -i0 -q 2001:738:2c01:8001:ffff:ffff:10.$i.$b.$((c+3)) \
      >> $b/nat64p-10-$i-$b-$((c+3))
  done
done
```

A script az *i* sorszámot a gép nevének utolsó karakteréből határozza meg, majd a 10.*i*.0.0/16 tartomány mind a 2^{16} darab IP címével generál IPv4 címet beágyazó IPv6 címet és azok mindegyike felé 11 db ICMP *echo request* kérést küld a ping6 parancs segítségével. A parancsok eredményét fájlokba naplózza, amelyeket később dolgozunk fel. A minél nagyobb terhelés elérése érdekében a 4 CPU magot kihasználva 4 db ping6 parancsot konkurens módon adunk ki (közülük ez első három végén & jel van).

Az egyes méréseknél a kliensek számának megválasztásával (1, 2, 4 és 8) tudtuk a terhelés mértékét megfelelően beállítani. A használni kívánt kliens gépeken a scriptek egyidejű indítását egy KDE grafikus felülettel rendelkező gépről a Konsole nevű grafikus terminál program *bemenetek szétosztása* (Send Input to All Sessions) funkciójával valósítottuk meg.



46. ábra: A NAT64 átjárók teljesítőképességének vizsgálatához használt teszthálózat [57]

Mértük a ping6 parancsok válaszidejét, valamint a NAT64 implementációt futtató teszt gépen a processzor terhelést és a memória fogyasztást. Linux alatt ezeket a következő paranccsal naplóztuk:

```
dstat -t -c -m -l -p --unix --output load.csv
```

OpenBSD alatt pedig az alábbi parancssort használtuk:

```
vmstat -w 1 >load.txt
```

Eredmények

A 14. táblázatban a TAYGA eredményeit mutatjuk be. Az első sorban az adott mérésben résztvevő kliensek száma található. A második sor a csomagvesztés értékét mutatja. A harmadik a negyedik és az ötödik sorban rendre a ping6 parancs válaszidejének átlagát, szórását és maximumát adtuk meg. A hatodik és a hetedik sor a teszt gépen a processzor kihasználtságának átlagát és szórását tartalmazza. A nyolcadik sor a másodpercenként továbbított csomagok számát mutatja. Az utolsó sor pedig a teszt gép memóriafogyasztását adja meg.

1	kliensek száma		1	2	4	8
2	csomagvesztés [%]		0,01	0,01	0,03	0,03
3	a ping6 parancs válaszideje [ms]	átlag	0,447	0,957	1,986	4,406
4		szórás	0,103	0,158	0,321	0,474
5		maximum	5,202	5,438	8,057	13,965
6	CPU kihasz-	átlag	69,8	84,3	97,8	100,0
7	náltság [%]	szórás	3,3	1,7	2,2	0,1
8	forgalom mértéke [csomag/s]		5 721	6 614	7 085	6 890
9	memóriafogyasztás [MB]		10,8	11,4	11,9	12,9

14. táblázat: A TAYGA NAT64 implementáció teljesítménye [57]

1	kliensek száma		1	2	4	8
2	csomagvesztés [%]		0,02	0,02	0,02	0,02
3	a ping6 parancs válaszideje [ms]	átlag	0,405	0,486	0,606	1,194
4		szórás	0,050	0,073	0,127	0,250
5		maximum	1,770	1,433	3,904	7,055
6	CPU kihasz-	átlag	31,7	50,1	80,1	90,3
7	náltság [%]	szórás	5,4	5,0	5,1	4,7
8	forgalom mértéke [csomag/s]		5 909	11 091	18 367	22 886
9	memóriafogyasztás [MB]		2,4	3,5	5,5	7,9

15. táblázat: A PF NAT64 implementáció teljesítménye [57]

Az eredmények elemzése:

- Bár csomagvesztés már egyetlen kliens esetén is előfordult, ennek mértéke mindig nagyon alacsony volt (a maximális érték 0,03%, ami 10 000-ből 3 elveszett csomagot jelent).
- A válaszidő közel lineárisan nőtt a terhelés függvényében: a terhelés megduplázásának hatására a válaszidő is közel duplázódott.
- A másodpercenként kiszolgált csomagok száma addig tudott nőni, amíg volt szabad CPU kapacitás. Négy kliensnél a CPU kihasználtsága már 97,8% volt, és nyolc kliensnél már nem tudott a TAYGA több csomagot kiszolgálni, sőt az átvitt csomagok száma kis mértékben még csökkent is 7 085-től 6 890-re, ami kisebb, mint 3%-os csökkenés.

- A memóriafogyasztás mindig alacsony volt (13MB alatt maradt), és csak nagyon gyengén nőtt a terhelés függvényében.

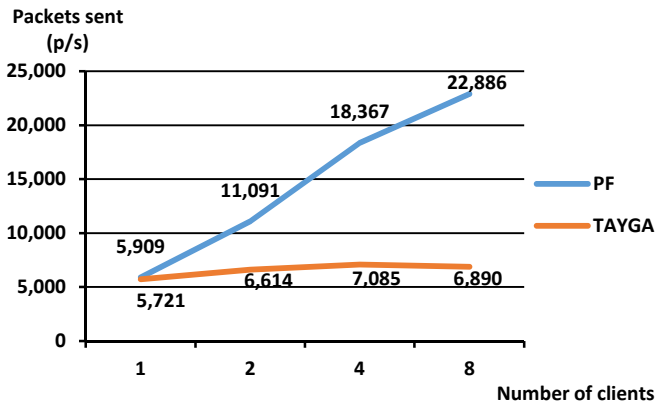
A fentieket összefoglalva megállapíthatjuk, hogy a TAYGA jól teljesített, a memóriafogyasztása alacsony volt, és nagy terhelés esetén a válaszideje a terhelés függvényében közelítően lienárisan nőtt, azaz, a TAYGA megfelel a *graceful degradation* [63] alapelvnek (azaz jelentős túlterhelés esetén sem omlik össze, és a teljesítménye nem csökken hirtelen, a terheléshez képes aránytalan mértékben).

A 15. táblázatban a PF eredményeit mutatjuk be. (A táblázat felépítése az előző táblázatéval azonos, ezért nem részletezzük.)

Az eredmények elemzése:

- A csomagvesztési arány nagyon alacsony volt, teheléstől függetlenül 0,02%.
- Amíg volt jelentés mennyiségű szabad CPU kapacitás, a válaszidő a tehelés függvényében lineárisnál kisebb mértékben nőtt. Azután 4 helyett 8 kliens használata esetén a válaszidő már közel megduplázódott.
- A másodpercenként kiszolgált csomagok száma lényegesen nőni tudott a kliensek számának növelésével: 1 helyett 2 kliens esetén közel megduplázódott (5 909 helyett 11 091), majd az arány csökkent: 4 kliens esetén 18 367 lett, végül 8 kliensnél 22 886, ami már a telítődés jeleit mutatja, de még mindig maradt közel 10% szabad CPU kapacitás.
- A memóriafogyasztás nagyon alacsony volt, és a kliensek számának függvényében a lienárisnál alacsonyabb mértékben emelkedett.

Összefoglalásként megállapíthatjuk, hogy a PF kiváló teljesítményt nyújtott, és amíg tesztelni tudtuk, viselkedése megfelelt a graceful degradation elvnek, de még 8 kliensnél is maradt szabad CPU kapacitása. Memóriafogyasztása is nagyon alacsony volt.



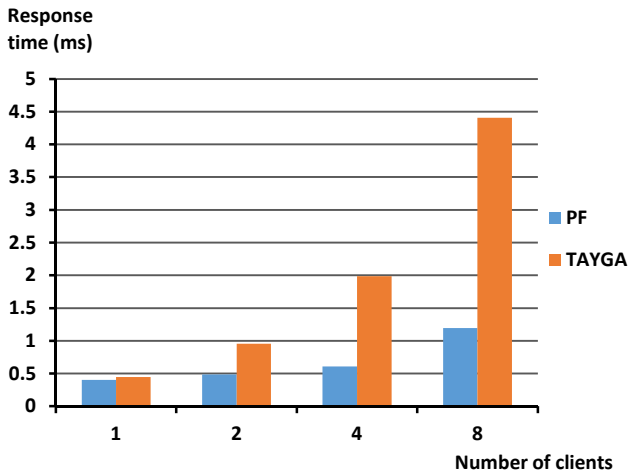
47. ábra: A továbbított csomagok másodpercenkénti száma a kliensek számának függvényében [57]

A TAYGA és a PF teljesítményét a 47. és 48. ábrán hasonlítjuk össze. Megállapítjuk, hogy mérsékelt terhelés (1 kliens gép) mellett a válaszidőik és az átvitt csomagok száma nagyon hasonló volt. Viszont erős túlterhelés (8 kliens gép) esetén a PF 22 886 csomagot tudott kezelni másodpercenként 1,2 ms átlagos válaszidő mellett, míg a TAYGA csak másodpercenként 6 890 csomag átvitelére volt képes 4,4 ms válaszidővel. Ez igen jelentős (3,3-szoros) teljesítménykülönbség. Ennek okát a következő két tényezőben látjuk:

- A TAYGA állapotmentes NAT64 megoldás, ezért egy állapottartó NAT44 megoldással (iptables) együtt kellett használnunk: így minden csomagot két külön programnak kellett kezelnie.

- Mivel a TAYGA felhasználói címtérben (user space) működik, minden csomagot át kell másolni először a kernel címteréből ide, aztán pedig vissza.

Végezetül megállapítjuk, hogy mindkét megoldás (a TAYGA és a PF) stabilnak bizonyult és alkalmasnak tartjuk őket NAT64 funkció ellátására. Amennyiben egy rendszerben az IPv6 kliensek és az IPv4 szerverek között nagy forgalom várható, akkor a PF használatát javasoljuk.



48. ábra: Válaszidő a kliensek számának függvényében [57]

5.5. DNS64 implementációk stabilitása és teljesítménye

A témával kapcsolatos kutatási eredményeinket két cikkben közöltük. Az elsőben [56] (egy NAT64 implementáció mellett) csak a BIND [64] által megvalósított DNS64 implementációval foglalkoztunk. A másodikban [32] a BIND a mellett a TOTD [65] DNS64 implementáció teljesítményét és stabilitását megvizsgáltuk. Azóta további két implementációval (Unbound és PowerDNS) is foglalkoztunk, és mind a négy implementáció esetén 1, 2 és 4 magos processzorral rendelkező tesztgépeken is elvégeztük a vizsgálatokat. Ezek eredményét nemzetközi tudományos folyóiratcikkben tervezzük publikálni [66]. A fentieken túl tervezzük még egy új, kísérleti stádiumban levő DNS64 implementáció, az MTD64 (Multi-Threaded DNS64) [11] vizsgálatát is.

Ami más kutatók eredményeit illeti, lényegében ugyanazokat sorolhatnánk fel, amit a NAT64 esetében (egy adott NAT64 implementáció és egy adott DNS64 implementáció együttes teljesítményének a vizsgálata), de nem szeretnénk önmagunkat ismételni, ezért ezzel a kérdéssel most többet nem foglalkozunk. A továbbiakban [32] alapján fogunk haladni.

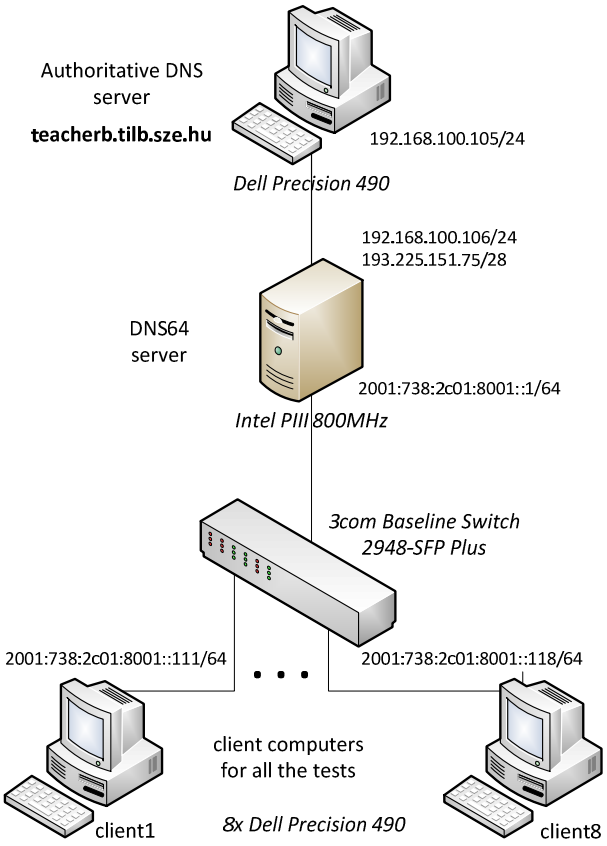
5.5.1. A megvizsgált DNS64 implementációk

A BIND – ami DNS szerverek közt de facto szabványnak számít – a 9.8 verziótól tartalmaz DNS64 támogatást, ezért ez kihagyhatatlan volt. Mivel a BIND egy nagy és komplex szoftver sokféle DNS funkcionalitással (pl. authoritative, recursive, DNSSEC támogatás), a másik választásunk egy pehelysúlyú DNS64 proxyra, nevezetesen a TOTD-re esett. Mindkét implementációt a következő három operációs rendszer alatt teszteltük: Linux, OpenBSD és FreeBSD.

5.5.2. A tesztkörnyezet

Az volt a célunk, hogy megvizsgáljuk és összehasonlítsuk a kiválaszott implementációk teljesítményét. Arra is nagy hangsúlyt fektettünk, hogy jelentős túlterhelés mellett vizsgáljuk a stabilitásukat. (A szoftverek teszteléséhez valamilyen hardvert kellett használnunk, de nem hardverek teljesítményének vizsálata volt a célunk.)

A tesztelésre használt hálózat topológiáját a 49. ábrán mutatjuk be. A teszthálózat központi eleme a DNS64 szerver.



49. ábra: A DNS64 teszthálózat [32]

A mérésekhez olyan névtérre volt szükségünk, amely:

- szisztematikusan leírható
- a nevekhez csak IPv4 címek tartoznak
- a névfeloldás helyben azonnal elvégezhető

Erre a célra a `10-{0..10}-{0..255}-{0..255}.zonat.tilb.sze.hu` névteret használtuk, amit a 10.0.0.0 – 10.10.255.255 IPv4 címekre képeztünk le az ábra tetején látható `teacherb` névkiszolgálóval, ami a 192.168.100.105 IPv4 címen volt elérhető.

A DNS64 szerver ezekhez az IPv4 címekhez *IPv4 címet beágyazó IPv6 cím*eket rendelt a 2001:738:2c01:8001:ffff:ffff:0a00:0000 – 2001:738:2c01:8001:ffff:ffff:0a0a:ffff tartományból.

Az ábra alsó részén látható, csak IPv6 címmel rendelkező DELL munkállomások játszották DNS64 teljesítménymérésekben a nagyszámú kliens szerepét. A mérésekhez lényegében ugyanazokat a gépeket használtuk, mint a korábban leírt, ICMP-vel végzet NAT64 teljesítménymérésekhez, azzal a különbséggel, hogy a DNS64 vizsgálatnál csak 128MB RAM volt a Pentim III teszgépben. Az eszközök pontos konfigurációja megtalálható [32] cikkünkben. Debian Squeeze 6.0.3 operációs rendszert telepítettünk minden gépre (a Pentium III tesztgépre is, amikor Linux alatt használtuk). Az OpenBSD verziója 5.1, a FreeBSD-é pedig 9.0 volt.

Az egyes eszközök IP címeit a 49. ábrának megfelelően állítottuk be. A részletes beállítások (BIND, TOTD konfigurációs állománya, valamint a BIND zónafájlt előállító script is) megtalálhatók a [32] cikkünkben.

5.5.3. A DNS64 teljesítménymérés módszere

A cache-elés hatásának kiköszöbölésére az egyes kliensek egymással át nem lapolódó névteret használtak. Az *i.* kliens a 10.*i*.0.0/16 hálózatra leképzett névtérre vonatkozó kéréseket küldött. Így minden kliens 2^{16} db névfeloldást végzett el. A végrehajtási idő megfelelő mérése érdekében minden kliens 256 kísérletet végzett, és egy-egy kísérletbe 256 névfeloldás tartozott, amihez a Linux szabványos `host` parancsát használtuk. A kísérletek végrehajtási idejét a GNU `time` paranccsal mértük (ami nem egyezik meg a `bash` shell `time` parancsával). A kliensek a következő script segítségével hajtották végre a 256 kísérletet:

```
#!/bin/bash
i=$(cat /etc/hostname|grep -o .$) # ordinal number of the client computer
rm dns64-$i.txt
for b in {0..255}
do
    /usr/bin/time -f "%E" -o dns64-$i.txt –a ./dns-st-c.sh $i $b
done
```

A mérésben részt vevő kliens gépeken a scriptek szinkronizált indítására a KDE `Konsole` nevű terminál programjának a „bemenetek szétosztása” (Send Input to All Sessions) funkcióját használtuk. A fenti scriptben meghívott `dns-st-c.sh` nevű script (amely két paramétert vett át) volt a felelős egy 256 névfeloldásból álló kísérlet végrehajtásáért. Ennek tartalma a következő volt:

```
#!/bin/bash
for c in {0..252..4} # that is 64 iterations
do
    host 10-$1-$2-$c.zonat.tilb.sze.hu &
    host 10-$1-$2-$((c+1)).zonat.tilb.sze.hu &
    host 10-$1-$2-$((c+2)).zonat.tilb.sze.hu &
    host 10-$1-$2-$((c+3)).zonat.tilb.sze.hu
done
```

A `for` ciklus magjában négy `host` parancsot indított el konkurrens módon, melyből az első három aszinkron módon („háttérben”) futott (így tudtuk kihasználni a két darab kétmagos processzor számítási teljesítményét a minél nagyobb terhelés létrehozására), és a ciklus magja 64-szer került végrehajtásra.

A méréssorozatban a mérésben részt vevő kliensek számát 1-től 8-ig duplázással növeltük. A kísérletek végrehajtási idején kívül mértük a DNS64 szerver programot futtató gépen a *memória- és processzorhasználat*ot is. Erre Linux alatt a következő parancsot használtuk:

```
dstat -t -c -m -l -p --unix --output load.csv
```

BSD rendszerek alatt pedig ezt:

```
vmstat -w 1 >load.txt
```

5.5.4. A DNS64 mérések eredményei

Mind a két szoftvert (BIND, TOTD) mind a három operációs rendszer (Linux, OpenBSD, FreeBSD) alatt megvizsgáltuk. Ráadásul a BIND-ot, recursorként is (amikor a névfeloldás lépéseit *iterative query*k segítségével maga végzi el) és forwarderként is (amikor egy másik szervert kér meg a névfeloldás lépéseinek elvégzésére egy *recursive query*vel). Így eredményül kilenc táblázatot kaptunk, amelyekből itt csak kettőt mutatunk be (a többi is megtalálható [32] cikkünkben): ezek a Linux alatti mérési eredmények (ahol a legjobb volt az implementációk teljesítménye), ezen belül az összehasonlíthatóság értekében a BIND-nak a forwarder módban mért teljesítménye, mivel a TOTD csak forwarderként képes működni.

1	kliensek száma		1	2	4	8
2	egy kísérlet	átlag	1,127	1,542	3,183	6,318
3	végrehajtási	szórás	0,049	0,037	0,081	0,104
4	ideje [s]	maximum	1,650	1,680	3,310	6,470
5	CPU kihasz-	átlag	61,77	95,56	100,00	100,00
6	náltság [%]	szórás	4,1	2,3	0,0	0,0
7	memóriafogyasztás [MB]		40	58	57	57
8	kérések száma [kérés/s]		227	332	322	324

16. táblázat: A BIND DNS64 implementáció teljesítménye, Linux, forwarder [32]

A BIND-nak Linux alatt forwarderként mért eredményeit a 16. táblázat tartalmazza, melynek értelmezése a következő. Az első sorban látható az adott kísérletben részt vevő kliensek száma. (A DNS64 szerver terhelése a kliensek számával nőtt.) A második, harmadik és negyedik sorok rendre az egy kísérlet (256 db `host` parancs) végrehajtási idejének átlagát, szórását és maximumát adják meg. Az ötödik és hatodik sorban található a processzor kihasználtságának átlaga és szórása. A hetedik sorban a becsült memóriafogysztást adtuk meg. (Ennek az értéknek viszonylag nagy a bizonytalansága, mert nem a DNS64 szerver processz memóriahasználatának változását mértük, hanem a Linux rendszer szabad memória értékének a csökkenését, amit más processzek is befolyásolhattak. Azért tartjuk helyesnek ezt a megoldást, mert ha pusztán a processz memóriafogyasztását vizsgáltuk volna, akkor esetleg nem mértünk volna bele olyan memóriafogyasztást, ami nem a DNS64 szerver processznél jelentkezik, de mégis annak a céljait szolgálja, pl. kernel pufferek.) A nyolcadik sorban egy számított érték található, nevezetesen a másodpercenként kiszolgált kérések N_R száma:

$$N_R = \frac{256 * n_k}{T_E} \tag{5}$$

ahol n_k a mérésben részt vevő kliensek száma, T_E pedig az egy kísérlet (256 db `host` parancs) átlagos végrehajtási ideje.

A mérési eredmények alapján megállapíthatjuk, hogy:

- A terhelés növelése nem okoz jelentős teljesítménycsökkenést, és a rendszer egyáltalán nem omlik össze a túlterhelés hatására. Még amikor a processzor kihasználtság 100%-os, a válaszidő akkor is csak közelítőleg lineárisan nő a terhelésnek (azaz a kliensek számának) függvényében.
- Bár nem tudunk pontos becslést adni a DNS64 szerver memóriafogyasztására, de látható, hogy az még nagyon erős túlterhelés esetén is mérsékelt.

- A táblázat utolsó sorában látható, hogy a másodpercenkénti kérések számának a maximumát a rendszer két kliens esetén érte el. A kliensek számának további növelése csak a válaszidőt növelte, de a másodpercenkénti kérések számát nem. Ennek oka, hogy a teszt program nem tudott addig új kérést küldeni, amíg a konkurrens módon indított négy `host` parancs közül az utolsóra meg nem jött a válasz.

A fenti eredmények azért nagyon fontosak, mert azt mutatják, hogy a BIND DNS64 szerver viselkedése megfelel a *graceful degradation* [63] elvnek; azaz amennyiben nincs elegendő erőforrás a kérések kiszolgálására, akkor a rendszer válaszideje csak *lineárisan* nő a terhelés függvényében.

Egy másik fontos megfigyelés, hogy a legnagyobb terhelés (8 kliens) esetén a végrehajtási idő szórása (0,104s) kisebb, mint 2%-a az átlagnak (6,318s) és maximuma (6,470) is csak 2,5%-kal több, mint az átlag. Ez azt mutatja, hogy a rendszer még komoly túlterhelés esetén is nagyon stabil.

Ezen két megfigyelés alapján a Linux alatt futtatott, forwarderként használt BIND egy kiváló jelölt üzemszerűen használható DNS64 szervernek „éles" kereskedelmi rendszerekben is.

1	kliensek száma		1	2	4	8
2	egy kísérlet	átlag	0,791	1,310	2,158	4,348
3	végrehajtási	szórás	0,038	3,863	3,754	5,301
4	ideje [s]	maximum	1,370	64,950	63,730	68,540
5	CPU kihasz-	átlag	38,00	58,05	80,16	84,79
6	náltság [%]	szórás	2,4	27,1	27,5	29,2
7	memóriafogyasztás [MB]		1,0	1,1	1,6	0,8
8	kérések száma [kérés/s]		324	391	474	471

17. táblázat: A TOTD DNS64 implementáció teljesítménye, Linux, forwarder [32]

A TOTD-nek Linux alatt (forwarderként, hiszen csak erre képes) mért eredményeit a 17. táblázat tartalmazza, melynek értelmezése a megegyezik a 16. táblázat értelmezésével.

A TOTD megmóriaigénye feltűnően alacsony, amit a cache-elés hiánya okoz. Mivel a tesztelési módszerünk szándékosan kiküszöbölte a cache-elés hatását, ezért mérésünkben ez nem okozott hátrányt a BIND-dal szemben, de egy valós rendszerben a TOTD teljesítményét a cache-elés hiánya lényegesen ronthatja. (Kicsi, beépített rendszerekben viszont komoly előny lehet az alacsony memóriaigény!)

Ami a TOTD teljesítményét illeti, alacsony terhelés (1 kliens) mellett a teljesítménye kiváló: másodpercenként 324 kérést szolgált ki 38 % CPU terhelés mellett, szemben a BIND-dal, ami másodpercenként csak 227 kérés kiszolgálására volt képes, ráadásul 61,77 % CPU terhelés mellett. Bár az egy kísérlet végrehajtási idejének maximuma 1,370s, ami lényegesen nagyobb, mint az átlag (0,791s), de még elfogadható, hiszen ez 64 iteráció ideje.

Azonban nagyobb terhelés (2-8 kliens) mellett a TOTD eredményei elfogadhatatlanok az egy kísérlet maximális végrehajtási idejére kapott 60-80s körüli értékek miatt. Megvizsgáltuk a jelenséget, és azt tapasztaltuk, hogy a TOTD *időnként* (ritkán, és látszólag szabálytalanul) 1 perc körüli ideig nem válaszolt, aztán folytatta a működését. (A TOTD egyébként mindhárom operációs rendszer alatt hasonlóan viselkedett.) Közben az authoritatív DNS szerver a `teacherb` gépen kifogástalanul működött.

A fentiek miatt a TOTD (jelentős terhelés mellett biztosan) nem alkalmas üzemszerű használatra. Viszont úgy ítéltük meg, hogy a BIND-hoz képest lényegesen kisebb számítási teljesítmény igénye miatt érdemes a TOTD hibáját megkeresni és kijavítani. (Ezt meg is tettük; a következőkben beszámolunk az eredményeinkről.)

Még annyit említünk meg, hogy a BIND BSD rendszereken gyengébb átlagos teljesítményt nyújtott, mint Linux alatt, viszont FreeBSD alatt ezt nagyon stabilan tette, tehát ha adott esetben biztonsági megfontolások miatt FreeBSD platformra van szükség (pl. *jail*ben való futtatás céljából), és a teljesítményáldozat elfogadható, akkor a BIND DNS64 szerver használata FreeBSD alatt jó megoldás lehet. (További részletek a [32] cikkünkben.)

5.6. A TOTD DNS64 implementáció stabilitási és biztonsági kérdései

Amint említettük, a BIND-nál lényegesen jobb átlagos teljesítménye (alacsonyabb számításigénye) alapján a TOTD szoftvert érdemesnek tartottuk arra, hogy megkeressük és kijavítsuk a hibáját. Erről a munkánkról [67] cikkünken adtunk számot, ahol vázlatosan leírtuk a tesztelés lépéseit is. Most csak a hibát és kijavítását mutajuk be.

5.6.1. A programozási hiba és kijavítása

A DNS kérés és válasz üzenetek egyaránt tartalmaznak egy 16 bites *trazakcióazonosító* (*Transaction ID*) mezőt, amelynek az értékét a kliens állítja be, és a szerver változatlanul beleteszi a válaszba. A kliens ennek alapján tudja azonosítani a szerver válaszát. (A DNS üzenetek felépítését részletesen bemutattuk [11] cikkünkben.) Mivel a TOTD proxyként működik, egy kérés vétele után kliensként kell viselkednie: egy DNS kérést kell küldenie egy olyan DNS szervernek, amely *recursive query*ket fogad (a TOTD ezt a szervert forwardernek nevezi). A TOTD-nek ilyenkor tranzakcióazonosítót kell generálnia az általa küldendő DNS kéréshez. Ellenőriztük a `ne_mesg.c` fájlban található `mesg_id()` nevű függvényt, amyely egy `id` nevű statikus változót tartalmaz:

```
uint16_t mesg_id(void) {
    static uint16_t id = 0;

    if (!id) {
        srandom (time (NULL));
        id = random ();
    }
    id++;

    if (T.debug > 4)
        syslog (LOG_DEBUG,"mesg_id() = %d",id);
    return id;
}
```

A programozó szándéka nyilvánvalónak tűnik: „válasszuk meg a statikus változó kezdőértékét véletlenszerűen, és növeljük az értékét 1-gyel minden végrehajtáskor”. De a fenti C program tényleges viselkedése „egy kicsit” más. Amikor az `id` nevű statikus változó értéke 0xffff, és az értékét 1-gyel megnöveljük, akkor az értéke 0 lesz, és a függvény azt minden további nélkül visszaadja. De a függvény következő végrehajtásakor nem 1-gyel fogja azt növelni, hanem ismét véletlenszerű értéket választ, ami kicsi (de pozitív) valószínűséggel 0xffff-hez közel lehet (akár 0xffff is lehet). Ez azt jelenti, hogy egy új DNS kérés tranzakcióazonosítója megegyezhet egy nemrég elküldött, és még válaszra váró DNS kérés tranzakcióazonosítójával. Ez problémát okozhat, amikor a válasz alapján a TOTD előkeresi a neki megfelelő kérést. Készítettünk egy gyors hibajavítást, amelyben ellenőrizzük, hogy a növelés után vajon nem lett-e 0 az `id` nevű statikus változó értéke, és ha igen, akkor azonnal megnöveltük azt 1-re. Ezzel elkerültük a kockázatos (ismételt) véletlenszám-generálást. A kijavított kód:

```
uint16_t mesg_id(void) {
    static uint16_t id = 0;

    if (!id) {
        srandom (time (NULL));
        id = random ();
    }
    if ( !++id ) ++id; /* correction for id==0 */

    if (T.debug > 4)
        syslog (LOG_DEBUG,"mesg_id() = %d",id);
    return id;
}
```

A fenti módosítás után a TOTD többé nem produkálta a korábbi hibát.

Megjegyzés: TOTD úgy működik, hogyha nem kap választ a „forwarder”-ként beállított névkiszolgálótól, akkor a következővel próbálkozik (ha van). A programozási hibát tartalmazó verzió a „forwarder”-ként beállított névkiszolgáló hibáját feltételezve a működését ezért megfelelő time-out letelte után folytatta (az egyetlen beállított névkiszolgáló használatával).

5.6.2. A biztonsági rés és annak kijavítása

A sebezhetőség

Mivel a TOTD szekvenciális tranzakcióazonosítókat használ, amelynek előrejelzése triviális, ezért a TOTD sebezhető a *tranzakcióazonosító előrejelzésen alapuló DNS cache „mérgezés” támadás*ra (DNS cache poisoning with transaction ID prediction attack) nézve. Ez a támadás úgy működik, hogy a támadó ráveszi az áldozat DNS kiszolgálót, hogy egy adott szimbolikus névre vonatkozó névfeloldási kérést küldjön ki, melyre a támadó még az authoritatív névkiszolgáló válaszának megérkezése előtt válaszol (hamis adatokkal). Ha a tranzakcióazonosító előrejelzése sikeres, akkor az áldozat DNS kiszolgáló elfogadja a támadó hamis válaszát. (A támadásról bővebb leírás található az RFC 5452 4. részében.) A legelterjedtebben használt BIND névkiszolgáló 8.2 előtti verzióiban is megvolt ez a sebezhetőség. A 8.2 verzióban vezették be az álvéletlen tranzakció ID-ket, és további fejlesztések voltak a 9-es verzióban, de elvileg még az is támadható [68] szerint.

Az elvi megoldás

Úgy döntöttünk, hogy mi is álvéletlen tranzakcióazonosítókat fogunk használni. Azonban az egyedileg generált álvéletlen tranzakcióazonosítók használata megkövetelte volna, hogy nyilvántartsuk, hogy mely tranzakcióazonosítók vannak még használatban. Ehhez viszont több helyen kellett volna módosítanunk a TOTD forrását (például nyilvántartásba venni a tranzakcióazonosítót, amikor a TOTD egy új kérést küld, és törölni a nyilvántartásból a tranzakcióazonosítót, amikor megjött a kérésre a válasz, vagy letelt a time-out). Ezért inkább az előre generált álvéletlen permutációk használatát választottuk, hogy a változásokat egyetlen forrásfájlon belül tudjuk tartani. Két további megfontolást tettünk még:

- Ha az álvéletlen permutációk mind a 65536 elemét felhasználnánk, akkor a tartomány kimerüléséhez közeledve egyre nagyobb valószínűséggel előrejelezhetők lennének a tranzakcióazonosítók.
- Egy adott permutáció néhány utolsó eleme még használatban lehet, amikor egy új permutáció elemeit kezdjük használni (így ismét ütközés fordulhatna elő).

Ezért a megoldásunkba két diszjunkt tartományt (0-0x7fff és 0x8000-0xffff) felváltva használunk, és a generált álvéletlen permutációknak mindig csak a felét használjuk fel. Ez a megoldás az eredetihez képest lényegesen javítja a biztonsági helyzetet, és csak kis mennyiségű programozói munkát igényel (konkrétan két függvényt egyetlen forrásfájlon belül). A megoldás ára továbbá aránylag kis mennyiségű számítási teljesítmény „elpazarlása”, hiszen az előállított véletlen permutációk elemeinek felét nem használjuk fel.

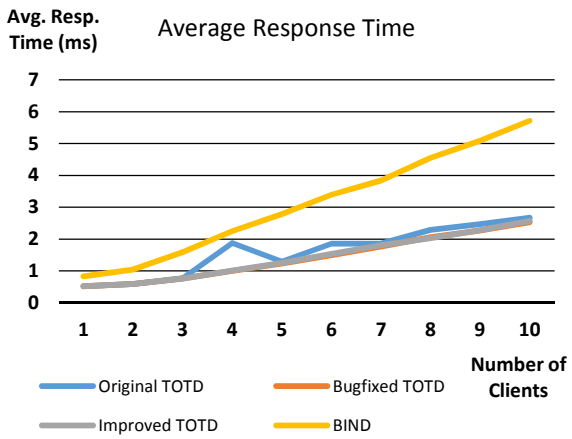
Az implementáció

A véletlen permutációk generálására a Durstenfeld által publikált algoritmus [69] *kifordított* (inside out) változatát használtuk. Míg az eredeti változat egy tömb elemeiből helyben készít véletlen permutációt, és így a tömb előzetes inicializálását igényli, a kifordított változat mindkét feladatot elvégzi egyetlen ciklusban. Mindkét változat komplexitása *O(N)*, de a kifordított változat megtakarítja az inicializálást és az elemek felcserélését.

A módosított `mesg_id()` függvény és az általa használt `make_random_permutation()` függvény megjegyzésekkel ellátott C forráskódja megtalálható [67] cikkünkben (a folyóirat open access).

Teljesítményvizsgálat

Az implementáció hatékonyságának ellenőrzésére összehasolítottuk az eredeti TOTD, a programozási hiba kijavítását tartalmazó TOTD, a biztonsági rést kiküszöbölését tartalmazó TOTD és a (forwarderként használt) BIND DNS64 implementációk teljesítményét. A méréseket a korább cikkünkben [32] megadotthoz hasonló módon, de azzal az eltéréssel végeztük, hogy most a `host` parancs `-t AAAA` opciójának használatával csak az AAAA rekordokat kértük le (enélkül az opció nélkül a `host` parancs az MX rekordot is lekéri), illetve a kliensek számát 1-től 10-ig egyesével növeltük (azért, hogy jól ábrázolható grafikont kapjunk). Itt most csak az egy kísérlet (256 db `host -t AAAA` parancs) végrehajtásának átlagos idejét hasonlítjuk össze az 50. ábrán, de az összes mért



50. ábra: Egy kísérlet átlagos végrehajtási ideje (a kisebb érték a jobb) [67]

érték táblázatosan is megtalálható [67] cikkünkben. Az ábrán az eredeti TOTD grafikonjának hullámzása teljesen esetleges, hiszen a TOTD véletlenszerűen előforduló hibája okozza. (Egy másik mérésben másként nézne ki a grafikon.) A programozási hiba kijavítását tartalmazó TOTD (bugfixed TOTD) grafikonját majdnem teljesen eltakarja a biztonsági rés kiküszöbölését tartalmazó TOTD (improved TOTD) grafikonja, ami azt jelenti, hogy ezek átlagos teljesítménye közt nincs

lényegi különbség. A BIND teljesítménye lényegesen elmarad ezeknek a teljesítményétől (10 kliensnél a BIND-dal egy kísérlet átlagos végrehatási ideje több, mint kétszerese a TOTD javított verzióival mért átlagos végrehajtási időnek).

Összefoglalva megállapítjuk, hogy a biztonsági rés kijavítását tartalmazó TOTD egy jól használható, és a tesztelt körülmények között a BIND-nál lényegesen hatékonyabb megoldásként olyan kiváló DNS64 implementáció, amelynek használatát éles környezetben is javasoljuk. A TOTD-nek az általunk készített javítást is tartalmazó 1.5.3. verziója forráskódban elérhető [70].

5.7. 6to4 implementációk stabilitása és teljesítménye

Felmerül a kérdés, hogy egyáltalán miért foglalkozunk 6to4 implementációk teljesítményének vizsgálatával, amikor a 6to4 technológia alkalmazása esetén biztonsági problémák vannak? Egyrészt azért, mert kényelmi és részben költség szempontok miatt (nem kell explicit tunnelt beállítani, használni, esetleg venni) mégis használják őket, másrészt pedig azért, mert a manuálisan konfigurált tunnel megoldásoknál is nagyon hasonló feladatot kell megvalósítani, így eredményeink valószínűleg ott is hasznosak lesznek.

A témával kapcsolatos kutatási eredményeinkről megjelent első cikkünkben [16] megmutattuk, hogy a 6to4 megoldást illetően a relayek teljesítménye lehet kritikus, hiszen azokon nagyszámú felhasználó forgalma haladhat keresztül (ráadásul ez a szám előre nem tervezhető), míg egy 6to4 routernek csak a „mögötte" levő IPv6 eszközök forgalmát kell kiszolgálnia. Ebben a cikkünkben a Linux sit, a FreeBSD stf és a NetBSD stf interfészek, mint 6to4 relay implementációk teljesítőképességét és stabilitását vizsgáltuk meg. Második (jelenleg megjelenés alatt álló) cikkünkben [71] pedig a Linux sit, OpenWrt sit és FreeBSD stf interfészeket teszteltünk. Már készül egy harmadik cikkünk is, amelyben a második cikkünkben szereplő implementációkkal azonos körülmények között teszelt (így velük összehasonlítható) Linux v4tunnel és NetBSD stf interfészek eredményei is szerepelnek [72]. Mivel ezt még nem publikáltuk, ezért a második cikkünk alapján mutatjuk be a 6to4 implementációk vizsgálatával kapcsolatos eredményeinket.

5.7.1. Más kutatók eredményei

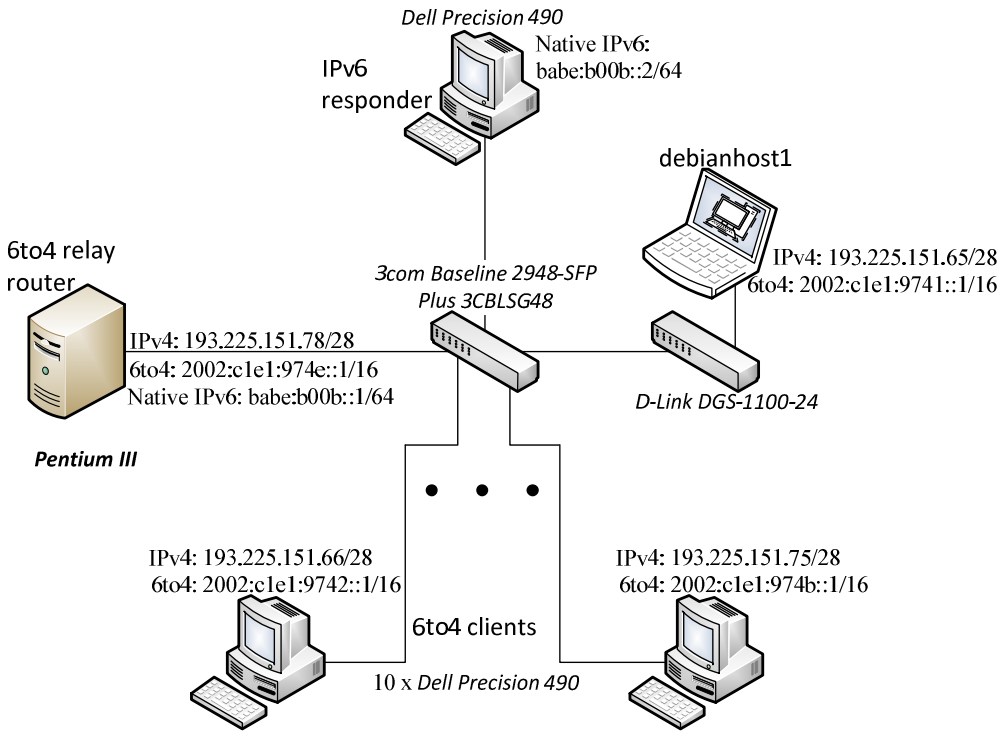
A 6to4 megoldás teljesítményével több cikk is foglalkozik. Az egyikben [73] egy teszthálózaton (két IPv6 sziget IPv4 hálózat fölött összekötve) a 6to4 megoldás teljesítmény jellemzőit (round trip time és througput) hasonlították össze azzal, amikor az adott hálózaton 6to4 nélkül, natív IPv4 illetve IPv6 protokoll használata mellett mérték meg ugyanazokat a jellemzőket. A vizsgálatokat a végpontok között TCP és UDP használatával is elvégezték. A teszthálózatban Cisco routereket használtak. Egy másik cikkben [74] az előre konfigurált tunnel és a 6to4 teljesítményét hasonlították össze egy egyszerű teszthálózaton (két kliens gép között két 6to4 router számítógéppel megvalósítva), és mindegyiket kétféle Linux (Fedora 9.10 és Ubuntu 11.0) és kétféle Windows (Server 2003 és 2008) esetén is megvizsgálták mind TCP-vel, mind UDP-vel elvégezve a méréseket. Azonban a Linux esetén használt interfészek típusát sajnos nem közölték. Egy harmadik cikkben [75] az előzőhöz hasonló vizsgálatokat végeztek, de csak Windows alatt (Server 2008 és 2012).

A fenti cikkek közös jellemzője, hogy szimmetrikus hálózaton, két darab 6to4 router használatával mértek. Sajnos olyan cikket nem találtunk, ahol különféle, név szerint megjelölt szabad szoftver 6to4 relay implementációk teljesítményét hasonlították volna össze egymással.

5.7.2. A teszt környezet

Mérési összeállítás

Az egyes 6to4 implementációk stabilitásának és teljesítményének a vizsgálatához használt hálózatot a 51. ábrán mutatjuk be. Mivel izolált környezetben mértünk, ezért tetszőleges IPv4 és IPv6 címeket használhattunk. Az összeállítás központi eleme az ábra bal oldalán található 6to4 relay router (egy Pentium III számítógép) volt. A terhelést az ábra alján található 10db Dell Precision 490 munkaállomás szolgáltatta. Ezek a gépek 6to4 hostként működtek: az IPv6 kliensek forgalmát IPv4 csomagokba ágyazták be. Az általuk küldött csomagokat a 3Com switch a 6to4 relaynek továbbította, ami kibontotta a beágyazott IPv6 csomagot és továbbította az ábra tetején található gép felé. Az összes címzett helyett ez a gép válaszolt; ehhez NAT66 segítségével magára irányította a csomagokat. A válaszokat a 6to4 relay ismét beágyazta, majd továbbította a megfelelő 6to4 hostnak. A laptop a mérések vezérlését látta el. Az egyes gépeken az IP-címeket az 51. ábrának megfelelően állítottuk be.



51. ábra: A 6to4 relay implementációk vizsgálatához használt teszthálózat [71]

Hardver és szoftver konfiguráció

A 6to4 relay router egy 800MHz-es Intel Pentium III számítógép volt 128MB memóriával és két darab TP-LINK TG-3269 Gigabit Ethernet hálózati kártyával. A Dell Precision 490 munkaállomásokban pedig 2 db kétmagos Intel Xeon 5130 2 GHz CPU üzemelt. (A gépek pontos konfigurációja megtalálható: [71] cikkünkben, további részletek pedig Horváth Viktor szakdolgozatában [76].) Ami az operációs rendszert illeti, minden kliens gépen Debian Squeeze 6.0.7 GNU/Linux volt, a válaszoló gépre pedig OpenBSD 5.3 operációs rendszert telepítettük. (Az utóbbira a NAT66-hoz volt

szükség.) Az egyes 6to4 relay implementációk teszteléséhez használt gépre mindig az adott implementációnak megfelelő operációs rendszer került.

A tesztelt 6to4 implementációk

A következő 6to4 implementációkat teszteltük, a megadott operációs rendszerek alatt:

- Debian 7.1.0_x86 – sit
- OpenWRT 12.09_x86 – sit
- FreeBSD 9.1_x86 – stf.

A kliens gépeken pedig mindig a Linux sit interfészt használtuk a 6to4 host funkció megvalósításához.

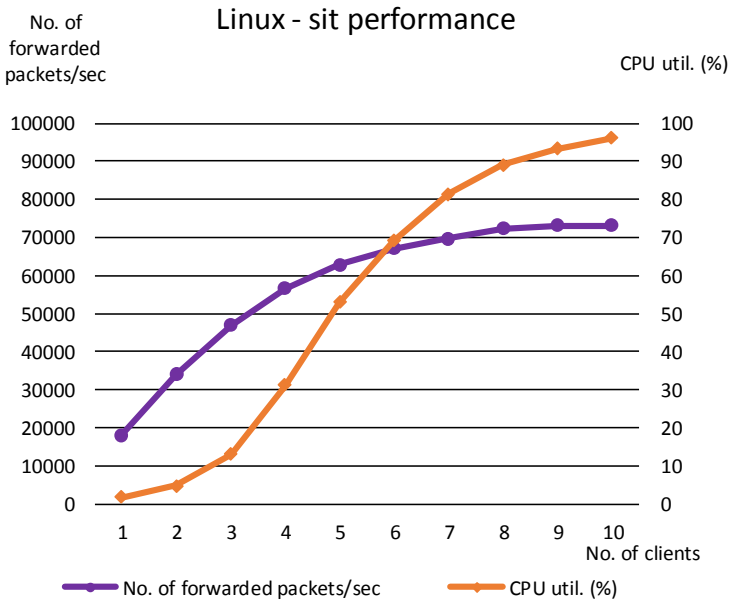
5.7.3. A 6to4 mérési módszer

A mérésekhez a terhelést a kilenseken az alábbi scripttel állítottuk elő:

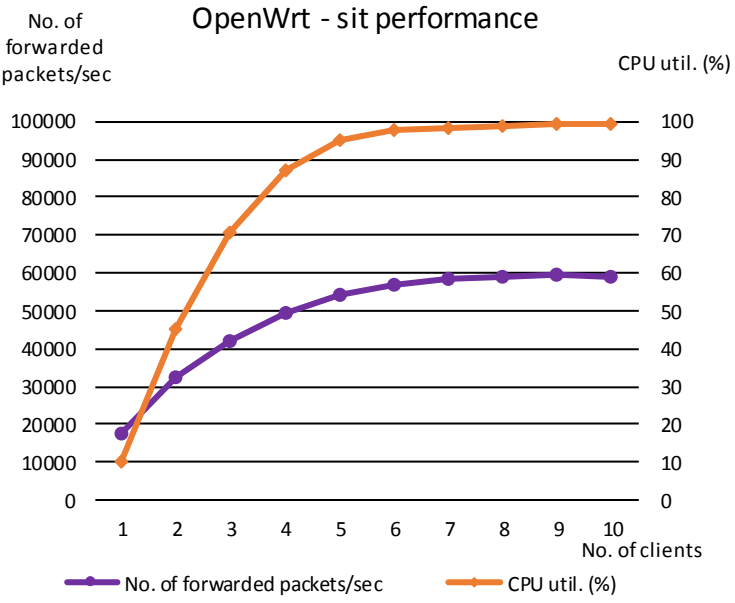
```
#!/bin/bash
i=`cat /etc/hostname | grep -o '[0-9]'`
for b in {0..255}
do
    rm -rf $b
    mkdir $b
    for c in {0..252..4}
    do
        ping6 2001:738:2c01:8000::193.$i.$b.$c -c8 -i0 \
            >> $b/6to4-193-$i-$b-$c &
        ping6 2001:738:2c01:8000::193.$i.$b.$c -c8 -i0 \
            >> $b/6to4-193-$i-$b-$c &
        ping6 2001:738:2c01:8000::193.$i.$b.$((c+1)) -c8 -i0 \
            >> $b/6to4-193-$i-$b-$((c+1)) &
        ping6 2001:738:2c01:8000::193.$i.$b.$((c+1)) -c8 -i0 \
            >> $b/6to4-193-$i-$b-$((c+1)) &
        ping6 2001:738:2c01:8000::193.$i.$b.$((c+2)) -c8 -i0 \
            >> $b/6to4-193-$i-$b-$((c+2)) &
        ping6 2001:738:2c01:8000::193.$i.$b.$((c+2)) -c8 -i0 \
            >> $b/6to4-193-$i-$b-$((c+2)) &
        ping6 2001:738:2c01:8000::193.$i.$b.$((c+3)) -c8 -i0 \
            >> $b/6to4-193-$i-$b-$((c+3)) &
        ping6 2001:738:2c01:8000::193.$i.$b.$((c+3)) -c8 -i0 \
            >> $b/6to4-193-$i-$b-$((c+3))
        done
done
```

Az előzetes mérések tapasztalatai alapján a scriptet úgy hangoltuk be, hogy még a leghatékonyabb 6to4 implementáció esetén is közel 100% legyen a 6to4 relay terhelése 10 kliens mellett. A programban az `i` változó tartalmazza az aktuális kliens sorszámát. A külső `for` ciklus 256-szor fut le (0-tól 255-ig), a belső pedig 64-szer (0-tól 252-ig 4-es lépésközzel). A ciklus magja 4 pár, azaz 8 darab `ping6` utasítást tarlamaz (bár a gépekben csak 4 db CPU mag volt, ez a megoldás nagyobb terhelést biztosított, mintha csak 4 darab `ping6` utasítást használtunk volna). Mindegyik `ping6` utasítás 8-8 darab ICMPv6 echo request üzenetet küldött, köztük közel 0 időintervallummal. A `ping6` utasítások az első hetet aszinkron módon indítottuk el, így a nyolc utasítás konkurrens módon futott, de a belső ciklus újabb iterációja csak a nyolcadik `ping6` utasítás lefutása után indulhatott el. Így minden kliens 256*64*8*8= 1048576 ICMP echo request üzenetet küldött ki, összesen 256*64*4=

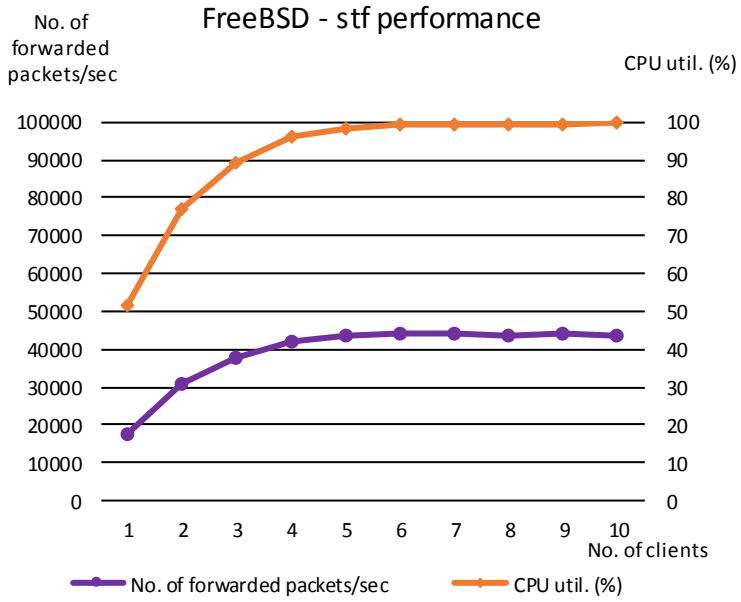
65536 különböző cél IP-cím irányába. Egy méréssorozatban a mérésben részt vevő kliensek számát egyesével növeltük 1-től 10-ig. A 6to4 relayen a CPU és memória használatát a `vmstat` paranccsal mértük. Annak érdekében, hogy a `vmstat` nagy terhelés mellett is megfelelően működjön, „-10”-es nice értéket használtunk.



52. ábra: Linux sit: másodpercenként továbbított csomagok száma és processzorhasználat [71]



53. ábra: OpenWrt sit: másodpercenként továbbított csomagok száma és processzorhasználat [71]



54. ábra: FreeBSD stf: másodpercenként továbbított csomagok száma és processzorhasználat [71]

5.7.4. A 6to4 mérések eredményei

A mérési eredmények táblázatos formában megtalálhatók [71] cikkünkben. A táblázatokban ugyanazok a mért jellemzők szerepelnek, mint amiket a NAT64 implementációk teljesítményének ICMP-vel való vizsgálatakor láttunk, viszont itt implementációnként 10 méréssorozatot végeztünk, ezért a táblázatok nem férnének el jelen könyv lapszélességében. Így az egyes implementációk teljesítményét ábrák segítségével mutatjuk be. Minden implementáció külön ábrán szerepel, és a másodpercenként továbbított csomagszám mellett feltüntetjük a processzorigény értékét is.

A Linux sit interfész teljesítményét az 52. ábrán láthatjuk. A másodpercenként átvitt csomagok száma kezdetben még majdnem duplázódik (1 kliensnél 18051 csomag/s, 2 kliensnél 33953 csomag/s), de aztán a görbe egyre inkább telítődést mutat, a végén kis mértékben csökken is (9 kliensnél 73129 csomag/s, 10 kliensnél 73050 csomag/s). A processzorigény kezdetben nagyon alacsony, de aztán a linárisnál lényegesen erősebben növekszik (1-5 kliensig rendre: 1,8%, 4,8%, 12,9%, 31,2% és 53%). Öt kliensnél a görbének inflexiós pontja van, és – feltehetőleg a CPU kapacitás korlátozott volta miatt – a görbe meredeksége csökken.

Az OpenWrt sit interfész teljesítményét az 53. ábrán láthatjuk. Egy kliens mellett ennek a teljesítmény nagyon közel van a Linux sit teljesítményéhez (17595 csomag/s), de aztán fokozatosan lemarad tőle, és a 60000 csomag/s értéket sohasem éri el (a maximum értéke 59332 csomag/s 9 kliens mellett). A processzorigény lényegesen magasabb értékről indul (1 kliens: 10,1%), a kliensszámmal arányosnál lényegesen erősebben nő (2 kliens: 45%), aztán hamarosan telítődik, és aszimptotikus jelleggel közelíti a 100%-ot.

A FreeBSD stf interfész teljesítményét a 54. ábrán láthatjuk. Egy kliens mellett ennek a teljesítménye gyakorlatilag megegyezik az OpenWrt teljesítményével (17594 csomag/s), de aztán fokozatosan lemarad tőle és a 44000 csomag/s értéket sohasem éri el (a maximum értéke 43970 csomag/s

9 kliens mellett, de a 6-10 kliens tartományban lényegében fluktuál). A processzorigény még magasabb értékről indul (1 kliens: 51,5%), és kezdettől fogva a telítődési tartományban van (2-4 kliensnél rendre: 77,1%, 89%, 96,4%).

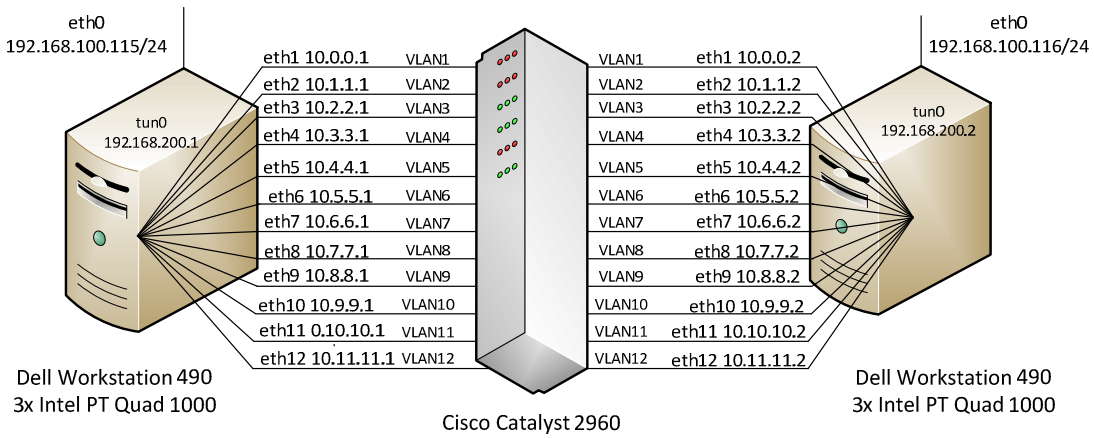
Összefoglalásként megállapítjuk, hogy mindhárom implementáció stabilan viselkedett, de a teljesítményükben lényeges különbség mutatkozott. Alacsony tehelés (1 kliens) esetén a másodpercenként átvitt csomagok számában nem volt lényeges különbség, de a processzorigényben már nagyon jelentős volt az eltérés (Linux sit: 1,8%, OpenWrt sit: 10,1%, FreeBSD stf 51,5%). Nagyobb terhelésnél már megmutatkozott a különbség az egyes implementációk teljesítőképessége között, csúcsteljesítményük 9 kliens mellet: Linux sit: 73129 csomag/s, OpenWrt sit: 59332 csomag/s, FreeBSD stf 43970 csomag/s. Bár az eltérés jelentős, a teljesítmények aránya messze nem tükrözi azt az arányt, amit az 1 kliensnél tapasztalt processzorigények aránya alapján vártunk volna. A jelenség okának vizsgálata érdekes feladat, de meghaladja méréseink célját. A képet még tovább árnyalja az egyes implementációk csomagveszési aránya. Ebben jellemzőben ugyanis a FreeBSD stf volt legjobb, csomagvesztése mindvégig 0,02% alatt maradt, míg 10 kliensnél az OpenWrt sit csomagvesztése 0,089%, a Linux sit csomagvesztése pedig a 0,061% volt.

A három vizsgált 6to4 implementáció küzül a legjobb átviteli teljesítményt a Linux sit érte el, de szükség esetén bármelyik implementáció használható, mert mindegyik stabilan viselkedett.

5.8. Az MPT könyvtár teljesítőképessége

Az MPT hálózati szintű többutas kommunikációs könyvtár teljesítőképességét az út aggregáció hatékonysága szempontjából vizsgáltuk [77]. Mivel az IPv6 áttérési technológiák között az MPT elsősorban, mint rugalamas alagútképzési protokoll érdekes (bármelyik IP verzió fölött bármelyik IP verzió átvihető vele), ezért ezeket az eredményeinket csak érintőlegesen mutatjuk be.

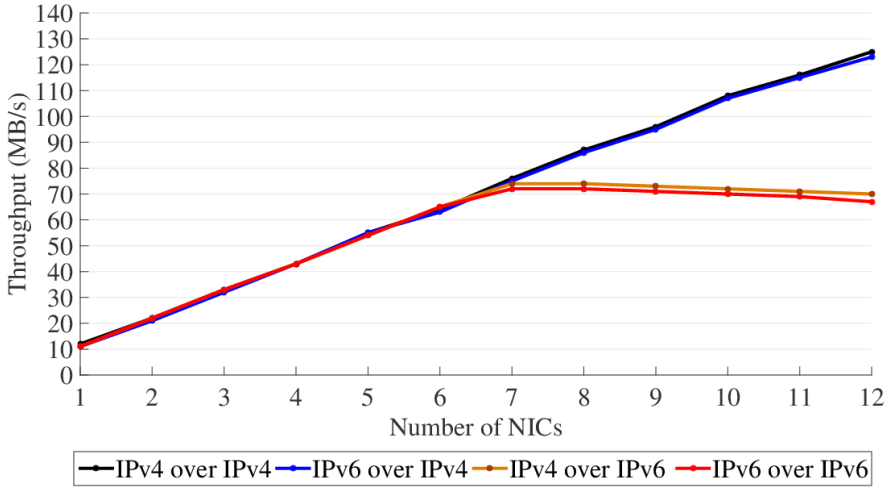
A mérések során mind a négy lehetséges esetet megvizsgáltuk: IPv4 alagút IPv4 fölött, IPv6 alagút IPv4 fölött, IPv4 alagút IPv6 fölött és IPv6 alagút IPv6 fölött. A mérési összeállítást az 55. ábrán mutatjuk be, amely az első esetet ábrázolja (IPv4 alagút IPv4 fölött). Részletek a [77] cikkünkben.



55. ábra: Az MPT teszthálózat (IPv4 alagút IPv4 fölött) [77]

Számos esetet megvizsgáltunk, a méréseket UDP és TCP protokoll fölött egyaránt elvégeztük, az MPT könyvtárnak a 32-bites és a 64-bites verzióját is teszteltük. Most csak a 32-bites verzió UDP fölötti vizsgálatának eredményeit mutatjuk be. Ezekhez a mérésekhez az `iperf` hálózati tesztprogramot használtuk. Eredményeink az 56. ábrán láthatók. Rögtön feltűnik, hogy a felső (alagútként használt) protokoll verziójától függetlenül, az alsó protokoll verziójának van döntő hatással a csa-

tornakapacitás összegzésének a hatékonyságára. Amikor az alsó protokoll IPv4 volt, a csatornakapacitás lényegében lineárisan nőtt. Amikor pedig IPv6 volt, akkor 7 hálózati interfész volt a lineáris növekedés határa, utána az átviteli kapacitás nem nőtt tovább. Azt is megmutattuk, hogy ez a határ nem az MPT valamilyen belső tulajdonsága, hanem a számítógépek erőforrásából következik (a processzorok cseréjével a határt sikerült megemelni).



56. ábra: Az MPT csatornakapacitás összegzésének hatékonysága (32-bit, iperf) [77]

Terveink között szerepel, hogy országos méretű hálózaton (Győr-Budapest-Debrecen viszonylatban) összehasonlítjuk az MPT-vel létrehozott különféle alagutak teljesítményét egymással és a natív IPv4 illetve natív IPv6 hálózatok teljesítményével.

6. Irodalomjegyzék

[1] Lencse Gábor: Számítógép-hálózatok, 2. kiadás, Universitas-Győr Nonprofit Kft., Győr, 2008. ISBN: 978-963-9819-15-3

[2] J. F. Kurose, K. W. Ross: Számítógép-hálózatok működése - Alkalmazásorientált megközelítés, Panem Kiadó, Budapest, 2008. ISBN: 978-9-635454-98-3

[3] A. S. Tanenbaum, D. J. Wetherall: Számítógép-hálózatok, 3. kiadás, Panem Kiadó, Budapest, 2012. ISBN: 978-963-545-529-4

[4] Cisco Systems, *Configuring BGP on Cisco Routers*, Student Guide, vol. 1, version 3.2, 2005.

[5] G. Lencse and I. Derka, "Investigation of the fault tolerance of the PIM-SM IP multicast routing protocol for IPTV purposes", *Infocommunications Journal*, vol. 5, no. 1. (March, 2013) pp. 21-28.

[6] B. Almási, A. Harman, "An overview of the multipath communication technologies", in *Proc. Conference on Advances in Wireless Sensor Networks 2013 (AWSN 2013)*, Debrecen University Press, Debrecen, Hungary, 2013. ISBN: 978-963-318-356-4, pages 7-11.

[7] S. Ferdous, F. Chowdhury and J. C. Acharjee, "An extended algorithm to enhance the performance of the current NAPT", in *Proc. International Conference on Information and Communication Technology 2007* (ICICT'07), Dhaka, Bangladesh, March 7-9, 2007, pp. 315-318, DOI: 10.1109/ICICT.2007.37540

[8] G. Lencse, "Estimation of the port number consumption of web browsing", *IEICE Transactions on Communications,* vol. E98-B, no. 8. (August, 2015) pp. 1580-1588. DOI: 10.1587/transcom.E98.B.1580

[9] M. Boye, "Netfilter connection tracking and NAT implementation", in *Proceedings of Seminar on Network Protocols in Operating Systems*, Department of Communications and Networking, Aalto University, 2013, pp. 34-39. http://urn.fi/URN:ISBN:978-952-60-4997-7

[10] S. Répás, P. Farnadi and G. Lencse, "Performance and stability analysis of free NAT64 implementations with different protocols", *Acta Technica Jaurinensis*, vol. 7, no 4, (October, 2014), pp. 404-427. DOI: 10.14513/actatechjaur.v7.n4.340

[11] G. Lencse and A. G. Soós, "Design of a tiny multi-threaded DNS64 server", in *Proc. 8th International Conference on Telecommunications and Signal Processing (TSP 2015)*, Prague, Czech Republic, July 9-11, 2015, Brno University of Technology, pp. 27-32.

[12] D. Liu, H. Deng, "NAT46 considerations draft-liu-behave-nat46-02", [Online]. Available: https://tools.ietf.org/html/draft-liu-behave-nat46-02

[13] A. Yourchenko, "OpenWRT feed with stateless NAT46 kernel module", [Online]. Available: https://github.com/ayourtch/nat46

[14] Bob Weeink, "Cisco ASA NAT46 (IPv4-to-IPv6)", [Online]. Available: https://supportforums.cisco.com/discussion/11809096/cisco-asa-nat46-ipv4-ipv6

[15] Brocade, *Brocade ServerIron ADX NAT64 Configuration Guide: Supporting Brocade ServerIron ADX version 12.5.02*, Part Number: 53-1003448-01 [Online]. Available: http://www.brocade.com/downloads/documents/html_product_manuals/SI_12502_NAT64/

[16] G. Lencse and S. Répás, "Performance analysis and comparison of 6to4 relay implementations", *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 9. (September, 2013), pp. 13-21. DOI: 10.14569/IJACSA.2013.040903

[17] B. Almási, Sz. Szilágyi, "Throughput performance analysis of the multipath communication library MPT", in *Proc. 36th International Conference on Telecommunications and Signal Processing (TSP 2013)*, Rome, Italy, July 2-4, 2013, pp. 86-90. DOI: 10.1109/TSP.2013.6613897

[18] B. Almási, Sz. Szilágyi, "Multipath FTP and stream transmission analysis using the MPT software environment", *International Journal of Advanced Research in Computer and Communication Engineering*, Vol. 2, No. 11, (November 2013) pp. 4267-4272.
[19] B. Almási, "A simple solution for wireless network layer roaming problems", *Carpathian Journal of Electronic and Computer Engineering*, vol. 5, no. 1, (2012) pp. 5-8.
[20] B. Almási, "A solution for changing the communication interfaces between WiFi and 3G without packet loss", In *Proc. 37th International Conference on Telecommunications and Signal Processing (TSP 2014)*, Berlin, Germany, July 1-3, 2014, pp. 73-77.
[21] E. Crabbe, L. Yong, X. Xu, T. Herbert, "GRE-in-UDP encapsulation", IETF Draft, [Online]. Available: https://tools.ietf.org/html/draft-ietf-tsvwg-gre-in-udp-encap
[22] B. Almási, G. Lencse, "Investigating the multipath extension of the GRE in UDP technology", unpublished
[23] L. Bokor, G. Jeney, "IPv4 / IPv6 coexistence and transition: concepts, mechanisms and trends" In: Katalin Tarnay, Gusztáv Adamis, Tibor Dulai (Ed.), *Advanced Communication Protocol Technologies: Solutions, Methods, and Applications*, Hershey: IGI Global, Information Science Reference, 2011. pp. 156-177.
[24] L. Bokor, Z. Kanizsai, G. Jeney, "IMS-centric evaluation of IPv4/IPv6 transition methods in 3G UMTS systems", *International Journal on Advances in Networks and Services*, vol. 3 no 3 & 4, (2010) pp. 402-416.
[25] L. Bokor, V. Simon, I. Dudás, S. Imre, "Anycast subnet optimization for efficient IPv6 mobility management", In: *Proc. First International Global Information Infrastructure Symposium (IEEE GIIS 2007)*, (Marrakech, Morocco, July 2-6, 2007), IEEE Press, New York, pp. 187-190 DOI: 10.1109/GIIS.2007.4404188
[26] M. Nikkhah, R. Guerin, "Migrating the internet to IPv6: An exploration of the when and why", to appear in *IEEE/ACM Transaction on Networking*, [Online]. Available: http://www.seas.upenn.edu/~mnikkhah/Migrating-the-Internet-02252015.pdf
[27] N. Skoberne, O. Maennel, I. Phillips, R. Bush, J. Zorz, M. Ciglaric, "IPv4 address sharing mechanism classification and tradeoff analysis", *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 391-404. DOI: 10.1109/TNET.2013.2256147
[28] Free Software Foundation, "The free software definition", [Online]. Available: http://www.gnu.org/philosophy/free-sw.en.html
[29] Open Source Initiative, "The open source definition", [Online]. Available: http://opensource.org/docs/osd
[30] Cisco, "End user license agreement", [Online]. Available: http://www.cisco.com/en/US/docs/general/warranty/English/EU1KEN_.html
[31] Juniper Networks, "End user license agreement", [Online]. Available: http://www.juniper.net/support/eula.html
[32] G. Lencse and S. Répás, "Performance analysis and comparison of different DNS64 implementations for Linux, OpenBSD and FreeBSD", in *Proc. IEEE 27th International Conference on Advanced Information Networking and Applications (AINA 2013)*, Barcelona, Spain, March 25-28, 2013, pp. 877-884. DOI: 10.1109/AINA.2013.80
[33] S. Répás, T. Hajas and G. Lencse, "Application compatibility of the NAT64 IPv6 transition technology", in *Proc. 37th International Conference on Telecommunications and Signal Processing (TSP 2014)*, Berlin, Germany, July 1-3, 2014, Brno University of Technology, pp. 49-55.
[34] N. Škoberne and M. Ciglarič, "Practical evaluation of stateful NAT64/DNS64 translation" *Advances in Electrical and Computer Engineering*, vol. 11, no. 3, (August 2011), pp. 49-54. DOI: 10.4316/AECE.2011.03008
[35] S. Perreault, J.-P. Dionne, and M. Blanchet, "Ecdysis: open-source DNS64 and NAT64", *AsiaBSDCon*, 2010, http://viagenie.ca/publications/2010-03-13-asiabsdcon-nat64.pdf

[36] V. Bajpai, N. Melnikov, A. Sehgal and J. Schönwälder, "Flow-based identification of failures caused by IPv6 transition mechanisms" in *Dependable Network Services: Proc. 6th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security (AIMS 2012)*, Luxembourg, Luxembourg, June 4-8, 2012, Springer LNCS, vol. 7279. pp. 139-150. DOI: 10.1007/978-3-642-30633-4_19
[37] X. Deng, L. Wang, T. Zheng, D. Gu, E. Burgey, "Analysis on IPv6 transition solutions and service tests", in *Proc. ICNS 2011: The Seventh International Conference on Networking and Services*, Venice/Mestre, Italy, May 22-27, 2011, IARIA, pp. 85-91.
[38] J. Beall, "Scholarly open access: Critical analysis of scholarly open-access publishing", [Online]. Available: http://scholarlyoa.com/publishers
[39] "TAYGA: Simple, no-fuss NAT64 for Linux" [Online]. Available: http://www.litech.org/tayga/
[40] P. N. M. Hansteen, *The Book of PF: A No-Nonsense Guide to the OpenBSD Firewall*, 2nd ed., San Francisco: No Starch Press, 2010. ISBN: 978-1593272746
[41] Simon Perreault, "[Ecdysis-discuss] NAT64 in OpenBSD", [Online]. Available: http://www.viagenie.ca/pipermail/ecdysis-discuss/2011-October/000173.html
[42] Hajas Tamás, *NAT64 implementációk alkalmazásokkal való kompatibilitásának vizsgálata*, szakdolgozat, Széchenyi István Egyetem, MTK, Távközlési Tanszék, 2013.
[43] S. Miyakawa, "IPv4 to IPv6 transformation schemes", *IEICE Trans. Commun.*, Vol. E93-B, No. 5, May 2010, pp. 1078-1084. DOI: 10.1587/transcom.E93.B.1078
[44] F. Fourcot, B. Grelot, "Migrating to IPv6 with Address+Port translation", SLR Project Report, Telecom Bretagne, March 2010, [Online]. Available: https://svn.fperrin.net/v6fication/documentation/rapport-Fourcot-Grelot.pdf
[45] I. Kraemer, F. Perrin, "Impact of the lack of ports in a DS-Lite architecture", Project Report, Telecom Bretagne, March 18, 2011, [Online]. Available: https://svn.fperrin.net/v6fication/article/simulation.pdf
[46] G. Chen, W. Li, T. Tsou, J. Huang, T. Taylor, "Analysis of NAT64 port allocation methods for shared IPv4 addresses: draft-chen-sunset4-cgn-port-allocation-05", [Online]. Available: https://tools.ietf.org/html/draft-chen-sunset4-cgn-port-allocation-05
[47] S. Répás, T. Hajas and G. Lencse, "Port number consumption of the NAT64 IPv6 transition technology", in *Proc. 37th International Conference on Telecommunications and Signal Processing (TSP 2014)*, Berlin, Germany, July 1-3, 2014, Brno University of Technology, pp. 66-72.
[48] Garai Gábor, *IPv6 bevezetésének támogatása a Telenor hálózatában*, szakdolgozat, Széchenyi István Egyetem, MTK, Távközlési Tanszék, 2013.
[49] P. E. Román, J. D. Velásquez, V. Palade, L. C. Jain, "New trends in web user behaviour analysis" in *Advanced Techniques in Web Intelligence-2: Web User Browsing Behaviour and Preference Analysis*, Springer Berlin Heidelberg, 2013, pp. 1-10. DOI:10.1007/978-3-642-33326-2_1
[50] C. Liu, R. W. White, S. Dumais, "Understanding web browsing behaviors through Weibull analysis of dwell time", in *Proc. 33rd international ACM SIGIR conference on Research and development in information retrieval*, Geneva, Switzerland, July 19-23, 2010, ACM New York, NY, USA, 2010, pp. 379-386, DOI: 10.1145/1835449.1835513
[51] L. D. Catledgea, J. E. Pitkowb, "Characterizing browsing strategies in the world-wide web", *Computer Networks and ISDN Systems*, vol. 27, no. 6, (April, 1995) pp. 1065-1073, DOI: 10.1016/0169-7552(95)00043-7
[52] Alexa, "The top 500 sites on the web", [Online]. Available: http://www.alexa.com/topsites
[53] Statistic Brain, "Top websites by traffic", 2014, [Online]. Available: http://www.statistic-brain.com/top-us-websites-by-traffic/
[54] BBC News, "SuperPower: visualising the Internet", [Online]. Available: http://news.bbc.co.uk/2/hi/technology/8562801.stm

[55] Quantcast, "Top sites", Quantcast Corporation, [Online]. Available: https://www.quantcast.com/top-sites
[56] G. Lencse and G. Takács, "Performance analysis of DNS64 and NAT64 solutions", *Infocommunications Journal*, vol. 4, no 2, (June, 2012), pp. 29-36.
[57] G. Lencse and S. Répás, "Performance analysis and comparison of the TAYGA and of the PF NAT64 implementations", in *Proc. 36th International Conference on Telecommunications and Signal Processing (TSP 2013)*, Rome, Italy, July 2-4, 2013, Brno University of Technology, pp. 71-76. DOI: 10.1109/TSP.2013.6613894
[58] NIC Mexico, "Jool NAT64 implementation", [Online]. Available: https://www.jool.mx/
[59] K. J. O. Llanto and W. E. S. Yu, "Performance of NAT64 versus NAT44 in the context of IPv6 migration", in *Proc. International MultiConference of Engineers and Compuer Scientists 2012 (IMECS 2012)*, Hong Kong, March 14-16, 2012, vol. I, pp. 638-645.
[60] C. P. Monte et al, "Implementation and evaluation of protocols translating methods for IPv4 to IPv6 transition", *Journal of Computer Science & Technology*, ISSN: 1666-6038, vol. 12, no. 2, (August, 2012). pp. 64-70.
[61] S. Yu, B. E. Carpenter, "Measuring IPv4 – IPv6 translation techniques", Technical Report 2012-001, Department of Computer Science, The University of Auckland, January 2012.
[62] E. Hodzic, S. Mrdovic, "IPv4/IPv6 transition using DNS64/NAT64: Deployment issues", in *Proc. 2012 IX International Symposium on Telecommunications (BIHTEL)*, Sarajevo, Bosnia and Herzegovina, Oct. 25-27, 2012, DOI: 10.1109/BIHTEL.2012.6412066
[63] NTIA ITS, "Definition of 'graceful degradation'" [Online]. Available: http://www.its.bldrdoc.gov/fs-1037/dir-017/_2479.htm
[64] Internet Systems Consortium, "Berkeley Internet Name Daemon (BIND)", [Online]. Available: https://www.isc.org/software/bind
[65] Martin Dunmore (Ed.), *An IPv6 Deployment Guide*, The 6NET Consortium, September, 2005. [Online]. Available: http://www.6net.org/book/deployment-guide.pdf
[66] G. Lencse, S. Répás, "Performance analysis and comparison of four DNS64 implementations under different free operating systems", unpublished
[67] G. Lencse and S. Répás, "Improving the performance and security of the TOTD DNS64 implementation", *Journal of Computer Science & Technology*, ISSN: 1666-6038, vol. 14, no. 1, (April, 2014) pp. 9-15.
[68] A. Klein, "BIND8 DNS cache poisoning – And a theoretic DNS cache poisoning attack against the latest BIND 9", Trusteer, July-August 2007, [Online]. Available: http://packetstorm.wowhacker.com/papers/attack/BIND_8_DNS_Cache_Poisoning.pdf
[69] R. Durstenfeld, "Algorithm 235: Random permutation", *Communications of the ACM*, vol. 7, no. 7, (July, 1964) p. 420. DOI: 10.1145/364520.364540
[70] TOTD source code at GitHub, [Online]. Available: https://github.com/fwdillema/totd.git
[71] S. Répás, V. Horváth and G. Lencse, "Stability analysis and performance comparison of three 6to4 relay implementations", in *Proc. 38th International Conference on Telecommunications and Signal Processing (TSP 2015)*, Prague, Czech Republic, July 9-11, 2015, Brno University of Technology, pp. 82-87.
[72] S. Répás, V. Horváth and G. Lencse, "Stability analysis and performance comparison of five 6to4 relay implementations", unpublished
[73] N. Bahaman, E. Hamid, A. S. Prabuwono, "Network performance evaluation of 6to4 tunneling" in *Proc. 2012 International Conference on Innovation Management and Technology Research* (ICIMTR), Malacca, Malaysia, May 21-22, 2012, pp. 263-268. DOI: 10.1109/ICIMTR.2012.6236400
[74] S. Narayan, S. Tauch, "IPv4-v6 transition mechanisms network performance evaluation on operating systems", in *Proc. 3rd IEEE International Conference on Computer Science and Information*

Technology (ICCSIT 2010), Chengdu, China, Jul 9-11, 2010, pp. 664-668, DOI: 10.1109/ICCSIT.2010.5564141

[75] D. Hadiya, R. Save, G. Geetu, "Network performance evaluation of 6to4 and configured tunnel transition mechanisms: An empirical test-bed analysis", in *Proc. 6th International Conference on Emerging Trends in Engineering and Technology (ICETET 2013)*, Nagpur, India, December 16-18, 2013, IEEE Computer Society, pp. 56-60. DOI: 10.1109/ICETET.2013.14

[76] Horváth Viktor, *6to4 relay implementációk teljesítőképességének és stabilitásának vizsgálata*, szakdolgozat, Széchenyi István Egyetem, MTK, Távközlési Tanszék, 2013.

[77] G. Lencse and Á. Kovács, "Advanced measurements of the aggregation capability of the MPT multipath communication library", *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, vol. 4, no. 2. (2015), pp 41-48. DOI: 10.11601/ijates.v4i2.112