Faculteit Toegepaste Wetenschappen

Vakgroep Electronica en Informatiesystemen

Voorzitter: Prof. Dr. Ir. Jan M. Van Campenhout

Academiejaar 2001-2002

Migratie van IPv4 naar IPv6

Kristof Verhenne

Promotor : Prof. Dr. Ir. Koen De Bosschere

Begeleider: Dr. Ir. Michiel Ronsse

Scriptie ingediend tot het behalen van de academische graad van

Licentiaat in de Informatica

Dankwoordje

Voor we aan de tekst beginnen wens ik eerst nog enkele mensen te danken voor hun steun en medewerking. In het bijzonder wens ik mijn ouders te danken voor het dat ik mocht verder studeren en dat zij hierbij mijn niet altijd zo schitterende humeur tijdens stress- en examenperiodes doorstonden. Ik wens hen ook te danken voor alle moeite die zij voor mij gedaan hebben opdat ik het in bepaalde periodes wat makkelijker zou hebben.

Ik wens ook enkele vrienden te bedanken die oor hadden naar mij bekommernissen en bij wie ik steeds terecht kon met mijn verhaal. In het bijzonder wens ik Klaas te bedanken voor de moed die hij me insprak. Ik wens ook hen te bedanken die het aandurfden mijn tekst proef te lezen.

Verder wens ik ook de mensen op de vakgroep te bedanken die mij een onderwerp lieten doen waarin ik wel echt geïnteresseerd was en voor de accommodatie die ik hierbij ter beschikking kreeg. Ik wens ook de systeembeheerder te bedanken voor zijn vertrouwen: met mijn activiteiten werd wel degelijk een potentieel veiligheidsgat in de firewall gemaakt.

Ik wil de mensen van verschillende mailinglists bedanken voor hun hulp bij mijn soms ambetante vragen en alle andere mensen die mij geholpen hebben en die ik hier vergat te vermelden.

Ik wens ook de mensen te bedanken die mijn tekst doorzocht hebben op fouten. Dit zijn Michiel Ronsse, Andy Georges, Olivier Verhooghen, yanu, Bruno De Maesschalck, Rudy Gevaert en Elke Jansen.

Toelating tot bruikleen

Migratie van IPv4 naar IPv6
door
Kristof VERHENNE

Scriptie ingediend tot het behalen van de academische graad van
Licentiaat Informatica optie: Informatie- en Communicatietechnologie

Academiejaar 2001-2002

Promotor: Prof. Dr. Ir. Koen De Bosschere
Scriptiebegeleider: Dr. Ir. Michiel Ronsse

Faculteit Toegepaste Wetenschappen
Universiteit Gent

Vakgroep Elektronica en Informatiesystemen
Voorzitter: Prof. Dr. Ir. Jan M. Van Campenhout

Samenvatting
IPv6 werd halverwege de jaren 90 ontwikkeld als opvolger voor het bestaande internetprotocol. Daar beide protocollen nog jaren naast elkaar zullen bestaan moesten er allerlei transitieschema's ontwikkeld worden voor de overschakeling naar het nieuwe protocol. Deze thesis situeert het huidige internetprotocol, bespreekt vervolgens het nieuwe protocol op verschillende besturingssystemen. Daarna wordt gekeken hoe de overgang naar het nieuwe protocol kan gemaakt worden op gebied van connectiviteit, maar ook op gebied van applicaties die met het nieuwe protocol moeten werken. Er wordt een blik geworpen op de veiligheidsrisico's die men loopt bij de invoering van IPv6 en bij de overschakelingsmechanismen.

Trefwoorden: IPv6, tunnels, transitie, encapsulatie

Terminologie

In dit stuk wensen we nog even enkele veelvoorkomende termen te definiëren opdat er geen dubbelzinnigheid zou over ontstaan.

knooppunt Een apparaat dat IPv4- of IPv6-pakketten kan ontvangen en verzenden.

router Een apparaat dat IPv4- of IPv6-pakketten kan ontvangen en/of doorsturen die niet uitdrukkelijk ervoor bestemd zijn.

host Een host is een knooppunt dat geen router is.

link Een communicatiemedium waarop apparaten kunnen communiceren op het niveau van de linklaag.

rfc Request For Comments: Dit is een reeks documenten die standaarden, gedragsregels en aanbevelingen voor het Internet en al zijn protocollen definiëert.

Internet draft Dit is het stadium voor een rfc. Meestal hebben we een werkgroep die een standaard probeert te definiëren. Als die werkgroep denkt dat de draft geen fouten meer bevat kan deze overgaan naar rfc-status. Een voorbeeld van zo'n werkgroep is ngtrans. Dit is de werkgroep die standaardmechanismen definieert voor overgang van IPv4 naar IPv6.

Inhoudsopgave

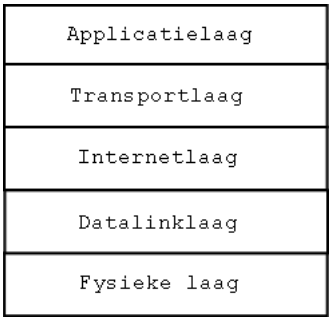
Hoofdstuk 1

IPv6: introductie

1.1 Situering van IPv6

1.1.1 het TCP/IP referentiemodel

Hier bekijken we het huidige internetmodel dat ook in de toekomst nog zal gebruikt worden. Dit model bestaat uit 5 lagen: de applicatielaag, transportlaag, netwerklaag, datalinklaag en fysieke laag. Een uitgebreide bespreking van alle lagen is te vinden in [2].



Figuur 1.1: TCP/IP referentiemodel

Fysieke en datalinklaag

De fysieke laag omvat de hardware; zij bestaat meestal uit kabels (kan natuurlijk ook de lucht zijn bij draadloze transmissie) van allerlei soort die van relatief goedkoop tot erg duur kunnen zijn naargelang hun vermogen om bits sneller te transporteren. Voorbeelden zijn: coax, fiber, utp, draadloos, ... De datalinklaag bestaat uit het type netwerk dat men op het

fysieke transportmedium uitbaat. Voorbeelden hiervan zijn ethernet, isdn, ppp, ...

Internetlaag

De internetlaag is het hoofdthema van deze scriptie. Gedurende een kleine 20 jaar was IPv4 hierop het enige protocol. Men mag niet verkeerdelijk denken dat IPv6 gewoon het huidige IP-protocol is, maar dan met versie 6 in plaats van 4 in de IP-header zoals de naam misschien doet vermoeden. Doordat de lagen niet zo onafhankelijk zijn van elkaar zoals men wel zou willen, moeten voor IPv6 de internetlaag en alle bovenliggende lagen die expliciet gebruik maken van IP-adressen aangepast worden.

Transportlaag

De transportlaag is de vierde laag in het tcp/ip referentiemodel. Ze zorgt voor de communicatie tussen de internet- en de applicatielaag. Er zijn 2 belangrijke protocollen: TCP en UDP. TCP (Transmission Control Protocol) zorgt voor een betrouwbare verbinding bovenop de internetlaag. Dit houdt in dat TCP bijhoudt welke pakketten al verzonden zijn, welke nog verzonden moeten worden en welke verloren gingen en dus opnieuw moeten verzonden worden. TCP regelt ook de snelheid waarmee pakketten verzonden worden. UDP is enkel een tussenstuk en voorziet geen extra functionaliteit. Het voordeel van UDP is dat er weinig overhead is wat belangrijk kan zijn bij applicaties waar een gecontroleerde verbinding opzetten te veel tijd vraagt en/of onnodig is.

Applicatielaag

In het OSI-referentiemodel[1] wordt deze laag nog verder opgesplitst in een sessielaag en een presentatielaag, maar in deze inleidende context is dit niet nodig. Voor de geïnteresseerden verwijzen we hier naar [2]. Hier bevinden zich alle bekende internetprotocollen die servers en eindgebruikersapplicaties implementeren zoals bv. FTP, SSH, SMTP,...

1.2 IPv6: Mythe en realiteit

Het nieuwe internetprotocol IPv6 (soms ook IP Next Generation (IPNG)) genoemd, zal de oplossing zijn voor alle huidige problemen met IPv4. Niets is uiteraard minder waar, bij het ontwerpen van IPv6 had men de bedoeling enkele belangrijke problemen van IPv4 op te

[1] Open Systems Interconnect

lossen. De oplossingen, die uiteindelijk reeds voor een stuk gestandaardiseerd werden, zijn echter compromissen geworden van medewerkers van verschillende onderzoeksgroepen.

IPv6 wordt soms al te snel als de heilzame oplossing beschouwd. Het is nu wel waar dat de tekorten van IPv4 steeds nijpender zullen worden en dat men in de toekomst naar eventuele alternatieven zal moeten grijpen. Laten we daarom eerst even de problemen van IPv4 schetsen, daarna de oplossingen die IPv6 biedt, om dan te eindigen met de extra functionaliteit van IPv6.

De 2 grootste problemen van IPv4 zijn de te korte adreslengte en de zogenaamde explosie van de routingtabellen. Adressen in IPv4 zijn 32 bit lang, dit geeft een mogelijkheid van 2^{32} of ruwweg een 4 miljard adressen. Dit aantal geeft echter een verkeerd beeld van de realiteit. Adressen worden namelijk niet per stuk uitgereikt, maar in blokken toegekend aan providers. Deze providers kennen op hun beurt weer blokken of individuele adressen toe aan hun klanten.

In de beginjaren van het Internet is men wat te kwistig geweest met het uitdelen van dergelijke blokken. Men had 3 klassen van netwerken, klasse A met 2^{24} adressen per blok, klasse B met 2^{16} adressen per blok en klasse C met 2^8 adressen per blok. Voor vele instellingen is een klasse C te klein, maar een klasse B netwerk veel te groot. Van de toegekende blokken werden slechts fracties gebruikt; onze universiteit is hier een mooi voorbeeld van. Hier wordt nog een relatief groot stuk van de adresruimte gebruikt, maar als men de DNS-server er op natrekt, komen er slechts een 10000-tal adressen overeen met een hostname. Dus pakweg 15 procent van het blok is in gebruik. Men schat nu dat met de nog overgebleven, niet toegekende adressen nog voortkan tot 2008.

De oplossingen in IPv4 voor dit probleem zijn een intelligentere toekenning van adressen (door af te stappen van het klasse A,B,C-netwerk systeem) en het gebruiken van NAT. NAT staat voor Network Address Translation. Hierbij gebruikt men gereserveerde private adresblokken voor interne netwerken. Zo kan men een groot aantal computers via 1 IP-adres op het Internet laten. NAT introduceert wel een hoop problemen omdat we geen *end-to-end*-verbinding meer hebben. Computers in een privaat netwerk kunnen dus niet rechtstreeks bereikt worden van op het Internet. Sommige protocollen hebben hier problemen mee. (bv. FTP, IPSec)

Een ander probleem is de explosie van de routingtabellen. Hiermee bedoelen we dat de belangrijkste routers van het Internet zoveel routes hebben dat ze deze nauwelijks nog kunnen verwerken. Ook dit is een gevolg van het onzorgvuldig toekennen van adresblokken. Bijvoorbeeld: 157.193.0.0/16 behoort toe aan RUGnet en 157.192.0.0/16 aan Sanyo in Japan. Dergelijke onzinnigheden zorgen ervoor dat men heel veel routes heeft in de routingtabellen.

IPv4 heeft ook het probleem van de groeiende routingtabellen een halt toegeroepen door invoering van het CIDR principe. Bijvoorbeeld 80.0.0.0/8 wordt naar RIPE gerouteerd en vandaar 80.200.0.0/16 naar Skynet België. Routers in de VS hoeven zo enkel 80.0.0.0/8 in

hun routingtabel te hebben.

Welke oplossingen biedt IPv6 nu voor deze belangrijke tekorten van IPv4? De grootte van de IPv6-adresruimte is 2^{128} bits. Dit is een immens aantal en lost dus zeker het tekort aan adressen op. De toekenning van IPv6-blokken zal ook via het CIDR-principe gebeuren en de IPv6-routingtabellen zullen dus in de hand gehouden worden.

Nu zijn er ook allerlei geruchten dat IPv6 oplossingen voor multicast, mobiel IP, QOS (Quality of Service), veiligheid biedt. Dit is echter niet zo. Deze diensten worden onderliggend, maar separaat, geïmplementeerd. Zij worden op zich dus niet eenvoudiger door een overgang naar IPv6.

1.3 Doelstelling

De doelstelling van deze thesis is na te gaan hoe we bestaande IPv4-netwerken kunnen omschakelen naar IPv6. Tevens dient er nagegaan te worden wat er gebeurt indien we een nieuw IPv6-netwerk aan het Internet hangen. Hoe kunnen we dit IPv6-netwerk met het bestaande Internet laten communiceren zonder dat we hiervoor nog IPv4 nodig hebben.

Om deze doelstellingen te bereiken dient eerst een studie gemaakt te worden van IPv6 zonder dat we IPv4 in het achterhoofd houden. De configuratiemogelijkheden worden bekeken op 3 belangrijke platformen. FreeBSD en Linux zijn vooral belangrijk als routerplatformen en beschikken ook reeds over heel wat transitie-implementaties. Linux en Windows XP zijn belangrijk als werkstations (belangrijk is daarom nog niet gelijk aan interessant).

Eerst zal een analyse gemaakt worden van IPv6 als protocol, waarbij de headers van IP- en ICMP-pakketten zullen bekeken worden. Voorts worden ook de adressen beschouwd: de notatie, de verschillende soorten en de manier waarop deze verdeeld worden onder de providers en eindgebruikers.

In een volgend hoofdstuk bespreken we hoe we IPv6 kunnen configureren: hoe we adressen toekennen aan interfaces en hoe deze automatisch kunnen toegekend worden. Voorts wordt er besproken over hoe we routers kunnen instellen.

In het vierde hoofdstuk wordt er even gekeken welke applicaties reeds IPv6 spreken. We onderscheiden serverprogramma's en eindgebruikersprogramma's. Een belangrijk programma hierbij is de nameserver die drastische uitbreidingen gekregen heeft.

In het vijfde hoofdstuk komt dan uiteindelijk de hoofdbrok van deze thesis: de transitiemogelijkheden. Er wordt bekeken welke mechanismen er zijn om van de clientzijde op het IPv6-netwerk te raken. Anderzijds behandelen we ook hoe men IPv6 kan aanbieden aan mensen die niet direct met het IPv6-netwerk verbonden zijn. Eens we verbonden zijn willen we via

IPv6 ook alles kunnen wat via IPv4 mogelijk was. Men wil ook via IPv6 connecteren met computers die nog niet via IPv6 bereikbaar zijn. Dit kunnen we realiseren via allerlei vertalingsmechanismen. Het verschil tussen al die mechanismen is de plaats waar de vertaling plaatsvindt.

Het zesde hoofdstuk beschrijft hoe we deze overgang veilig kunnen laten verlopen. Veel netwerken zijn beveiligd voor IPv4, maar hoe kunnen we die dan ook afdoende voor IPv6 beveiligen. Langs de vertalers heen kan men ook allerlei constructies opzetten om ongeauthoriseerde toegang te verkrijgen tot bepaalde diensten. Het is dus belangrijk dat we ons hiertegen afdoende beveiligen om niet voor ongewenste verrassingen te komen staan.

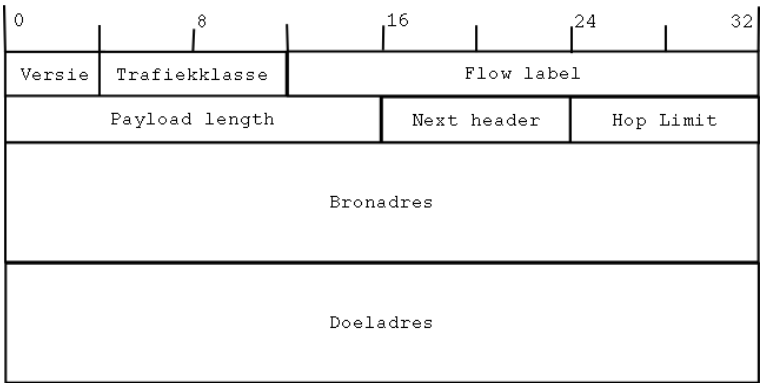
In de appendices bespreken we de kernverschillen tussen Linux en FreeBSD. Dit is interessant omdat FreeBSD een stuk verder staat dan Linux op het gebied van IPv6. Op dit moment wordt FreeBSD veel te weinig gekozen als serverplatform; mensen neigen eerder de hype van Linux te volgen of nog erger, iets uit Redmond... Voorts bespreken we de implementatie van een IPv6-netwerk op rugnet met enkele transitiemethodes.

Hoofdstuk 2

IPv6 protocollen en specificaties

We wensen in dit hoofdstuk de belangrijkste specificaties van de IPv6-laag te bespreken. Dit houdt in dat de IPv6 en de ICMPv6 (controlepakketten voor IPv6) zullen besproken worden. Daarnaast worden ook de types van IPv6-addressen besproken en hoe de toekenning ervan gebeurt.

2.1 IPv6-header



Figuur 2.1: IPv6-header

Bij het ontwerp van de IPv6-header heeft men de keuze gemaakt om de basisheader zo eenvoudig mogelijk te houden. Allerlei extensies kunnen dan eventueel in een *extension header*. De filosofie is echter nog steeds dat men op veilig speelt, of zeg maar op erg veilig, want de keuze van het aantal bits voor de opties is meestal erg toekomstgericht.

Het eerste veld, het *version field*, bevat de versie van het protocol en is dus gelijk aan 6.

Hiervoor werden vier bits gereserveerd. Men mag hier niet verwarren met IPv4 waar een vier staat in het *version field*, want bij deze is de header volledig anders en het protocol in de onderliggende laag verschilt dus. IPv6 is dus geen nieuwe versie van IPv4, maar een nieuw protocol.

Het tweede veld (8 bits, traffic class) bevat het type verkeer dat we wensen te verzenden. Dit kan interessant zijn om bepaalde types verkeer voorrang te geven op andere. Het is gewenst dat waretijdsapplicaties voorrang krijgen op andere. In rfc1883[5] was dit veld nog vier bits en noemde dit het prioriteitsveld. In rfc2460 staat de laatste versie van de IPv6-specificatie. Om de header even groot te houden werd het *flow label* met vier bits ingekort.

Het *flow label* wordt op dit moment niet gebruikt en op 0 gezet. Het kan eventueel gebruikt worden om een volgorde in een stroom van pakketten aan te geven zodat routers weten welk pakket ze eerst dienen te forwarden, bijvoorbeeld voor waretijdsapplicaties.

Voorts zijn er zestien bits gereserveerd (*payload length*) om de grootte van het pakket te specificeren. Dit kan als een beperking gezien worden en daarom kan men ook lengte 0 nemen. Hier geeft men aan dat het pakket nog groter is: dit zijn jumbogrammen. Een uitgebreide bespreking over jumbogrammen is te vinden in rfc2675[17].

De *next header* bevat het type header dat onmiddelijk volgt na deze van IPv6. Meest gebruikt zijn ofwel een *extension header* ofwel TCP ofwel UDP, een overzicht van alle types header is te vinden op `http://www.iana.org/assignments/protocol-numbers`.

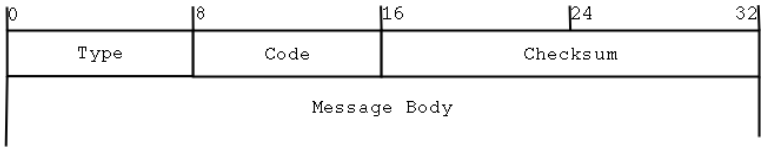
De *hop limit* is een gelijkaardig mechanisme zoals bij IPv4. Nu zijn er echter acht bits voorzien in plaats van vijf. IPv6-pakketten kunnen dus maximaal 254 routers passeren. Bij elke router wordt dit veld met één gedecrementeerd.

Uiteindelijk is er nog een bron- en doeladres van elk 128 bits. Wanneer men de *routing header* gebruikt is het doeladres niet gelijk aan die van de bestemmeling, maar van de router waar het pakket heen moet om vandaar verder getransporteerd te worden.

De totale legnte van de IPv6-header is dus minimaal gelijk aan 40 bytes.

2.2 ICMPv6

ICMPv6-pakketten zijn de controlepakketten bij het IPv6-protocol. We onderscheiden 2 soorten. Enerzijds zijn er informatieve berichten en anderzijds zijn er foutberichten.



Figuur 2.2: ICMPv6-header

2.2.1 ICMPv6-header

Het eerste veld is het type ICMPv6-pakket. Het veld is 8 bits lang en de hoogste orde bit geeft aan of we met een informatief bericht (hoogste bit=1) of een foutbericht (0) te maken hebben.

Voorts is er nog een code veld die extra informatie meegeeft en een *checksum* die transmissiefouten detecteert in het ICMPv6-pakket.

2.2.2 Foutberichten

Destination Unreachable

Wanneer de bestemming van een pakket onbereikbaar is, wordt een pakket teruggezonden naar de afzender van het oorspronkelijke pakket. De redenen waarom dit gebeurt zijn uiteenlopend.

- Er is geen route naar de bestemming in de routingtabel.
- Er staat een firewall die de pakketten blokkeert.
- Bestemmingspoort onbereikbaar.
- Andere ...

Packet Too Big

Dit soort pakketten wordt gebruikt om de MTU[1] te ontdekken. De zender probeert zo groot mogelijke pakketten te verzenden, komt men op een link die dit niet aankan dan wordt een pakket teruggezonden. De zender past zich dan aan aan de MTU van de link. Fragmentatie gebeurt dus niet meer wanneer een pakket onderweg is. Fragmentatie van het pakket kan wel gebeuren bij de zender.

[1]MTU=maximal transfer unit; dit is het grootste pakket dat een link kan verzenden

Time Exceeded

Als het maximale aantal routers overschreden is wordt een pakket van dit type teruggezonden. Dit wordt gedetecteerd doordat het *hop limit* veld in de IPv6-header op nul komt.

Parameter Problem

Als er een probleem optreedt met een optie of een extra header die niet herkend wordt of die een verkeerde waarde heeft, wordt dit ICMPv6-bericht teruggezonden. Dit kan gebeuren omdat we in een heterogeen netwerk zitten met allemaal verschillende implementaties van de IPv6-stapel.

2.2.3 Informatieve berichten

Echo Request

Dit is het type bericht dat typisch door het commando *ping*[2] wordt verzonden. Hierbij vraagt men aan de bestemming om een bericht terug te zenden. Hiermee kunnen we dus achterhalen of onze bestemmeling bereikbaar is.

Echo Reply

Dit is het type bericht dat teruggezonden wordt als antwoord op een echo request. Als alles goed gaat, weten we dus of knooppunten met elkaar kunnen communiceren. Als de pakketten niet aankomen, wil dat daarom nog niet zeggen dat de onbereikbaar zijn voor elkaar. Firewalls[3] durven nog al eens ICMPv6-pakketten te filteren.

Overige berichten

Er zijn nog de ICMPv6 berichten die te maken hebben met *neighbour discovery*, hiervoor verwijzen we naar 3.2.

[2]ping is een programma dat nagaat of een knooppunt bereikbaar is. ping is meestal het commando voor IPv4, voor IPv6 noemt het dikwijls ping6.

[3]Dit zijn toestellen die op een link of router bepalen welke pakketten het netwerk al dan niet binnen mogen of mogen verlaten.

2.3 IPv6-adressen

In deze sectie bekijken we hoe de 128 bits van een IPv6-adres ingevuld worden, wie deze adressen uitdeelt, hoe men hierbij de doelstelling van kleinere routingtabellen probeert te bereiken. Voor alle voordelen van IPv6 zijn er verschillende blokken gereserveerd en met zo'n enorme adresruimte is er wel voldoende ruimte om voor de meest exotische dingen ook adressen te reserveren.

Een mogelijke indeling van de adressen is ze te selecteren op hoe de bestemmeling wordt gevonden. Bij unicast is er één bestemmeling en is die uniek bepaald. Bij anycast hebben we ook één bestemmeling, maar die is afhankelijk van waar men zich op het netwerk bevindt. Men neemt de dichtsbijzijnde bestemmeling. Een pakket dat men zendt naar een multicastadres wordt naar meerdere bestemmelingen gezonden.

Eerst zullen we de notatie van IPv6-adressen uitleggen en we zullen eindigen met de toekenningen van de IPv6-blokken.

2.3.1 Notatie

Adres

IPv6-adressen worden hexadecimaal genoteerd in 8 blokken van 4 cijfers gescheiden door een dubbele punt (:). bv.

```
2001:06a8:BBBB:CCCC:D216:A873:123F:FCD4
```

In de praktijk is het gebruik van dergelijke adressen een ramp en daarom zullen de meeste ook niet zo ingewikkeld zijn. Het gebruik van automatische toekenning van adressen dringt zich toch op en dit zal ook gebeuren zoals we later zullen zien.

Realistischer voorbeelden van adressen zijn bv.

```
2001:06a8:00ff:0000:0000:0000:0000:0001
```

Nu gelden twee belangrijke regels in de notatie. Leidende nullen hoeven niet geschreven te worden. Dus 06a8 wordt 6a8, 00ff wordt ff, enz. Een tweede belangrijke regel is dat een groep nullen over : heen vervangen mag worden door :: Hierbij wordt de vereenvoudiging van bovenstaand voorbeeld:

```
2001:6a8:ff::1
```

Deze laatste regel is slechts 1 maal per adres toepasbaar, anders is het niet meer uniek bepaald. In het andere geval dient men de nullen uit te schrijven. Beschouw het fictieve adres: 2001:0000:0000:1111:2222:0000:0000:0001. De volgende schrijfwijzen zijn allen correct:

```
2001::1111:2222:0:0:1
2001:0:0:1111:2222::1
2001:0:0:1111:2222:0:0:1
```

Prefix

Een IPv6-prefix geeft het netwerkdeel aan van een IPv6-adres. Alle hosts op hetzelfde netwerk dienen dus hetzelfde prefix te hebben. De notatie is als volgt

```
ipv6-adres/prefixlengte
```

Een voorbeeld met een 64 bits-prefix:

```
2001:6a8:1904:1::/64
```

Meestal noteert men nooit het netwerkadres, maar enkel het hostadres met de bijbehorende prefixlengte, zo kan met direct het netwerkadres afleiden.

```
2001:6a8:1904:1::1/64
```

2.3.2 Unicast

Deze klasse van adressen is de meest voor de hand liggende. Een unicast adres kan maar aan precies één interface gegeven worden. Voorts kunnen we deze adressen verder indelen naar gelang hun functie: lokaal, loopback, internetadres, enz.

Aggregatable global unicast adressen

Dit type adressen begint steeds met 001. Ze worden toegekend via het CIDR principe. Zie toekenning van adressen voor meer info hierover. Verder zal ik deze vrij vertaald de globale unicast adressen noemen.

Geografisch gebaseerde adressen

Een ander achtste van de IPv6 adresruimte werd hiervoor gereserveerd. De eerste drie bits zijn steeds 100. Het principe is mooi, maar kent te veel tegenstand om gebruikt te worden. Het idee is dat men aan de hand van het IP-adres kan zien van waar men komt. Dit is ook een poging om de routingtabellen in de hand te houden. Dit was althans de theorie, in de laatste RFC over IPv6-adressering (rfc2373[8]) heeft men dit type adressen geanuleerd.

Lokale link-adressen

Een IPv6-netwerk werkt op zich onmiddellijk. Dit is de verdienste van de lokale adressen die automatisch gegenereerd worden. De eerste tien bits van een lokaal linkadres zijn steeds 1111 1110 10. De laatste 64 bits worden gevormd aan de hand van het interface-id. Hoe dit precies gebeurt zien we verder. De overige 54 bits worden 0 genomen.

Een voorbeeld van zo'n adres wanneer de netwerkinterface-id gelijk is aan `00:bd:45:50:00:01` `fe80::2bd:45ff:fe50:1`

Lokale site-adressen

Met 1 netwerk komt men dikwijls niet toe; anderzijds heeft men niet steeds internetadressen nodig. Daarom zijn er ook in IPv6 intranet-blokken gereserveerd. De eerste eerste bits van deze adressen zijn 1111 1110 11. Deze adressen zijn te vergelijken met de private klasses in IPv4 en dienen dus ook als dusdanig gebruikt te worden. Hierbij kan men wel de opmerking maken dat men NAT ten alle prijze wenst te vermijden in IPv6. Dus dit blok is echt bedoeld voor IP-netwerken die niet op het Internet zitten, of die toegang hebben via bv een proxyserver. Dat is althans de theorie. Een voordeel van deze adressen is dat men niet de rompslomp heeft om een blok te moeten aanvragen. We vrezen hier wel dat men zijn doelstelling van het vermijden van NAT hier verliest.

Het ongespecificeerde adres

Het adres 0000:0000:0000:0000:0000:0000:0000:0000 of kortweg :: is het ongespecificeerde adres. Het kan gebruikt worden wanneer een knooppunt zichzelf configureert om zijn adres te bekomen. In alle andere gevallen zou dit adres niet mogen gebruikt worden.

Het loopback-adres

Het adres ::1 is het loopback-adres. Dit is wanneer een knooppunt een pakket naar zichzelf zendt, sommige netwerkprogramma's kunnen nu eenmaal niet zonder netwerk, ook al is dit louter fictief. Dit adres kan natuurlijk ook gebruikt te worden om te zien of uw host IPv6 verstaat. (bv. ping6 ::1)

Ingebedde IPv4-adressen

We onderscheiden meerdere soorten: we hebben de IPv4-compatibele IPv6 adressen, de IPv4-gemapte IPv6 adressen, de IPv4-vertaalde adressen en ISATAP-adressen. Vaak vormen hier de IPv4-adressen de laatste 32 bit van het IPv6-adres en om geen berekeningen naar hexadecimale notatie te moeten maken noteert men meestal a:b:c:d:e:f:x.y.z.w. Met x.y.z.w het IPv4-adres en daarvoor de notatie van de eerste 96 bits van het IPv6-adres.

IPv4-compatibele adressen bestaan voor de eerste 96 bits uit nullen gevolgd door het IPv4-adres. Deze wordt gebruikt door hosts die zowel een IPv4 als een IPv6 stapel hebben.

IPv4-gemapte IPv6-adressen hebben de eerste 80 bits nullen dan 16 bits enen en dan het IPv4 adres. Dit is een adres voor knooppunten die enkel IPv4 ondersteunen.

IPv4-vertaalde adressen worden gebruikt bij SIIT en hebben de vorm ::ffff:0:0:0:a.b.c.d met a.b.c.d het ingebedde IPv4-adres.

Er is verder ook nog een draft van Microsoft waarin ze ISATAP-adressen[4] gebruiken. Deze bevatten ook ingebedde IPv4-adressen.

Andere

Voor de volledigheid vermelden we nog dat NSAP-adressen en IPX-adressen ook een plaats gekregen hebben in IPv6. De toekenning hiervan is nauwelijks gestandaardiseerd. Voor meer informatie verwijzen we naar rfc1888 en rfc2373.

2.3.3 Anycast

Anycast adressen zijn adressen die worden toegekend aan meer dan één interface. Wanneer een pakket verzonden wordt naar zo'n adres dan wordt het gerouteerd naar de dichtsbijzijnde interface. Hoe de routing dan precies gebeurd wordt uitgelegd in rfc2373[8]. De gebruikte adressen komen uit het blok gereserveerd voor unicastadressen.

2.3.4 Multicast

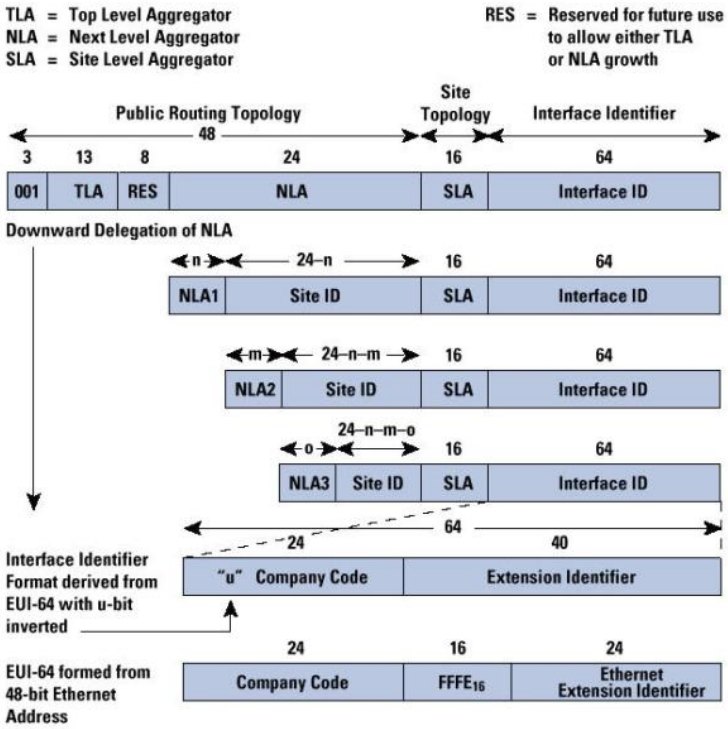
Een IPv6-multicast adres komt overeen met een groep knooppunten. Ze kunnen een beetje vergeleken worden met de IPv4-broadcast adressen, maar zijn veel uitgebreider. Alle adressen die beginnen met 1111 1111 zijn multicast adressen. Hier vermeld ik enkel de twee belang-

[4]ISATAP is nog in ontwikkeling door de ietf-werkgroep ngtrans

rijkste adressen zijnde ff02::1 (alle knooppunten) en ff02::2 (alle routers) die op een lokale link liggen.

2.3.5 Toekenning van adressen

In dit stuk zullen we het vooral hebben over de globale unicastadressen omdat deze de belangrijkste groep vormen bij de toekenning van de adressen. De eerste drie bits hiervan liggen dus vast en zijn 001. Er vallen dus nog 125 bits op een zinvolle manier te verdelen. De verdeling wordt weergegeven in volgende figuur



Figuur 2.3: Toekenning van unicastadressen

Eerst hebben we een TLA van dertien bit. Dit is de top van de hiërarchie, slechts enkelen zullen een TLA toegewezen krijgen. De volgende in de hiërarchie zijn de NLA's. Hiervoor zijn 24 bits voorzien. Een adresverdeler die een NLA heeft kan opnieuw NLA's toekennen of kan SLA's toekennen. Een SLA hoort toe aan een individuele organisatie. Deze kan dan zelf de laatste zestien bits toekennen aan de verschillende netwerken binnen die organisatie. De laatste 64 bits worden dan gebruikt op hetzelfde netwerk om de hosts van elkaar te onderscheiden. Dus per netwerk kunnen 2^{64} knooppunten staan, wat een enorm aantal is.

Men zou kunnen kiezen om die hosts te nummeren, een handiger methode is het hostnummer af te leiden van het hardwareadres zodat dit automatisch kan gebeuren. Hoe dit gebeurt zien we in de volgende paragraaf.

Laatste 64 bit aan de hand van het hardware adres

Aan de hand van het 64-bits of het 48-bits hardware adres worden de laatste 64 bits gevormd. Bij de 64-bits hardware adressen wordt de universele bit op 1 gezet (7de bit). Bij de 48-bits hardware adressen (bv. Ethernet Mac-adres) wordt dit ook gedaan, en worden nog zestien bit enen toegevoegd tussen de 24 eerste en laatste bits. Voor andere types interfaces verwijzen we naar rfc2373[8]. Deze laatste 64 bit worden ook wel eens de EUI-64 identifier genoemd.

Aggregatable globale unicast adressen

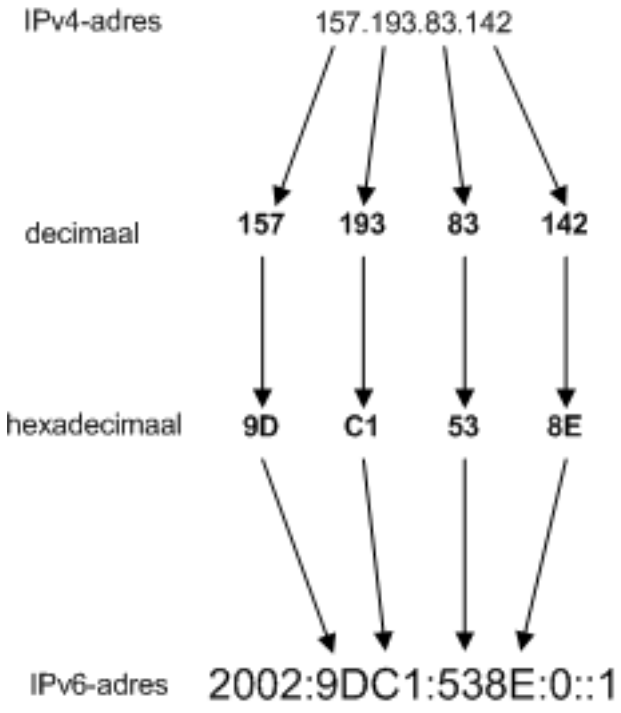
Hoe verkrijgt men nu effectief een adres op het huidige IPv6-netwerk? Op dit moment is het IPv6-netwerk een combinatie van drie netwerken die grotendeels samenhangen, maar er zijn plaatsen waarbij het ene netwerk niet kan routeren naar het andere. Men kan een IPv6-adres bekomen aan de hand van je IPv4-adres; deze manier noemt 6to4. Men kan ook deel uit maken van het eerste experimentele IPv6-netwerk dat voor testdoeleinden werd opgezet, de 6bone. Een derde netwerk is het netwerk dat uiteindelijk het huidige IPv4-netwerk zal moeten overnemen.

6to4 Het principe bij deze IPv6-adressen is eenvoudig; zodra men beschikt over een IPv4-adres beschikt men automatisch over een /48[5]. Het IPv6-adres kan berekend worden aan de hand van het IPv4-adres als volgt:

De eerste zestien bits staan vast voor 6to4 en worden hexadecimaal gegeven door 2002. De volgende 32 bits zijn de 32 bits van het IPv4-adres. De laatste 80 bits zijn vrij te kiezen door de gebruiker. Waarbij er zestien bit netwerken kunnen gevormd worden en de laatste 64 worden gevormd aan de hand van het hardware adres van de hosts op het netwerk.

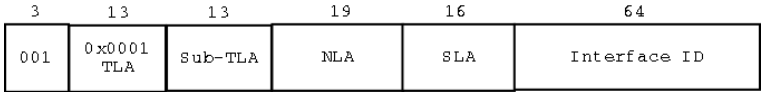
6bone De 6bone is het experimentele IPv6-netwerk dat opgericht werd voor testdoeleinden. Op dit moment is dit het adrestype dat het meest gebruikt wordt. Het prefix dat werd voorzien voor de 6-bone is 3ffe::/16. In rfc2921 [24] kun je nalezen hoe de interne toekenning gebeurt van de adresblokken.

[5]Dit is een notatie die vaak gebruikt wordt; hiermee wordt bedoeld dat de eerste 48 bit vast liggen en dat men voor de overige bits zelf mag kiezen



Figuur 2.4: 6to4

Productieadressen: 2001::/16 Uiteindelijk is het de bedoeling dat we in de toekomst gaan naar een volwaardig IPv6-netwerk dat bestaat uit zuivere IPv6-verbindingen en niet meer uit tunnels tussen IPv6-eilandjes. Hierbij zou men kunnen verwachten dat IANA enkele TLA's zou toekennen aan de grote adresverdelers APNIC, RIPE, IANA en ARIN. In de RFC's worden deze ook wel eens de *registries* genoemd. Niets is minder waar. Men heeft slechts 1 TLA gealloceerd en deze dan verder weer ingedeeld. Dit heeft men gedaan om de overgang geleidelijk aan te laten gaan. Men merkt hier duidelijk precies het omgekeerde effect als bij IPv4. Toen werden in het begin adresblokken veel te kwistig uitgedeeld (organisaties die een volledig klasse-A netwerk konden krijgen). In dit ontwerp werkt men met sub-TLA's (sTLA). Dit is een onderverdeling van de eerste TLA (0x0001). Het aantal organisaties dat in aanmerking komt voor een sTLA en dus ook een TLA is op 1 hand te tellen, terwijl we anderzijds over 8192 TLA's kunnen beschikken.



Figuur 2.5: Toekenning productieadressen

De filosofie is dat als een organisatie kan aantonen dat hij niet genoeg heeft met een sTLA, hij dan een TLA toegekend zal krijgen van IANA. Laten we nu even de sTLA beschouwen. In de klasse 2001::/16 is dus het gereserveerde veld gebruikt voor de sTLA's. De *registries* krijgen allen een blok sTLA's toegekend. Zij kunnen deze dan verder toekennen aan organisaties. Wie welke blokken aan wie mag toekennen zou ons te ver leiden. Hiervoor verwijzen we naar de site van IANA (`http://www.iana.org/`) of naar rfc2928[25] en rfc2450[10].

Binair	IPv6 Prefix Range	Toekenning
0000 000X XXXX X	2001:0000::/29 - 2001:01F8::/29	IANA
0000 001X XXXX X	2001:0200::/29 - 2001:03F8::/29	APNIC
0000 010X XXXX X	2001:0400::/29 - 2001:05F8::/29	ARIN
0000 011X XXXX X	2001:0600::/29 - 2001:07F8::/29	RIPE NCC
0000 100X XXXX X	2001:0800::/29 - 2001:09F8::/29	(toekomst)
0000 101X XXXX X	2001:0A00::/29 - 2001:0BF8::/29	(toekomst)
0000 110X XXXX X	2001:0C00::/29 - 2001:0DF8::/29	(toekomst)
0000 111X XXXX X	2001:0E00::/29 - 2001:0FF8::/29	(toekomst)
0001 000X XXXX X	2001:1000::/29 - 2001:11F8::/29	(toekomst)
...	...	...
1111 111X XXXX X	2001:FE00::/29 - 2001:FFF8::/29	(toekomst)

Hoofdstuk 3

Configuratie voor IPv6: IP-laag

In dit hoofdstuk zullen we kijken hoe we een systeem met IPv6 kunnen uitrusten. We zullen vaststellen dat de meeste besturingssystemen al een aardig woordje IPv6 spreken en verstaan. Op de IP-laag is dan ook al wel een groot deel van het werk verricht. Op sommige besturingssystemen moet men wel nog komen tot 'productie'-code; waarbij men bedoelt dat er reeds een IPv6-laag is geïmplementeerd, maar dat de code nog niet klaar is voor professioneel en alledaags gebruik.

Daar we vanaf dit hoofdstuk met platformafhankelijke zaken zitten zullen we de volgende aanpak volgen. Eerst wordt elk onderwerp algemeen besproken; daarna wordt het toegepast op enkele besturingssystemen. De besturingssystemen die praktisch zullen besproken worden zijn: FreeBSD 4.5-STABLE, Windows XP en Linux Debian Sid(kernel 2.4.13). Soms wordt er ook gebruik gemaakt van code die nog in volle ontwikkeling is. Dat is een KAME-kernel voor FreeBSD en een USAGI-kernel voor Linux.

Om al die besturingssystemen te testen heb je een pak computers nodig. We hadden een pak pc's ter beschikking verspreid over elis, zeus, thuis en op kot. Soms hebben we ook al eens gebruik gemaakt van vmware. Hierbij was het platform FreeBSD. Deze kan vmware draaien door middel van Linux-emulatie, want vmware brengt enkel linux- en windowsbinaries uit.

Men spreekt hier van een gast en een gastheer. De gast is het besturingssysteem dat draait op de virtuele computer, de gastheer is het besturingssysteem dat vmware draait. Als gast hebben we verschillende besturingssytemen geprobeerd, o.m. een Linux met een USAGI-kernel.

Het besturingssysteem waar we het meest mee gewerkt hebben, is FreeBSD. Dit omwille van zijn stabiliteit, het gemak om routes aan te passen en routers te configureren en omwille van het ports-systeem, waarin men gemakkelijk de recentste applicaties kan compileren met IPv6-support indien die daar reeds voor voorzien zijn of indien er een patch bestaat. Meer

informatie over FreeBSD voor linuxgebruikers is te vinden in Appendix A.

3.1 Handmatige configuratie

Onder het besturingssysteem voorzien van een IPv6-stack verstaan we dat de IPv6-specificaties voor de internet- en transportlaag geïmplementeerd worden. Voor de netwerklaag geldt dan dat het IPv6 en het ICMPv6 protocol moeten geïmplementeerd worden. Deze protocollen worden beschreven in rfc2460[11] en rfc2463[14].

Interfaces moeten dus één of meerdere IPv6-adressen kunnen krijgen, IPv6-sockets moeten kunnen aangemaakt worden. De routing van IPv6-pakketten moet mogelijk zijn. ICMPv6 moet werken.

Alle besproken besturingssystemen hebben reeds een IPv6-stapel. Het enige waar ze in verschillen, is de mate van productiecode. Daar de implementatie van de IP-laag kernelcode is, is het erg belangrijk dat deze code zo foutloos mogelijk is.

Omdat het effectief toekennen van IP-adressen verschillend is voor routers als voor hosts bespreken we die in de volgende secties apart. Routers zullen dikwijls nog handmatig geconfigureerd worden, terwijl de configuratie in de hosts bij voorkeur allemaal automatisch verloopt. In de volgende paragrafen beschouwen we de kennis van IPv6 in de besproken van besturingssystemen.

3.1.1 FreeBSD

Op dit moment zit *BSD (FreeBSD, NetBSD, OpenBSD) en in het bijzonder FreeBSD met de stabielste code. De standaard installatie van FreeBSD komt met IPv6 in de kernel. De drijfveer achter de ontwikkeling van deze code is het KAME-project, dat te vinden is op `http://www.kame.net/`.

Eerst dienen we onze kernel te voorzien van IPv6-ondersteuning. In de stable-branch van FreeBSD bevatten de kernels reeds standaard IPv6-ondersteuning. Dus meestal hoeven we die zelfs niet te hercompileren. De optie INET6 volstaat.

```
options INET6 #IPv6 communications protocols
```

Een adres kunnen we dan een aan een interface toekennen als volgt:

```
root# ifconfig ed0 inet6 2001:6a8:1904:1::1 prefixlen 64
```

Een controle geeft dan:

```
root# ifconfig ed0
ed0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet6 fe80::2e0:7dff:fe84:37d%ed0 prefixlen 64 scopeid 0x1
        inet6 2001:6a8:1904:1::1 prefixlen 64
        ether 00:e0:7d:84:03:7d
```

3.1.2 Linux

Linux spreekt ook een goed woordje IPv6. Hier wordt de ontwikkeling gedaan door het USAGI-project, te vinden op `http://www.linux-ipv6.org/`. Om IPv6-support in Linux aan te zetten moet men deze in de kernel compileren of de module laden. Men neemt best de recentste kernel hiervoor want sommige basiseigenschappen werken nog niet in oudere kernels. Een goed startpunt voor Linux en IPv6 is de Linux-IPv6-HOWTO van Peter Bieringer[35].

Afhankelijk van de gebruikte distributie zit IPv6 al dan niet al in de standaardkernel meegecompileerd. De gebruikte distributie (debian unstable) bevat nog de 2.2.20 kernel zonder IPv6 ondersteuning. We gebruiken hier een Linux-2.4.13 met volgende kernelopties:

```
Code maturity level options
[*] Prompt for development and/or incomplete code/drivers
Networking options
<*> Packet socket
[*] Kernel/User netlink socket
[*]   Routing messages
<*> Unix domain sockets
[*] TCP/IP networking
<*>   The IPv6 protocol (EXPERIMENTAL)
```

Er is ook nog wel een verschil tussen wat de *vanilla kernel*[1] ondersteunt en wat het USAGI-project reeds geïmplementeerd heeft. Een stand van zaken is te vinden op `http://www.bieringer.de/linux/IPv6/status/IPv6+Linux-status-kernel.html`. Het is ook mogelijk om IPv6 als module te compileren. Een interface kan dan als volgt geconfigureerd worden:

```
root:~# ifconfig eth0 inet6 add 2001:6a8:1904:7::1/64
```

Een controle geeft:

[1] Dit is de officiële kernel zoals die gelanceerd wordt op ftp.kernel.org

```
root:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:10:5A:DA:36:7B
          inet addr:192.168.1.2  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: 2001:6a8:1904:7::1/64 Scope:Global
          inet6 addr: fe80::210:5aff:feda:367b/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:216 errors:0 dropped:0 overruns:0 frame:0
          TX packets:174 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:20207 (19.7 KiB)  TX bytes:19209 (18.7 KiB)
          Interrupt:10 Base address:0x300
```

3.1.3 Windows

Windows XP komt ook met een IPv6-stack, al is die wel erg goed verborgen. Het is niet via de vertrouwelijke aanvink- en ok-venstertjes, maar via de shell[2] dat IPv6 dient geactiveerd te worden. Eerst dien je de stack te installeren:

```
ipv6 install
```

Windows XP heeft 4 IPv6-interfaces in een pc met 1 netwerkkaart. Deze kunnen opgevraagd worden als volgt:

```
bash$ipv6 if
Interface 4: Ethernet
  gebruikt Neighbor Discovery
  gebruikt Router Discovery
  adres van Link-laag: 00-e0-7d-84-03-7d
    preferred link-local fe80::2e0:7dff:fe84:37d, levensduur infinite
    multicast interface-local ff01::1, 1 refs, kan niets over worden gemeld
    multicast link-local ff02::1, 1 refs, kan niets over worden gemeld
    multicast link-local ff02::1:ff84:37d, 1 refs, laatste melding
  koppelt MTU 1500 (ware MTU-koppeling 1500)
  huidige hop-limiet 128
  bereiktijd 21000 ms (basistijd 30000ms)
  herverzendingsinterval 1000ms
```

[2]Standaard komt bij Windows de DOS-prompt, het is echter sterk aan te raden iets deftiger te installeren, bv. cygwin. (`http://www.cygwin.com`) Dit bevat een unix-basissysteem met een bash onder Windows.

```
  DAD-verzendingen 1
Interface 3: 6to4-tunneling-pseudo-interface
  gebruikt geen Neighbor Discovery
  gebruikt geen Router Discovery
    preferred global 2002:d464:b633::d464:b633, levensduur infinite
  koppelt MTU 1280 (ware MTU-koppeling 65515)
  huidige hop-limiet 128
  bereiktijd 36000 ms (basistijd 30000ms)
  herverzendingsinterval 1000ms
  DAD-verzendingen 0
Interface 2: Automatische tunneling-pseudo-interface
  gebruikt geen Neighbor Discovery
  gebruikt geen Router Discovery
  adres van Link-laag van router: 0.0.0.0
  In EUI-64 ingesloten IPv4-adres: 0.0.0.0
    preferred link-local fe80::5efe:212.100.182.51, levensduur infinite
    preferred global ::212.100.182.51, levensduur infinite
  koppelt MTU 1280 (ware MTU-koppeling 65515)
  huidige hop-limiet 128
  bereiktijd 25500 ms (basistijd 30000ms)
  herverzendingsinterval 1000ms
  DAD-verzendingen 0
Interface 1: Loopback-pseudo-interface
  gebruikt geen Neighbor Discovery
  gebruikt geen Router Discovery
  adres van Link-laag:
    preferred link-local ::1, levensduur infinite
    preferred link-local fe80::1, levensduur infinite
  koppelt MTU 1500 (ware MTU-koppeling 4294967295)
  huidige hop-limiet 128
  bereiktijd 42000 ms (basistijd 30000ms)
  herverzendingsinterval 1000ms
  DAD-verzendingen 0
```

We hebben het loopback-device, 2 tunnelinterfaces (6to4 en automatische tunnels) en dan hebben we nog de echte interface van onze netwerkkaart. De tunnelinterfaces worden later besproken. De tweede interface geven we dan als volgt een IP-adres:

```
ipv6 adu 2/3ffe:8100:100:a::1459
```

Uiteindelijk wordt dat dan:

```
Interface 2: Automatische tunneling-pseudo-interface
  gebruikt geen Neighbor Discovery
  gebruikt geen Router Discovery
  adres van Link-laag van router: 0.0.0.0
  In EUI-64 ingesloten IPv4-adres: 0.0.0.0
    preferred global 3ffe:8100:100:a::1459, levensduur infinite (handmatig)
    preferred link-local fe80::5efe:212.100.182.51, levensduur infinite
    preferred global ::212.100.182.51, levensduur infinite
  koppelt MTU 1280 (ware MTU-koppeling 65515)
  huidige hop-limiet 128
  bereiktijd 25500 ms (basistijd 30000ms)
  herverzendingsinterval 1000ms
  DAD-verzendingen 0
```

3.2 Neighbor discovery protocol

Het neighbor discovery protocol is een complex protocol met meerdere functies. Het wordt gebruikt om de link-laag adressen te vinden van hosts op hetzelfde netwerk - zeg maar de ARP-functie[3] in IPv4. Voorts beschikt het ook over de mogelijkheid routers te vinden en ICMP redirect berichten te verzenden en nog heel wat extra's.

Hosts gebruiken neighbor discovery om routers te vinden op het netwerk waar op ze zich bevinden en om de configuratiegegevens van die routers op te halen. Hosts gebruiken dit protocol ook om de adressen en prefixen van naburige hosts te laten ontdekken.

Routers gebruiken neighbor discovery om zichzelf en hun configuratiegegevens kenbaar te maken. Routers zenden ook berichten naar hosts die een beter next-hop adres voorstellen voor een bepaalde bestemming.

Knooppunten gebruiken neighbor discovery om link-laag adressen te ontdekken of vast te te stellen wanneer die veranderen. Tevens wordt neighbor discovery door knooppunten gebruikt om vast te stellen of een ander knooppunt op dezelfde link bereikbaar is.

Een volledige discussie van het protocol zou ons te ver leiden. Daarvoor verwijzen we naar de rfc's 2461[12] en 2491[15]. Wat we wel zullen bespreken, zijn die onderdelen van het

[3] ARP is een methode om in IPv4 aan de hand van het IP-adres een opzoeking te doen naar het hardware-adres van een interface.

protocol die ons toelaten interfaces te configureren. Eerst zullen we het hebben over router advertisement en router solicitation. Dit zijn de berichten die ervoor zorgen dat knooppunten in het IPv6-netwerk kunnen komen zonder dat men die hoeft te configureren.

3.2.1 Router advertisement

Router advertisement pakketten worden uitgezonden door IPv6-routers. De router geeft hiermee aan dat hij een prefix heeft, zodat de hosts zich ernaar kunnen configureren. De router zal ook pakketten van de knooppunten aanvaarden en ze verder te routeren.

3.2.2 Router solicitation

Router solicitation is het expliciet aanvragen van een router advertisement pakket. Daar routers niet constant router advertisement pakketten uitzenden, kan dit nodig zijn wanneer de configuratie onmiddellijk moet gebeuren.

3.2.3 Neighbor solicitation

Dit zijn berichten die door een knooppunt uitgezonden worden om vast te stellen of een buur nog steeds aanwezig is, of om diens link-laag adres te achterhalen. Deze berichten kunnen ook gebruikt worden om dubbel gebruik van adressen te ontdekken.

3.2.4 Neighbor advertisement

Dit is dan het antwoord op vorig bericht. Het kan ook gebeuren dat een knooppunt zo'n bericht uitzendt als gevolg van een verandering van zijn linklaag-adres.

3.3 Automatische adresconfiguratie

IPv6 beschikt over 2 methodes om hosts zichzelf te laten configureren. Enerzijds hebben we *stateless* automatische adresconfiguratie en anderzijds hebben we *stateful* automatische adresconfiguratie. Ook een combinatie van beide manieren kan gebruikt worden.

- stateless
  De adressen worden bepaald aan de hand van router advertisement berichten. Deze berichten bevatten de adresprefixen en de hosts ondersteunen geen stateful (DHCPv6) adresautoconfiguratie.

- stateful
  Het grote verschil hierbij is dat een centrale server bijhoudt wie welk IP heeft. Deze server verspreidt ook configuratiegegevens voor de hosts en hoeft niet noodzakelijk op de lokale link te staan.

- beide
  De adressen worden configureerd aan de hand van stateless adresconfiguratie. Verdere configuratie gebeurt aan de hand van stateful automatische adresconfiguratie.

3.3.1 Stateless automatische adresconfiguratie

Er wordt eerst een lokaal linkadres gegenereerd aan de hand van de prefix fe80::/64 en de 64 bits interface-id. Hierbij wordt er gecontroleerd of het gegenereerde adres eventueel nog niet in gebruik is.

Vervolgens verzendt de host een router solicitation bericht. De router stuurt als antwoord daarop een router advertisement bericht terug met de configuratiegegevens. De host configureert zichzelf opnieuw aan de hand van het ontvangen prefix (ditmaal een prefix waarbij routing verder dan de router mogelijk is) en zijn interface-id. Opnieuw wordt er gecontroleerd of het adres nog niet in gebruik is.

In FreeBSD worden router advertisements standaard niet aanvaard, in Linux wel. Uiteraard moeten de kernels uitgerust zijn met IPv6-ondersteuning. Met sysctl[4] kan het gedrag bepaald worden. In FreeBSD is het de variabele
`net.inet6.ip6.accept_rtadv`, in Linux hebben we `net/ipv6/conf/all/accept_ra`.

Het uitzenden van router advertisement pakketjes gebeurt door een daemon. In FreeBSD hebben we `rtadvd`. Deze komt standaard met de IPv6-stapel voor FreeBSD. Voor Linux moet nog een apart programma geïnstalleerd worden: `radvd`. Radvd is te vinden op
`http://v6web.litech.org/radvd/`.

Windows XP aanvaardt router advertisements en beschikt ook over de mogelijkheid om zelf als router op te treden en router advertisement pakketten uit te zenden. Als volgt kunnen we Windows router advertisement pakketten laten uitzenden:

```
ipv6 ifc 4 forwards advertises
ipv6 rtu 2001:6a8:1904:10::/64 4 publish
```

[4]Dit is een tool waarbij het kernelgedrag kan aangepast worden terwijl het systeem aan het draaien is.

3.3.2 Statefull automatische adresconfiguratie: DHCPv6

Hiervan is op dit moment nauwelijks een implementatie beschikbaar. Ook is er nog geen RFC-document hiervan. Er zijn enkele *Internet drafts* beschikbaar over DHCPv6. Voor een volledige discussie verwijzen we naar `http://www.ietf.org/ids.by.wg/dhc.html`.

Voor FreeBSD en Linux is er een implementatie beschikbaar op `http://www.hycomat.co.uk/dhcp/`. Het project lijkt echter redelijk dood, men wacht op een rfc over dit onderwerp vooraleer men gaat implementeren. Windows XP biedt geen ondersteuning voor dit protocol.

3.4 IPv6-routing

We kunnen routers handmatig configureren of we kunnen routers zich laten configureren aan de hand van routingprotocollen zoals OSPFv6, RIPv6, enz...

3.4.1 Handmatige configuratie

Handmatige configuratie van routers is handig in omgevingen waar men weet dat de routes niet zullen veranderen. Dit is typisch het geval aan het uiteinde van een netwerk waar het niet meer mogelijk is om routes langs een andere router te laten lopen omdat er slechts één router is. Men spreekt ook wel van statische routing.

Men dient aan te geven dat men zich als router wenst in te stellen en dat kan in FreeBSD en Linux met behulp van sysctl. Voor FreeBSD is dat de variable `net.inet6.ip6.forwarding` die men op 1 moet zetten, voor Linux is dat `net.ipv6.conf.all.forwarding`.

Eens men zich als router heeft ingesteld kan men routes toevoegen aan de routingtabel. Nemen we bijvoorbeeld de situatie dat we we een NE2000-PCI netwerkkaart hebben en dat de router met adres 2001:6a8:1904:5::1 via deze netwerkkaart bereikbaar is. In FreeBSD noemt de interface dan ed met rangnummer 0.

```
bash# route add -inet6 2001:6a8:1904:5::1 -interface ed0
bash# route add -inet6 default 2001:6a8:1904:5::1
```

In Linux gaat het als volgt:

```
bash# route add -A inet6 2001:6a8:1904:5::1 eth0
bash# route add -A inet6 2000::/3 gw 2001:6a8:1904:5::1
```

We gebruiken hier niet de route default omdat de huidige linuxkernels daar blijkbaar problemen mee hebben. Daarom zetten we een route naar alle anycastadressen wat praktisch op hetzelfde neerkomt.

In Windows:

```
ipv6 rtu 2001:6a8:1904:5::1/128 4
ipv6 rtu ::/0 4/2001:6a8:1904:5::1 pub
```

3.4.2 Configuratie aan de hand van routingprotocollen

De bestaande IPv4 routingprotocollen hebben ook een equivalent gekregen in IPv6. We verwijzen naar het pakket zebra dat ondermeer ospf6d en ripngd bevat. Dit zijn de IPv6 implementaties van de bekende protocollen OSPF en RIP.

Hoofdstuk 4

Applicatielaag en IPv6

In dit hoofdstuk beschouwen we in hoeverre de huidige software klaar is om te werken met IPv6 en dat is reeds heel wat. Er is natuurlijk nog wel veel werk te doen. We kunnen drie stadia beschouwen. Ofwel werkt de software standaard reeds met IPv6, ofwel dient men de broncode te patchen, ofwel ondersteunt de software helemaal nog geen IPv6 en zijn er nog vertalingsmechanismen nodig om ook die met IPv6 te laten werken. Deze mechanismen worden besproken in het volgende hoofdstuk.

4.1 Servers

Qua serverapparatuur zit men voornamelijk nog op het patch-niveau. Weinig servers ondersteunen volledig IPv6, maar voor de meeste bestaan er patches die hen toelaten om via IPv6 bereikbaar te zijn. Hier volgt een tabel met services en servers die hun service ook via IPv6 aanbieden.

DNS	BIND8, BIND9
Proxy	Socks, wwwoffle, squid
SMTP	Sendmail, exim
POP3	Courier, Cyrus, QPopper
SSH	OpenSSH
DHCP	DHCPv6
News	Inn
FTP	pureftpd
HTTP	Apache 1.3+patch, Apache 2
IMAP	UW-IMAP, Courier,Cyrus

4.1.1 DNS

Met de aanpassing van de adressen moet het DNS-systeem ook heel wat uitgebreid worden. Eerst heeft men de AAAA-records ingevoerd in het DNS-systeem die qua systeem gelijklopend zijn met de klassieke A-records. Later heeft men de A6-records ontwikkeld die wat extra voordelen toevoegen aan het klassieke DNS-systeem.

AAAA-records

Om opzoekingen te doen van naam naar adres werd een nieuw type records uitgedacht zijnde AAAA-records. In de configuratiebestanden wordt dat dan:

```
@               IN      A       157.193.83.142
                IN      AAAA    2001:6a8:ff::1
                IN      AAAA    2001:6a8:1904:1::1
                IN      AAAA    2001:6a8:1904:2::1
```

Dit komt uit het zone-bestand voor abraham.elis.rug.ac.be. Als we een opzoeking doen naar abraham.elis.rug.ac.be krijgen we dus 4 records terug, 1 A-record en 3 AAAA-records.

Voor omgekeerde opzoekingen werd een nieuw domein opgezet, namelijk het IP6.INT domein. Hierbij kunnen we aan de hand van een IPv6-adres de bijhorende hostnaam opzoeken. De notatie is als volgt voor het eerste AAAA-record:

```
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.f.f.0.0.8.a.6.0.1.0.0.2.ip6.int
```

Dit is naar analogie van de omgekeerde IPv4-opzoekingen en dat is vreselijk. Gelukkig bestaat er een tool `ip6_int` die aan de hand van een IPv6-adres het omgekeerde genereert voor gebruik in configuratiebestanden. `ip6_int` is te vinden op `http://www.bieringer.de/linux/IPv6/tools/`.

Stel dat we bijvoorbeeld voor het netwerk 2001:6a8:1904::/48 omgekeerde opzoekingen wensen te doen. In het configuratiebestand van BIND staat er:

```
zone "4.0.9.1.8.a.6.0.1.0.0.2.IP6.INT" {
        type master;
        file "db.2001.6a8.1904";
};
```

In het bestand db.2001.6a8.1904 definiëren we vervolgens de rest van het domein. We hebben dan records als:

```
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0 IN      PTR     abraham.elis.rug.ac.be.
```

A6-records

Het bleek dat het AAAA-record systeem iets te snel uit de grond was gestampt en dat men ook enkele nieuwe opties wilde hebben in DNS. Stel dat een klant van provider zou willen veranderen. Het hernummeren van het netwerk zou dan vloeiend moeten gebeuren. Dat kan makkelijk aan de hand van router advertisement en dit zou ook in DNS moeten opgevangen worden. Ook multihoming[1] zou makkelijker moeten kunnen gaan met A6. Zo heeft men dus nog een extra transitie opgebouwd van AAAA- naar A6-records. Op dit moment worden A6-records nauwelijks gebruikt.

Een A6-record kan een volledig of gedeeltelijk IP-adres bevatten met in het laatste geval extra informatie waar de rest van het IP-adres kan gevonden worden. Men werkt met A6-record-kettingen. Laten we een voorbeeld bestuderen om het gebruik te verduidelijken.

Veronderstel een domein rug6 dat *multihomed* is via 2 providers XS26 en Belnet. XS26 krijgt zijn prefix van 6 bone. Belnet krijgt zijn prefix van ripe en gebruikt dus productieadressen. Er kunnen daar nog meerdere providers tussenin liggen, maar dat is hier niet het geval.

XS26 kreeg 3FFE:80E0::/28 van 6bone en Belnet kreeg 2001:6a8::/35 van RIPE. XS26 kent nu 3ffe:80ee:0511::/48 en Belnet kent 2001:6a8:1904::/48 toe aan rug6. Laten we nu een knooppunt oerbier onderstellen op rug6 dat intern op subnet 3 zit en interface-id 250:fcff:fe44:8d4 heeft.

Dan heeft oerbier de volgende adressen:

```
2001:06a8:1904:0003:0250:fcff:fe44:08d4
3ffe:80ee:0511:0003:0250:fcff:fe44:08d4
```

In de DNS van rug6 komt dan:

```
$ORIGIN rug6.
oerbier     A6 64 ::250:fcff:fe44:8d4 zeus
zeus        A6 48 0:0:0:3:: IP6
IP6         A6 48 0::0 ns.belnet.be.
IP6         A6 48 0::0 ns.xs26.net.
```

In die van Belnet, en die van XS26 en die van tla komt:

[1] Hierbij is eenzelfde website bereikbaar via verschillende IPv6-adressen. Als 1 link wegvalt is er nog altijd een andere en dus hebben we een betere beschikbaarheid.

```
$ORIGIN belnet.be.
ns  A6 35 0:0:1904:: ripe.tla.org.

$ORIGIN xs26.net.
ns A6 28 0:E:0511:: 6bone.tla.org.

$ORIGIN tla.org.
ripe A6 0 2001:6a8::
6bone A6 0 3ffe:80E0::
```

Stel dat men nu een opzoeking doet naar oerbier en men komt terecht op de nameserver van rug6. Daar komt men reeds de laatste 80 bits te weten. Voor de eerste 48 bits moet men gaan zoeken in ns.belnet.be of ns.xs26.net. Daar bekomt men respectievelijke de volgende 13 en 20 bits te weten. Voor de laatste bits dient men nog een niveau hoger te gaan.

Het voordeel van deze methode is dat een site makkelijk van provider kan veranderen of een extra provider kan nemen. Het nadeel is dat als één van de tussenliggende nameservers plat is de opzoeking faalt. Men kan echter nog steeds de oude manier van werken implementeren als volgt:

```
$ORIGIN rug6.
oerbier     A6 0    2001:06a8:1904:0003:0250:fcff:fe44:08d4
                    3ffe:80ee:0511:0003:0250:fcff:fe44:08d4
```

Voor omgekeerde opzoekingen wordt gebruik gemaakt van het *bitstring* formaat. Men gebruikt hiervoor niet het ip6.int domein maar ip6.arpa domein. Het domein 2001:6a8:1904::/48 wordt in bitstringnotatie:

```
\[x200106a81904/48]
```

In een DNS bestand wordt dan het omgekeerd opzoeken van oerbier:

```
$ORIGIN \[x200106a81904/48].ip6.arpa.
\[x0003:0250:fcff:fe44:08d4/80] IN PTR oerbier.rug6.
```

Voorts heeft men DNAME . Veronderstellen we dat de 6bone-zone wordt bijgehouden in 6bone.org. DNAME geeft dan aan in welk zonebestand we moeten zijn voor een bepaald prefix, om zo af te dalen tot het volledige adres.

```
$ORIGIN IP6.ARPA.
\[x3ffe/16] DNAME 6bone.org.
```

Vervolgens is er een toekenning van het prefix 3ffe:80e0::/28 aan XS26.

```
$ORIGIN \[x3ffe/16].ip6.arpa.
\[x80e/12] DNAME xs26.net.
```

Vervolgens is er een toekenning van het prefix 3ffe:80ee:0511::/48 van XS26 aan rug6.

```
$ORIGIN \[x3ffe80e/28].ip6.arpa.
\[xe0511/20] DNAME rug6-rev.
```

In rug6 hebben we een host oerbier.rug6.

```
$ORIGIN \[x3ffe80ee0511/48].ip6.arpa.
\[x0003:0250:fcff:fe44:08d4/80] IN PTR oerbier.rug6.
```

Hoewel A6-records meer mogelijkheden bieden is het toch de vraag of dit wel de moeite loont. Het compliceert het systeem heel wat en als we dan toch van provider wensen te veranderen zouden we toch ook gewoon een script kunnen schrijven om de zonebestanden aan te passen. Het is nu wel zo dat rfc 2874[22] het gebruik van AAAA-records afraadt en aanmoedigt om over te schakelen naar A6-records. De vraag blijft of dit zal gebeuren. De meeste resolvers kunnen enkel met AAAA-records werken.

DNS-servers

Als DNS-server hebben we de alomgekende applicatie BIND. Er zijn ruwweg 3 reeksen in gebruik: BIND4, BIND8 en BIND9. De eerste begint stilaan in onbruik te raken, maar er bestaan patches om de nieuwe AAAA-records te ondersteunen. BIND8 verstaat standaard AAAA-records, maar kan nog steeds niet aangesproken worden via IPv6. BIND9 kent de nieuwe A6-records en AAAA-records en kan via IPv6 aangesproken worden. BIND9 heeft dus volledige IPv6-support.

4.1.2 Proxies

We onderscheiden proxies die werken op de applicatielaag (deze werken meestal enkel voor de protocollen HTTP en FTP) en proxies als Socks die op de transportlaag werken. Een uitgebreide bespreking van het Socks-mechanisme is te vinden in het volgende hoofdstuk.

Squid komt met een patch voor een oude proxyserver. WWWoffle is een relatief onbekende proxyserver met native IPv6-ondersteuning. Hij is kleinschalig; maar dat siert de

proxyserver. WWWoffle is te vinden op `http://wwwzenger.informatik.tu-muenchen.de/persons/huckle/bugse.html`.

Proxyservers kunnen als een handig koppelstuk dienen tussen een IPv4- en een IPv6-domein. De aanvragen naar een IPv4-server van een IPv6-host kunnen perfect via zo'n proxyserver lopen.

4.1.3 DHCP

DHCPv6 is meer dan het zomaar ondersteunen van IPv6. De specificatie ervan bevindt zich nog in draft-status en wordt ontwikkeld door de dhcp-werkgroep. De laatste drafts zijn te vinden op `http://www.dhcp.org/`.

4.1.4 Veelgebruikte webservices

De meestgebruikte internetservices kunnen bijna ook alle reeds gepatcht worden voor IPv6. Apache 2 en opensshd ondersteunen reeds standaard IPv6 ondersteuning. Zie de tabel voor een overzicht.

4.2 Gebruikersprogramma's

De webservices die dagelijks gebruikt worden zoals news, ftp, email en www worden het best ondersteund. Van de 3 grote browsers (Mozilla/Netscape, Internet Explorer, Opera) ondersteunen 2 van de 3 reeds IPv6. Enkel Opera blijft achter zonder IPv6-ondersteuning.

ssh

Ssh (secure shell) kan eveneens gebruikt worden om een IPv6-host te voorzien van IPv4 connectiviteit. Dit kan door een ssh-tunnel te leggen naar een dual stack knooppunt (dit is een knooppunt dat zowel IPv4 als IPv6 ondersteunt). Stel dat we bijvoorbeeld een IPv6-host 6star hebben en een dual stack knooppunt digital. We wensen te surfen door middel van een proxyserver die enkel via IPv4 te bereiken is: proxywww. De proxyserver draait hierbij op poort 8080. Als volgt kunnen we dan browsen:

```
ssh -L 8080:proxywww.rug.ac.be:8080 kristof@digital
```

We leggen de tunnel aan, loggen in en zetten onze proxyserver als localhost op poort 8080. Bij deze ssh-tunnel luistert de proxyserver nu zowel op 127.0.0.1 als op ::1 op poort 8080.

browser	mozilla/Netscape, Internet Explorer, wget, w3m, lynx
ftp	ncftp, wget
mail	mozilla, fetchmail, sylpheed
news	mozilla, slrn, sylpheed
ssh	openssh, putty, teraterm ssh
mp3	xmms, mpg123
irc	irssi, xchat, bitchx

Hoofdstuk 5

Migratie van IPv4 naar IPv6

Dit hoofdstuk vormt het belangrijkste van deze thesis. Hierin bespreken we de organisatorische problemen bij de overschakeling naar IPv6. Men zal doorgaans wensen dat het netwerk ook nog via IPv4 bereikbaar is. Sterker nog, men wil tijdelijke onbeschikbaarheid zoveel mogelijk vermijden of zo kort mogelijk houden. Dat bij overschakeling naar IPv6 er enkele veiligheidsrisico's zijn zal niemand ontkennen, dus ook deze moet men in het oog houden. De veiligheid wordt in het volgende hoofdstuk besproken.

Een andere vraag is hoe je de overschakeling maakt. Het is duidelijk dat er niet een magische dag kan zijn waarbij je van het ene moment op het andere alles omschakelt. Hypothetisch gesproken kan dit wel, maar je moet gek zijn om het ook te doen. Het is niet noodzakelijk en om de beschikbaarheid van het netwerk te waarborgen doe je het dus beter ook niet.

We zullen bespreken met welke setup je kan beginnen, je netwerkconnectiviteit verkrijgt als IPv6 ver van je LAN ligt. Ook de vertalingsmechanismen zullen worden besproken. Met andere woorden: hoe kunnen we computers en programma's die de verschillende IP protocollen spreken met elkaar laten communiceren? Er zijn heel wat combinaties mogelijk, en dus zijn er ook heel wat mechanismen.

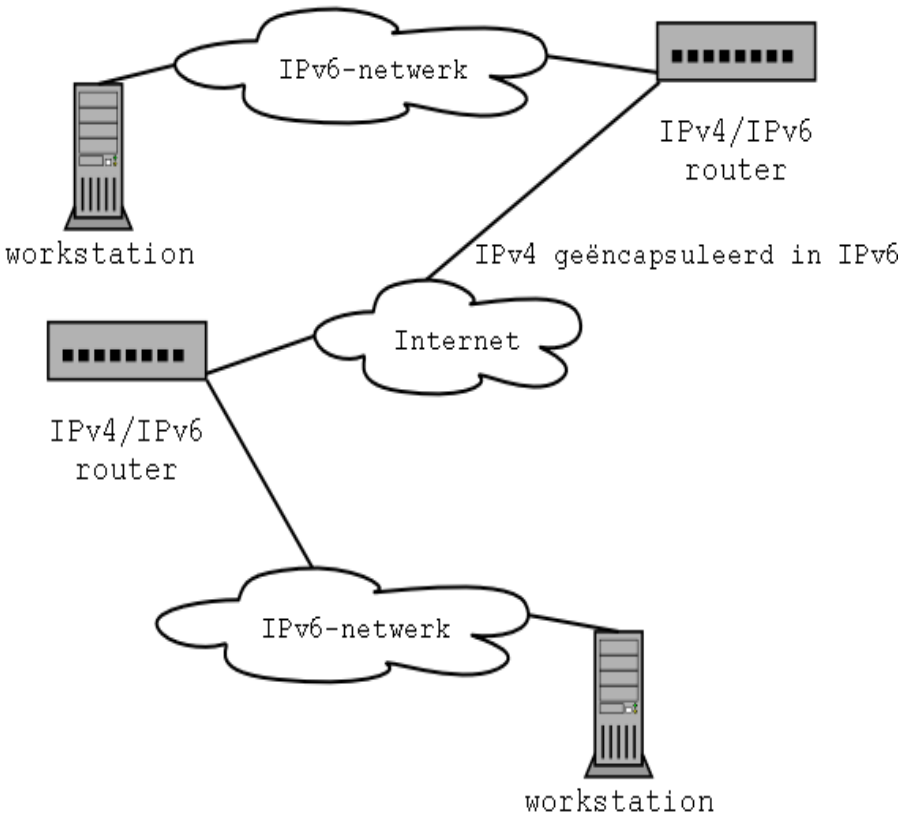
5.1 Dual stack

De aanpak die het meest wordt gebruikt en in het begin het meeste zekerheid biedt, is de *dual stack*-methode. Hierbij laten we een knooppunt zowel IPv4 als IPv6 spreken. Daar we met verschillende protocollen en niet met verschillende versies van hetzelfde protocol te maken hebben is dit geen probleem. Dus het knooppunt blijft met IPv4 werken. Hij krijgt nu gewoon extra functionaliteit.

5.2 IPv6-eilanden verbinden

Hoe raakt men nu op het IPv6-netwerk? Voorziet uw provider u van IPv6-adressen en bent u direct via IPv6 met uw provider verbonden dan heeft u de meest ideale situatie. Op dit moment is dit echter zeer zelden het geval. In het beste geval kunt u een IPv6-blok verkrijgen bij uw provider. Indien u niet direct met het IPv6 netwerk verbonden bent dan dienen IPv6 pakketten over het IPv4 netwerk verzonden te worden.

IPv4 kan echter niet zomaar overweg met IPv6-pakketten: er wordt dan aan de IPv6 pakketten een IPv4-header toegevoegd. Het IPv6-pakket wordt dus de inhoud van een IPv4-pakket. Eens het pakket weer verder op het IPv6-netwerk kan, wordt het IPv4-jasje van het pakket ontdaan en kan het pakket verder naar zijn bestemming. Dit proces wordt respectievelijk encapsulatie en decapsulatie genoemd. Op figuur 5.1 zien we een typische situatie.



Figuur 5.1: IPv6-eilanden verbinden

5.2.1 6to4

Zoals we reeds zagen is het prefix 2002::/16 gereserveerd voor 6to4. 6to4 is een methode waarbij we aan de hand van ons IPv4-adres automatisch zelf een IPv6-blok hebben. Het enige wat we nog moeten doen is een route leggen naar het IPv6-netwerk.

Er bestaan publieke 6to4 routers die verkeer aanvaarden van knooppunten die met 6to4 geconfigureerd zijn. Voor IPv6 telt het verkeer van het knooppunt naar de IPv6-router als één link. In de praktijk liggen er nog verschillende IPv4-netwerken tussen die u brengen tot die router.

We willen natuurlijk wel de dichtsbijzijnde router gebruiken zodat de snelheid van ons IPv6-netwerk aanvaardbaar is. Daarvoor is gelukkig een anycast adres voorzien: 2002:c058:6301:: Dit adres is de 6to4-vorm van het IPv4 adres 192.88.99.1. Voor meer informatie over dit anycast adres en de routing naar dit adres verwijzen we naar RFC3068[28].

Nu hebben we zelf een /48 netwerk en zenden we ons IPv6-verkeer naar een router. Er is wel een tussenliggend IPv4-netwerk. De pakketten worden dus geëncapsuleerd in IPv4-pakketten en naar hun bestemming verzonden. Ons /48 netwerk kunnen we nu vrij indelen. Eén IPv4-adres volstaat dus om quasi elk netwerk van om het even welke organisatie te voorzien van IPv6-adressen.

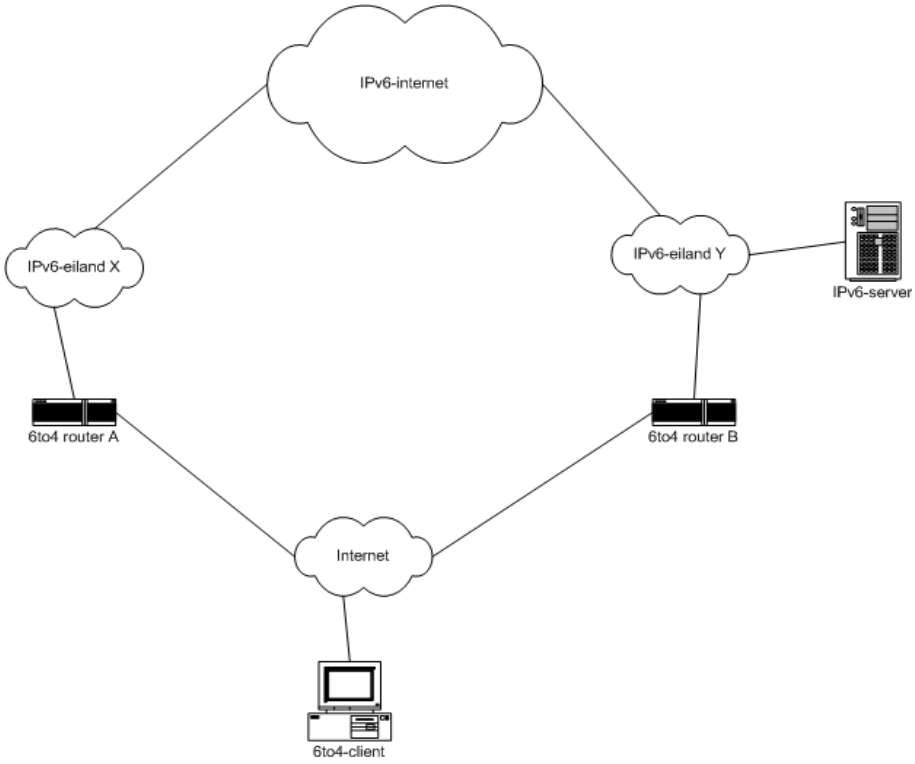
De routing bij 6to4 kan soms nogal vreemde bokkesprongen maken. Hiermee bedoelen we dat de router waarop u uw pakketten verzendt niet noodzakelijk de router is waar uw pakketten op toekomen. Daarom onderscheiden we alle mogelijke gevallen van connecties om de routing wat van dichterbij te bekijken.

Bij een connectie tussen twee knooppunten die met 6to4 geconfigureerd zijn, ligt dus de routing volledig vast. Het decapsuleren gebeurt in de eindpunten dus de routing is dezelfde als ware het een IPv4-connectie.

Als we verkeer hebben van een 6to4-knooppunt naar een IPv6-knooppunt gaat, dat niet met 6to4 geconfigureerd is, dan gaat het eerst naar onze default open relay router en wordt het van daar verder gerouteerd naar zijn uiteindelijk bestemming.

De terugweg is het meest opmerkelijk. Hierbij vertrekt het verkeer van op een IPv6-knooppunt en wordt het gerouteerd naar een knooppunt dat met 6to4 configureerd is en van daar terug naar het eerste knooppunt. Men weet dus wel langs welke 6to4 router men het pakket verzendt, maar niet langs dewelke het terug zal keren. Op figuur 5.2 zien we dat een pakket langs router A het IPv6-Internet op gaat, maar via router B zal terugkeren naar de client.

Laten we nu naar de configuratie zelf overgaan. We nemen de situatie met het IP adres 212.100.182.44 en we gebruiken de default anycast 6to4-router met IP 2002:c058:6301::. Dan kunnen we ons IPv6-adres berekenen. Hierbij kiezen we voor het netwerk nul en voor het hostadres 1. Als men strikt de rfc's wenst na te leven zou men ook nog de EUI-64 identifier



Figuur 5.2: 6to4

moeten bereken. Een berekening kan bijvoorbeeld gebeuren aan de hand van deze regel in shellscript (die we leenden uit de Linux-IPv6-HOWTO[35]):

```
ipv4="212.100.182.44"; printf "2002:%02x%02x:%02x%02x::1" ‘echo $ipv4 | tr "." " "‘
```

FreeBSD

In FreeBSD wordt er gebruik gemaakt van de stf-interface. Dit is de 6-to-4 tunnel pseudo-device. Men dient de regel

```
pseudo-device   stf
```

in de kernel te compileren. We configureren de stf-interface en voegen de default route toe.

```
root# ifconfig stf0 inet6 2002:d464:b62c::1
root# route add -inet6 default 2002:c058:6301::
```

Linux

In Linux gebruiken we het commando *ip* om tunnels te configureren; met *ifconfig* en *route* is het ook mogelijk. We kiezen hier (en verder) echter ip boven ifconfig omwille van de grotere flexibiliteit van ip.

We voegen hier een tunneldevice toe dat we tun6to4 zullen noemen (dit kan om het even welke naam zijn). *mode sit* wil hier zeggen dat we IPv6-verkeer zullen tunnelen in IPv4 pakketten. Bij 6to4 zijn we nooit zeker vanwaar het andere verkeer komt, dus: *remote any*. Daarna brengen we de tunnel online en kennen we ons 6to4-adres aan de tunnel toe.

```
digital:~# ip tunnel add tun6to4 mode sit remote any local 212.100.182.44
digital:~# ip link set dev tun6to4 up
digital:~# ip -6 addr add 2002:d464:b62c::1/16 dev tun6to4
```

Windows XP

Interface 3 in Windows is het 6-to-4-pseudo-device. Windows configureert al automatisch een IPv6-adres voor deze interface aan de hand van je IPv4-adres. Indien je meerdere niet-lokale IPv4-adressen hebt krijgt deze interface ook meerdere 6to4 adressen.

```
Interface 3: 6to4-tunneling-pseudo-interface
  gebruikt geen Neighbor Discovery
  gebruikt geen Router Discovery
    preferred global 2002:d464:b648::d464:b648, levensduur infinite
  koppelt MTU 1280 (ware MTU-koppeling 65515)
  huidige hop-limiet 128
  bereiktijd 18000 ms (basistijd 30000ms)
  herverzendingsinterval 1000ms
  DAD-verzendingen 0
```

Volgens de Windows documentatie zou alles direct moeten werken (Ware het niet niet dat de default route gezonden wordt naar een router van Microsoft en die blijkt nogal plat te zijn.). Dus men herlegt de default route naar het IPv6-anycast-adres voor 6to4 gateways.

```
ipv6 rtu ::/0 3/2002:c058:6301::
```

5.2.2 Geconfigureerde tunnel

Een geconfigureerde tunnel is een IPv6-in-IPv4-tunnel waarbij er aan beide uiteinden van de tunnel configuratie nodig is. Het ene eindpunt van de tunnel ligt bij de provider, het andere ligt bij u. Het eindpunt van de provider ligt meestal vast en dus kan men met behulp van een script gemakkelijk het netwerk configureren. Bij een geconfigureerde tunnel weet echter de provider niet naar welk IPv4-adres een bepaald IPv6-blok moet gerouteerd worden. Dus ook de provider moet dit weten.

Indien aan beide kanten met vaste IP-adressen gewerkt wordt dan is er slechts eenmalig een overeenkomst nodig tussen provider en gebruiker. Echter, de meeste gebruikers op het Internet hebben geen vast IPv4-adres, maar sommigen willen toch wel eens experimenteren met IPv6. Daarom bestaan er nu verschillende *tunnel brokers*. Een tunnel broker is een provider die voorziet in IPv6 adressen en die ook een manier voorziet om het eindpunt van de tunnel te veranderen. Eerst zullen we de eenvoudigste situatie beschouwen, namelijk die met 2 vaste IPv4-adressen.

Vast IP-adres bij de klant

Laten we er van uitgaan dat om organisatorische redenen de provider steeds een vast IP-adres gebruikt aan zijn eindpunt van de tunnel. Als de klant ook een vast IP-adres heeft kan men een permanente IPv6-tunnel opzetten. De klant krijgt een blok en zet zijn default route naar het eindpunt van de provider. De provider op zijn beurt laat de route naar dat blok naar het eindpunt van de gebruiker gaan.

In deze sectie zullen we met benoemde eindpunten werken. Dat wil zeggen dat de eindpunten van de tunnel zowel een IPv4 als een IPv6-adres hebben. Dikwijls zal men echter hier geen extra adressen voor uittrekken en gebruikt men ingebedde IPv4-adressen.

In het geval van onbenoemde eindpunten wordt het verkeer gewoon rechtstreeks door de interface gestuurd en wordt er niet eerst een route toegevoegd door de interface voor het andere eindpunt.

FreeBSD Onder FreeBSD hebben we de gif-interface. Dit is een tunnelende interface met verschillende mogelijkheden. Hij kan IPv4 in IPv6 en omgekeerd tunnelen. Tevens kan deze interface IPv4 in IPv4 tunnelen en IPv6 in IPv6. Ons interesseert echter de IPv6 in IPv4 tunneling.

Laten we dit even demonstreren met een concreet voorbeeld dat we gebruikten in ons netwerk. We hebben een eindpunt in Belnet en een eindpunt in ELIS, respectievelijk met de IP's

193.190.197.234 en 157.193.83.142. De IPv6-adressen van de eindpunten zijn 2001:6a8:ff:: en 2001:6a8:ff::1. Dit is de configuratie aan de kant van de klant. Aan de serverside gebeurt iets analoogs.

```
ifconfig gif0 create
gifconfig gif0 157.193.83.142 193.190.197.234
ifconfig gif0 inet6 2001:6a8:ff::1/128 2001:6a8:ff::
route add -inet6 default 2001:6a8:ff::
```

Hierbij maakten we eerst de tunnelinterface aan. Support hiervoor dient men mee te compileren in de kernel. Daarvoor moeten we volgende regel in de kernel zetten:

```
pseudo-device   gif     4       # IPv6 and IPv4 tunneling
```

We geven als argument mee hoeveel dergelijke devices we wensen te hebben. Dan leggen we de tunnel van de klant naar de provider. We benoemen de eindpunten. Uiteindelijk voegen we nog een route toe naar de default router.

In FreeBSD kunnen we echter ook het volgende doen als we met onbenoemde eindpunten werken.

```
route add -inet6 default -interface gif0
```

Op dit moment werkt dit echter enkel met FreeBSD 4.5-STABLE na ergens halverwege april 2002. Daarvoor zat er een bug in de software.

Linux In Linux wordt gebruik gemaakt van de sit-interface (Simple Internet Transition).

```
ip tunnel add sit1 mode sit remote 193.190.197.234 local 157.193.83.142
ifconfig sit1 up
route -A inet6 add 2000::/3 dev sit1
ip addr add 2001:6a8:ff::1/128 dev sit1
```

De tunnel wordt aangelegd en online gebracht, vervolgens wordt de default route toegevoegd (aangepast aan Linux dan) en het eindpunt benoemd. Linux blijkt niet het probleem te hebben dat FreeBSD had. We kunnen uiteraard ook eerst een route naar de default router toevoegen en de default route daarnaar laten wijzen. Dit zal ook werken.

Windows XP We hebben 2 manieren om een geconfigureerde tunnel te leggen in Windows XP.

```
netsh inteface ipv6 add v6v4tunnel interface=abraham localadres=212.100.182.32\
remoteaddress=157.193.83.142

ipv6 ifcr v6v4 212.100.182.32 157.193.83.142
```

Verder dienen we nog een IP-adres toe te kennen en de default route toe te voegen. Dat kan als volgt:

```
ipv6 adu 5/2001:6a8:1904:5::1
ipv6 rtu ::/0 5
```

Uiteindelijk is onze interface geconfigureerd:

```
Interface 5: Geconfigureerde tunnelinterface
  gebruikt geen Neighbor Discovery
  gebruikt geen Router Discovery
  adres van Link-laag: 212.100.182.32
  adres van externe Link-laag: 157.193.83.142
    preferred global 2001:6a8:1904:5::1, levensduur infinite (handmatig)
    preferred link-local fe80::5:d464:b620, levensduur infinite
  koppelt MTU 1280 (ware MTU-koppeling 65515)
  huidige hop-limiet 128
  bereiktijd 21500 ms (basistijd 30000ms)
  herverzendingsinterval 1000ms
  DAD-verzendingen 1
```

Dynamisch IP-adres bij de klant

Indien we aan 1 kant van de tunnel over een dynamisch IP-adres beschikken dan dienen de tunneleindpunten regelmatig aangepast te worden. Een standaard geconfigureerde tunnel volstaat hier niet meer. Men heeft dit geautomatiseerd naar het concept *tunnel broker* (zie verder).

5.2.3 Automatische tunnel

Bij automatische tunneling wordt er gebruik gemaakt van IPv4-compatibele adressen. Het andere eindpunt van de tunnel is dus onmiddellijk bepaald. Hierbij mag enkel gebruik gemaakt

worden van globale IPv4-adressen.

De communicerende knooppunten zijn beide dual stack en zijn beide geconfigureerd met een automatisch tunneladres zijnde ::IPv4-adres. Er dient ook een route aanwezig te zijn voor automatische tunneling. Dit is een route naar het prefix ::/96.

FreeBSD heeft geen ondersteuning voor automatische tunneling. Linux werkt via de sit interface en heeft ook een entry in de routingtabel. Men dient sit0 online te brengen - tenzij men al andere tunnels zou hebben die sit gebruiken. Dan wordt een route toegevoegd naar ::/96 en kan men automatische tunneling gebruiken. In tegenstelling tot wat rfc2893[23] stelt, kan men wel ::127.0.0.1 gebruiken.

In Windows XP is interface 2 voorzien voor automatische tunneling. De configuratie gebeurt hier ook automatisch.

```
Interface 2: Automatische tunneling-pseudo-interface
  gebruikt geen Neighbor Discovery
  gebruikt geen Router Discovery
  adres van Link-laag van router: 0.0.0.0
  In EUI-64 ingesloten IPv4-adres: 0.0.0.0
    preferred link-local fe80::5efe:212.100.182.51, levensduur infinite
    preferred global ::212.100.182.51, levensduur infinite
  koppelt MTU 1280 (ware MTU-koppeling 65515)
  huidige hop-limiet 128
  bereiktijd 25500 ms (basistijd 30000ms)
  herverzendingsinterval 1000ms
  DAD-verzendingen 0
```

5.2.4 Tunnel broker

Het manueel configureren van IPv6-tunnels vraagt nogal wat administratie en dus hebben we nood aan een automatisme om IPv6-tunnels om te zetten. Steeds meer particulieren wensen ook eens IPv6 uit te proberen. Als deze dan nog over een dynamisch IP beschikken is een gewone geconfigureerde tunnel onbegonnen werk.

Wat is nu een tunnel broker? Het is een provider van IPv6-adressen voor gebruikers die reeds via IPv4 op het Internet zitten. Gebruikers kunnen kiezen tussen verschillende tunnel brokers naargelang hun behoeftes. Iedereen wenst uiteraard een zo laag mogelijke pingtijd, een zo laag mogelijke prijs en een zo groot mogelijk prefix.

De opstelling bestaat uit de gebruiker die een dual-stack-knooppunt heeft, een tunnel broker

met daarachter de tunnelservers en eventueel een DNS-server. De gebruiker connecteert met de tunnelbroker en past daar zijn gegevens aan. De tunnelbroker kan tunnels aanmaken, verwijderen of het eindpunt aanpassen. Hij kan ook de DNS-informatie aanpassen.

De tunnel server is een dual-stack router die geïnstrueerd wordt door de tunnelbroker en dus de tunnels aanpast. Voorts kan een tunnel broker ook nog een connectie hebben met een DNS-server.

De tunnel broker gebruiken

De klant connecteert met de tunnel broker server en authentificeert zich. De klant moet zeggen wat zijn IPv4-adres is zodat de tunnel aangepast kan worden. De klant kan ook eventueel een DNS-server opgeven waar de omgekeerde opzoekingen zullen gedaan worden voor het IPv6-blok dat hij toegewezen krijgt.

Aan de kant van de tunnel broker wordt een tunnel server gekozen en een IPv6-prefix dat aan de klant wordt toegewezen (meestal is dat ofwel een /48 (site prefix), een /64 (subnet prefix) of een /128 (host prefix)). Er wordt ook een leeftijd voor de tunnel gekozen en de DNS-server wordt aangepast. Uiteindelijk wordt het eindpunt aangepast en verslag uitgebracht aan de klant.

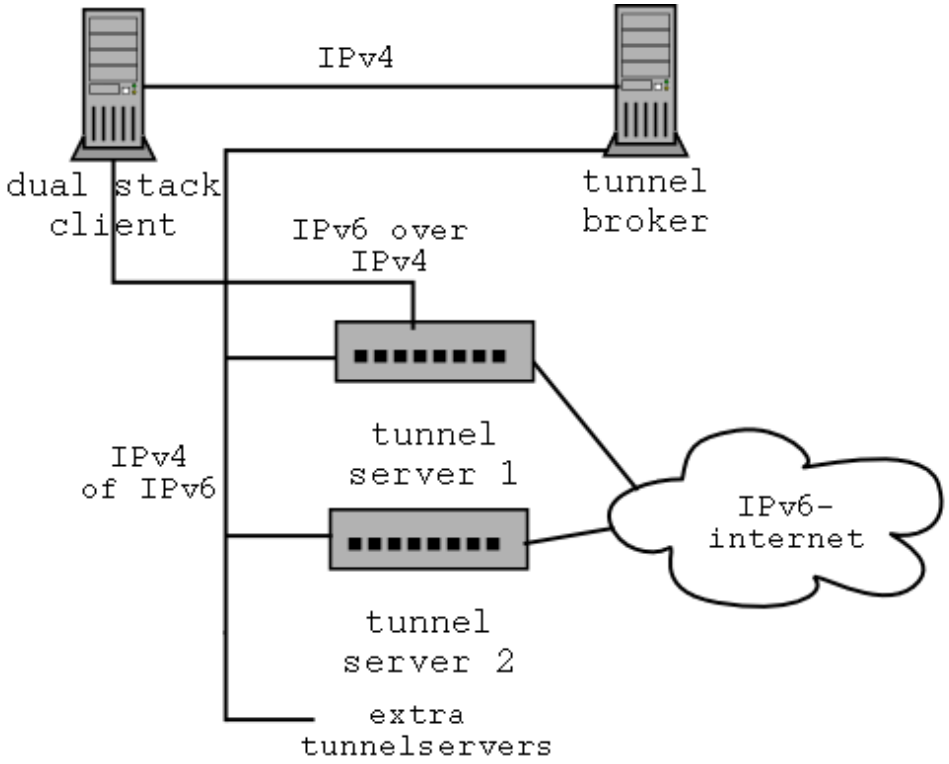
De IPv6-adressen die toegekend worden, moeten uiteraard ook routeerbaar zijn op het globale IPv6-netwerk. De tunnelservers moeten reeds verbonden zijn hiermee. Ook moet de provider effectief over de IPv6-adresruimte beschikken die hij zelf verder toekent.

De interactie tussen klant en tunnelbroker gebeurt doorgaans via een webinterface. De klant authenticeert zich via een formulier en krijgt onmiddellijk feedback. Het nadeel hiervan is dat de klant aan zijn zijde alles manueel moet doen. Een oplossing hiervoor is het tspc-protocol van Freenet6.

Bestaande tunnelbrokers

Wanadoo Wanadoo is een Belgische provider die tevens voorziet in IPv6-connecties. Je kunt een IPv6tunnel aanvragen op `http://tunnel.be.wanadoo.com/`. Je krijgt een IPv6-adres op de 6-bone.

XS26 XS26 is een grotere tunnel broker met veel meer mogelijkheden. Je krijgt een /48 netwerk en je kan per account ook meerdere tunnels configureren. Je kan per /48 een eigen nameserver draaien. Het uitwisselen van informatie verloopt ook volledig via een webinterface met bevestigingsemails.



Figuur 5.3: tunnelbroker

xs26.net bestaat uit een netwerk van tunnel servers. Je kan afhankelijk van je locatie de dichtstbijzijnde tunnelserver kiezen. Dit resulteert in goede pingtijden naar de tunnelserver en dus ook naar het IPv6-netwerk.

Freenet6 Freenet6 is een beetje een buitenbeentje in de tunnelbrokers. Het aanpassen van de tunnelinformatie gaat niet via een webinterface, maar via een eigen clientprogramma en protocol. Het voordeel hiervan is dat de klant niet meer zelf scripts hoeft te schrijven om de webinterface aan te spreken en dat alles automatisch gaat door eenmalig een configuratiebestand aan te passen. Ook hier krijg je een /48 toegewezen als je al een paswoord gebruikt, zo niet krijg je een /64. Er is mogelijkheid om zelf DNS te draaien voor je account.

Tunnelbrokersoftware

De meeste tunnelbrokers werken met cgi-scripts om dynamisch de tunnels aan te passen. We hebben op FreeBSD eens een poging gedaan met PHP, sudo en Apache. Sudo werd gebruikt om als gebruiker www het commando gifconfig te kunnen uitvoeren dat het tunneleindpunt

aanpast. PHP werd gebruikt om via een webinterface data door te geven aan sudo en gifconfig. Apache was de webserver hierbij. Veiligheid kan eventueel bekomen worden door nog een mysql database daar aan te hangen en met authenticatie te werken. Men bezoekt de site dan best ook via https. De veiligheid van het system commando in PHP dat commando's als de gebruiker www uitvoert en sudo valt ook te betwijfelen.

Microsoft heeft ook een implementatie gemaakt van een tunnel broker. Deze is te vinden op `ftp://ftp.research.microsoft.com/users/msripv6/broker-1.1.exe`.

5.2.5 Overige methodes

Verder hebben we nog 2 andere methodes om IPv6 over IPv4 te tunnelen. De eerste is 6over4 en is beschreven in rfc2529. 6over4 biedt ook mogelijkheden om multicast te tunnelen en is ondersteund in Windows XP. De mensen van Microsoft ontwikkelden nog een andere manier van IPv6 over IPv4 die luistert naar de naam ISATAP. ISATAP bevindt zich echter nog in draftstatus. Voor meer info verwijzen we naar [40].

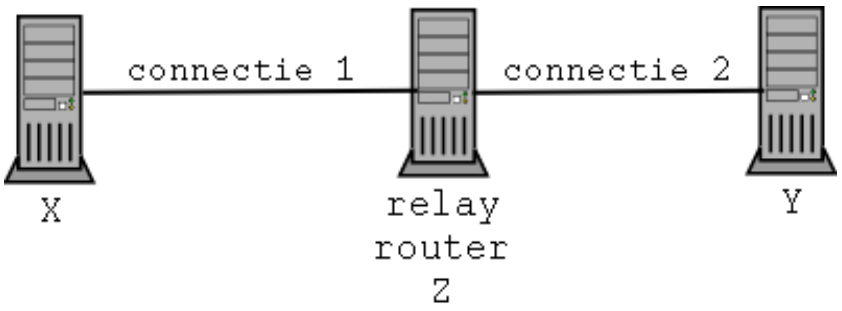
5.3 Vertalingsmechanismen

5.3.1 Transport Relay Translator (TRT)

Hier bespreken we een vertaler die werkt op de transportlaag. We wensen knooppunten die enkel IPv6 spreken te laten communiceren met knooppunten die enkel IPv4 spreken. Dit systeem vertaalt TCP en UDP over IPv6 naar TCP en UDP over IPv4 en vice versa.

Het grote voordeel van dit systeem is dat noch op de IPv4 knooppunten noch op de IPv6 knooppunten aanpassingen dienen te gebeuren. Dit vertalingsmechanisme staat op een apart dual stack systeem.

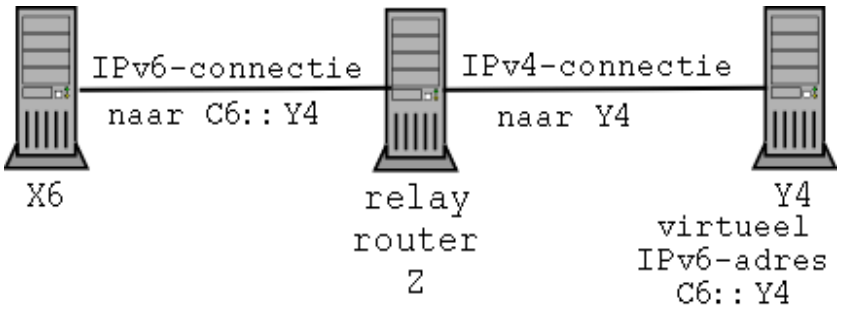
Het nadeel van dit systeem is dat het een *single point of failure* vormt. Een ander nadeel is dat de vertaling ook last heeft met NAT onvriendelijke protocollen (bv. IPsec).



Figuur 5.4: Transport relay translator

Om het mechanisme duidelijk te maken zullen we eerst een TCP-relay voor IPv4 naar IPv4 bespreken. (zie figuur 5.4) Voor UDP is alles analoog. Een zender(X) zendt een pakket naar een bestemming(Y). Het pakket komt op een gegeven moment aan bij de TCP-relay(Z). In plaats van pakket verder te sturen maakt de TCP-relay de connectie met de zendende host pretenderend dat hij Y is. Nu maakt de TCP-relay ook nog een tweede TCP-connectie met de bestemming Y. Hierbij gebruikt hij zijn eigen IP als zender. Dus als er een connectie wordt gemaakt van X naar Y via Z, dan lijkt het op Y dat men vanaf Z een connectie maakt.

Laten we dit mechanisme nu uitbreiden naar IPv6. (zie figuur 5.5) We laten een IPv6-host X6 verbinding maken met een IPv4-host met adres Y4. Deze laatste heeft echter een IPv6-adres nodig, waarbij de routering gebeurt langs de relayserver Z. Daarom hebben we een /64 - prefix nodig (C6::/64). De IPv6-host kan dan de IPv4-host bereiken langs het adres C6::Y4. De IPv6-host maakt nu een connectie met Y4; de relay doet zich voor als de bestemming. Op zijn beurt maakt deze dan een IPv4 connectie met Y4. Als de relayrouter verkeer terug krijgt, stuurt hij dat naar de IPv6-host, met als afzender C6::Y4.



Figuur 5.5: TRT: voorbeeld

De toepasbaarheid van dit systeem is behoorlijk groot. Alhoewel je wel een statische mapping (in bv. /etc/hosts) of een DNS-server nodig hebt om de nieuwe IPv6-adressen te creëren. Dit systeem is goed te gebruiken voor protocollen die het belangrijkste internetverkeer vormen: HTTP, SMTP, IMAP, SSH, POP3, FTP enz.

faithd

Onder FreeBSD is er een TRT implementatie *faithd*. We hebben een faith-interface waarnaar we het IPv6-prefix laten routen en een daemon per tcp/udp-poort. Om de faith-interface te hebben, dienen we in de kernel volgende regel te plaatsen.

```
pseudo-device   faith   1       # IPv6-to-IPv4 relaying (translation)
```

Vervolgens dienen we de variabele `net.inet6.ip6.keepfaith` op 1 te zetten met behulp van sysctl. We dienen ook een route te leggen voor onze prefix naar faith. Dat kan als volgt gebeuren:

```
root# route add -inet6 2001:6a8:1904:4:: -prefixlen 64 -interface faith0
```

Vervolgens kan men een relay service opstarten als volgt:

```
root# faithd ssh
```

Alle aanvragen voor ssh naar IPv4-knooppunten vanuit het IPv6-netwerk worden nu geroouteerd naar de faith-interface. Faithd luistert nu zelf op poort 22. Daardoor kan er wel geen lokale service meer draaien op die poort. Dit kan opgelost worden door de service op een andere poort te draaien of door faithd wijs te maken dat hij een lokale service moet opstarten als er een aanvraag komt voor het lokale ip. Dat kan als volgt:

```
root# faithd ssh /usr/sbin/sshd sshd -i
```

Voorts is het ook belangrijk dat men instelt wie allemaal gebruik mag maken van deze relay service. Want iedereen ter wereld heeft nu dezelfde mogelijkheden als de router die faithd draait voor de aangeschakelde services. Als er dus een faithd op smtp draait, heb je direct een open relay mailserver, enz. Het is dus belangrijk dat je enkel het lokale netwerk de relay laat gebruiken. Misschien is het zelfs beter om een sitelocal prefix te reserveren voor faithd.

Een probleem met faith is dat het niet-actieve connecties na een half uur verbreekt. Hoewel dit voor de meeste services geen struikelblok vormt, is het erg storend bij een service als ssh. De huidige implementatie voorziet geen mogelijkheid dit te configureren, dus dient men hiervoor de broncode te wijzigen. Dit kan in het bestand faithd.h waar we volgende definitie aanpassen.

```
#define FAITH_TIMEOUT   (30 * 60)
```

DNS

Het probleem met TRT is dat men telkens het IP-adres dient te berekenen. Men kan al die IP-adressen bijhouden in een gewone DNS-server, maar telkens er een site bijkomt, dient men te herberekenen. Daarom werden er enkele DNS-servers ontwikkeld die dienen om samen te werken met een TRT systeem. Bij een opzoeking wordt naar een host gekeken of er reeds een AAAA-record bestaat voor die naam, in het andere geval wordt het A-record vertaald naar een AAAA-record met behulp van het gereserveerde prefix. Hiervan bestaan 2 implementaties: newbie (`http://www.sfc.wide.ad.jp/~doi/softs/`) en totd (`ftp://ftp.pasta.cs.uit.no/pub/Vermicelli/totd-latest.tar.gz`).

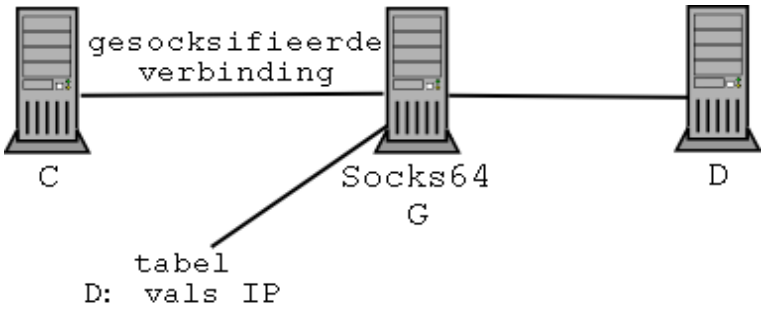
5.3.2 Socks64

Een andere methode om de overgang van IPv4 naar IPv6 te laten gebeuren, is het gebruik van een Socks64 proxyserver. Eerst zullen we uitleggen wat de taak is van een Socks proxy om vervolgens uit te breiden naar Socks64.

De Socks proxy is een service die standaard draait op poort 1080. Zo'n proxy staat meestal op een knooppunt tussen 2 netwerken, waarbij het ene netwerk niet rechtstreeks aan het andere kan. Via de Socks proxy kunnen dan TCP-aanvragen gedaan worden van het ene netwerk om een service te kunnen verkrijgen op het andere netwerk. Dat was het principe van Socks 4. Later is Socks 5 gekomen en konden ook UDP-aanvragen gedaan worden en was er authenticatie met Kerberos mogelijk.

Het nadeel van zo'n Socks proxy is dat de applicaties *gesocksifieerd* moeten zijn. Want eerst moet een communicatie opgezet worden tussen de client en de Socks proxy om af te spreken of ze al dan niet encryptie zullen gebruiken. Eens dat afgesproken is moet er een gepast antwoord op de vragen gegeven kunnen worden en moet dus de client die ook verstaan.

Hoe werkt nu zo'n Socks64 proxy? Als een IPv6-client een aanvraag wenst te doen voor een connectie naar een IPv4-knooppunt langs de proxy, dan zitten we ergens met een probleem van DNS. Socks helpt ons hierbij met een eigen API tussen de applicatie en de resolver.



Figuur 5.6: Socks64

Laten we nu het mechanisme in detail bespreken. (zie figuur 5.6) We hebben een client C, een router G en een bestemming D. De router is dual stack en draait de Socks64 server. Aan de zijde van de klant hebben we een Socks-bibliotheek tussen de applicatielaag en tussen de socketlaag. Deze Socks-bibliotheek kan bestaande typische aanvragen op een eigen manier implementeren (gethostbyname(), getaddrinfo(), enz.). Elke applicatie heeft zo zijn eigen Socks-bibliotheek of kan zich laten socksifiëren.

Op de router draait dus de Socks server die klanten en bestemmingen van eventueel verschillende internetprotocollen voorziet (bv. klant IPv4 en bestemming IPv6, klant IPv6 en bestemming IPv4, enz.).

De verbinding tussen klant en Socks server is gesocksifieerd en kan dus ook controleinformatie transporteren. De verbinding tussen de Socks server en de bestemming is een gewone verbinding. De bestemmeling heeft dus geen weet wie nu eigenlijk de zender is.

Daar we vooral geïnteresseerd zijn in de overgang van IPv4 naar IPv6, interesseren ons dus de connecties waarbij de zender en de bestemmeling verschillende internetprotocollen spreken. Stel nu dat de zender IPv4 spreekt en de bestemmeling IPv6.

De IPv4-applicatie heeft een bestemming nodig, maar het heeft geen plaats om daarin een IPv6-adres te plaatsen. Hetgeen gebeurt in Socks64 is dat het opzoeken van naam naar IP plaats vindt bij de server in plaats van bij de klant.

Het Socks5 protocol aanvaardt als bestemming de adrestypes IPv4, IPv6 of de logische bestemming (FQDN[1]). Indien in een IPv4-applicatie een IPv6-bestemming wordt meegegeven, wordt deze als adrestype FQDN doorgestuurd over de gesocksifieerde verbinding.

Om nu het probleem op te lossen, wordt er een vals IPv4-adres geïntroduceerd door de server en wordt het als bestemmingsadres gegeven aan de IPv4-klant. Daarom wordt er in de server een tabel bijgehouden met daarin welke valse IPv4-adressen overeenstemmen met welke FQDN.

We beschrijven nu in detail hoe een aanvraag verloopt als het bestemmingstype FQDN is. De applicatie van de klant probeert het IP-adres van de bestemming te achterhalen en roept de nameresolver aan. Hierbij wordt de FQDN van de bestemming doorgegeven als argument van de Socks-bibliotheek van de klant.

De Socks-bibliotheek heeft echter het resolven vervangen door zijn eigen implementatie en de FQDN informatie komt in een tabel. Een vals IP adres wordt teruggegeven aan de applicatie, dat tevens bij de FQDN in de tabel komt. Men gebruikt hiervoor een gereserveerde adresruimte die niet gebruikt wordt.

Het vals IP-adres wordt nu gebruikt bij het aanmaken van de internetsockets en applicatie-oproepen die door die socket gebeuren. Nu is het zo dat Socks ook die implementaties heeft vervangen door zijn eigen implementaties. Als de Socks server merkt dat er een vals IP-adres gebruikt wordt, zal hij zijn tabel raadplegen om de echte FQDN-informatie te achterhalen.

Vervolgens zet de Socks server een normale connectie op met de bestemming waarbij hij de standaard resolvers oproept.

De toepasbaarheid van het protocol is gelijklopend met de socksifieerbaarheid van de client. Vermits we een centrale Socks server hebben en elke client dient aangepast te worden aan Socks is deze methode niet echt ideaal te noemen. We hebben een *single point of failure* en een hoop configuratiewerk aan de clients. Anderzijds, indien het netwerk reeds uitgerust was met een Socks server omwille van veiligheidsredenen, loopt de overgang naar IPv6-connectiviteit vrij eenvoudig. Men dient enkel de Socks server dual stack te maken en een nieuwe gepatchte versie van Socks te installeren.

Implementatie

De Socks5 server kan bekomen worden bij nec op aanvraag via `http://www.socks.nec.com`. Men verkrijgt de broncode van de Socks server; vervolgens kan men die patchen met een patch die te vinden is bij het kame project: `ftp://ftp.kame.net/pub/kame/misc/`

[1]FQDN=Fully Qualified Domain Name; dat is de host en het domein, bv. www.google.com

`socks64-v10r10-20000322.tgz` Na het patchen dienen we in het configure script nog een controle weg te laten, want de software dateert nog uit de spreekwoordelijke prehistorie (meer dan 2 jaar oude software...). Na het compileren kan men de Socks server configureren als ware het een Socks5 server.

Bij Socks servers is het belangrijk dat men goed instelt wie deze allemaal mag gebruiken, want slecht geconfigureerde Socks servers worden in dank aangenomen door allerlei Internet-ongedierte of mensen die hun afkomst wensen te maskeren.

Zo kan men bijvoorbeeld een ping6 request door de proxy laten uitvoeren terwijl de client niet over IPv6 beschikt. In de logs van de proxyserver krijgen we dan:

```
Apr 25 21:15:20 abraham Socks5[14224]: PING Proxy Request: \
(157.193.28.16 to 3ffe:80b0:0:1:a00:20ff:fed1:f824) for user
Apr 25 21:15:20 abraham Socks5[14224]: PING Proxy Established: \
(157.193.28.16 to 3ffe:80b0:0:1:a00:20ff:fed1:f824) for user
Apr 25 21:15:22 abraham Socks5[14224]: PING Proxy Terminated: \
Normal (157.193.28.16 to 3ffe:80b0:0:1:a00:20ff:fed1:f824) for user :\
0 bytes out, 502 bytes in
```

Er wordt dus een aanvraag gedaan, er is geen authenticatie hier; de proxy zendt op zijn beurt de ICMPv6-pakketen en laat die via de Socks-verbinding terug weten aan de client.

5.3.3 Stateless IP/ICMP Translation Algorithm (SIIT)

SIIT is eveneens een methode die als doel heeft de overgang tussen IPv4 en IPv6 te verzachten. De werkwijze hier is dat de IP-headers vertaald worden van IPv4 naar IPv6. Ook ICMP-headers worden vertaald.

Bij deze methode wordt gebruik gemaakt van zogenaamd *IPv4-vertaalde IPv6-adressen*. Voor de IPv6-knooppunten worden namelijk tijdelijk IPv4-adressen toegewezen die gebruikt worden in de IPv4-vertaalde IPv6-adressen. Wanneer deze SIIT passeren, worden ze vertaald naar IPv4-adressen. Er wordt tevens gebruik gemaakt van IPv4-gemapte IPv6-adressen.

De IPv6-host die wenst te connecteren met een IPv4-host zal een IPv4-gemapt adres als bestemming zien en zal het IPv4-vertaalde adres als lokaal adres gebruiken. Wanneer de IPv4-host een pakket terugzendt, wordt de bestemming vertaald naar het IPv4-vertaalde adres. Men kan geen gebruik maken van IPv4-gemapte adressen omdat deze terug naar de SIIT-vertaler zouden gerouteerd worden. Tevens kan men geen gebruik maken van IPv4-compatibele adressen, omdat deze in IPv4 geëncapsuleerd zouden worden. Het is namelijk zo dat de SIIT-vertaler zich enkele hops van de IPv6-host kan bevinden en dat de IPv6-host op

een netwerk staat waarbij de router slechts IPv6 kent.

In hoeverre is dit nu toepasbaar? Enerzijds zijn er heel wat problemen bij de vertaling, wat gebeurt er met checksums, met ICMP-foutberichten, enz. Anderzijds is de vertaler *stateless*; er wordt geen status bijhouden, er dienen geen connecties bekeken te worden. Er kunnen eventueel meerdere SIIT-vertalers zijn. Er zijn ook problemen met multicast adressen en met IPsec; daarom wordt het vertalen hier en daar toch wat beperkt.

Waar rfc2765[18] ook niet echt over duidelijk is, is hoe die IPv4-vertaalde adressen toegekend moeten worden. Men gaat er van uit dat er ergens een voorraad IPv4-adressen is die gebruikt kunnen worden om die IPv4-vertaalde adressen te maken. Deze kunnen bijvoorbeeld met DHCP toegekend worden waarbij een erg korte lease geldt.

Vertalen van IPv4 naar IPv6

Wanneer de vertaler een pakket ontvangt bestemd voor een IPv6-eindpunt dan wordt de IPv4-header vervangen door een IPv6-header. De transport- en datalaag worden gelaten zoals ze waren bij de IP-pakketen, bij ICMP wordt er wel aangepast.

Een belangrijk verschil tussen IPv4 en IPv6 is de manier waarop met MTU wordt omgesprongen. Bij IPv6 is het verplicht om de MTU te bepalen, zodat fragmentatie nooit nodig zal zijn. Fragmentatie kan enkel bij de zendende host. Bij IPv4 is dit optioneel.

In het geval dat de IPv4-host de MTU berekent worden de pakketten goed verzonden en is er geen extra fragmentatie nodig. ICMP-*packet-too-big*-pakketten moeten bij die berekening dan wel vertaald worden. Dat de IPv4-host dit doet kan gedetecteerd worden als de DF-vlag gezet is.

Indien dit niet gebeurt, is er wel fragmentatie nodig. Pakketten worden gefragmenteerd in stukken van 1280 byte; IPv6 garandeert namelijk dat dergelijke stukken nooit gefragmenteerd moeten worden.

IP Een probleem duikt dus op bij gefragmenteerde IPv4-pakketten. We bespreken eerst ongefragmenteerde IPv4-pakketten. Dan wordt de header als volgt vertaald:

- Versie
  6
- Trafiekklasse
  In principe wordt het TOS-veld van IPv4 hierin gekopieerd; eventueel kan het veld ook op 0 gezet worden.

- Flow label
  0

- Payload length
  Dit is gelijk aan de lengte die opgegeven is in de IPv4-header min de lengte van de IPv4-header en eventuele opties. In tegenstelling tot IPv6 is in IPv4 de lengte van het totale pakket in de header opgenomen, in IPv6 is dat de lengte van de inhoud van het pakket.

- Next Header
  Is gelijk aan het protocol veld van de IPv4-header.

- Hop limit
  Dit is gelijk aan het TTL-veld van de IPv4-header; deze moet uiteraard gedecrementeerd worden en er moet controle gedaan worden om te zien of die niet nul is zoals bij een gewone router. Indien zo, dan dient een ICMP-*ttl-exceeded*-pakket verzonden te worden.

- Bronadres
  Het IPv4-gemapte adres van het IPv4-bronadres.

- Doeladres
  Het IPv4-vertaalde adres van het IPv4-doeladres.

Andere IPv4 opties worden genegeerd en niet vertaald, tenzij er gebruik gemaakt wordt van *source routes*[2]. Dan wordt een ICMPv4 *destination unreachable/source route* pakket teruggezonden.

Is er wel een *fragment header* dan wordt in de IPv6-header:

- Payload length
  De lengte van de IPv4 header + 8 (=lengte fragment header) - lengte van IPv4-header en opties.

- Next header
  Fragment Header (=protocol 44)

In de fragment header vervolgens:

- Next Header
  Het protocol veld uit de IPv4-header.

[2]Hierbij wordt aangegeven langs welke routers het IP-pakket moet verzonden worden.

- Fragment offset
  Is gelijk aan dezelfde uit de IPv4-header.
- M flag:
  Ook gelijk aan de *more fragments* bit uit de IPv4-header.
- Identification
  De hoogste 16 bits zijn gelijk aan 0, de laagste 16 bits worden gekopieerd uit het IPv4 identificatie veld.

ICMP In de header is de voornaamste wijziging de checksum; het veld wordt op een verschillende manier berekend bij ICMPv4 als bij ICMPv6. Het type dient vertaald te worden en eventueel ook de inhoud van het pakket.

Uit de ICMP-*query* berichten worden enkel *echo request* en *echo reply* vertaald. Types 8 en 0 in IPv4 worden respectievelijk types 128 en 129 in IPv6.

Alle ICMP foutberichten overlopen zou ons wat te ver leiden, de meeste foutberichten worden naar hun equivalenten vertaald, enkel type en/of code durven al eens te verschillen. Anderen zijn niet meer van toepassing of kunnen niet toegepast worden en worden genegeerd.

Ook de inhoud van de ICMP-berichten moet vertaald worden. Daardoor verandert echter de grootte van het volledige IP-pakket dat het ICMP-bericht draagt en moet dus van dat pakket de *payload length* aangepast worden.

Vertaling van IPv6 naar IPv4

Ook hier wordt de IPv6-header naar een IPv4-header vertaald en dan worden de pakketten verder via IPv4 doorgezonden. De inhoud van de pakketten wordt ongewijzigd gelaten tenzij het ICMP-pakketten zijn.

We dienen ook hier het probleem van de MTU aan te pakken. IPv6 verzekert namelijk dat de MTU minimaal gelijk is aan 1280 bytes, maar bij IPv4 is dat slechts 68 bytes. Dus wat gebeurt er als zo'n vertaald IPv6 pakket toekomt bij een IPv4 netwerk dat een kleinere MTU heeft? Dan treedt de IPv4 fragmentatie in werking, want die vereist niet dat fragmentatie enkel bij de zendende host optreedt.

IP

- Version
  4

- Header length
  5, er worden geen IPv4 opties meegegeven.
- TOS
  Traffiekklasse, of eventueel nullen.
- Totale lengte
  Grootte van de IPv4-header + lengte in de IPv6-header.
- Vlaggen
  More Fragments=0; Don't Fragments=1.
- Fragment offset
  0
- TTL
  Is gelijk aan de Hop limit uit de IPv6-header; ook hier decrementeren en controle op 0 uitvoeren.
- Protocol
  Next header.
- Header checksum
  Wordt berekend als de header gemaakt is.
- bronadres
  Het IPv4 adres wordt afgeleid uit het IPv4-vertaalde IPv6-adres; indien het IPv6-bronadres niet dat type is, wordt het bronadres op 0.0.0.0 gezet.
- doeladres
  Het IPv4-gemapte IPv6 adres wordt weer een gewoon IPv4-adres.

Optionele headers worden genegeerd en het *Total Length* veld wordt aangepast. Bij fragmentatie hebben we het volgende:

- Total Length
  Gelijk aan *Payload length* - 8 (Fragment Header) + grootte IPv4-header.
- Identification
  Is gelijk aan de 16 laagste bits van hetzelfde veld uit de IPv6-header.
- Flags
  More fragments wordt 1, *Don't Fragments* wordt 0.

- Fragment Offset
  Zelfde als in IPv6-header.
- Protocol
  Gelijk aan de *Next Header* waarde.

ICMP Bij ICMP gebeuren nu ook weer de omgekeerde vertalingen, enkel *Echo request* en *Echo reply* worden vertaald bij de informatieve berichten. De foutberichten worden naar hun gelijke types vertaald; als er geen equivalenten zijn, dan worden die genegeerd. De checksum dient ook hier weer aangepast te worden in de header.

Implementaties

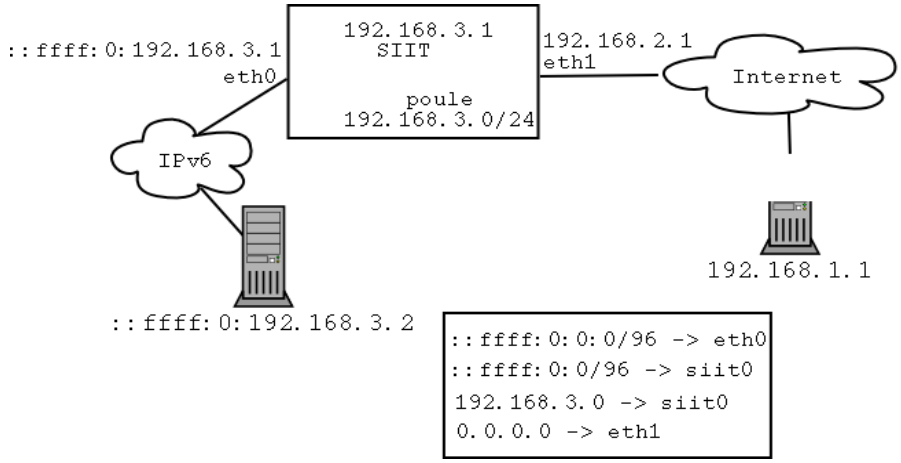
Op dit moment is er slechts 1 implementatie waar nog actief aan gewerkt wordt. Ze is terug te vinden op `http://www.ispras.ru/~ipv6/linux_siit.html`. Ze werkt enkel op een Linux 2.2.x USAGI-kernel. We dienen de SIIT-vertaler te zetten op een dual stack router; dus de variabelen `net/ipv4/conf/all/forwarding` en `net/ipv6/conf/all/forwarding` dienen gelijk aan 1 te zijn.

We laden de SIIT-module met behulp van insmod en we kennen een adres uit de pool van IPv4-adressen toe aan siit0. Beschouw figuur 5.7; hier hebben we het subnet 192.168.3.0/24 als pool genomen. Dan wordt dat:

```
[root@siitbak /root]# ifconfig siit0 192.168.3.1
[root@siitbak /root]# ifconfig siit0
siit0     Link encap:Ethernet  HWaddr 00:53:49:49:54:30
          inet addr:192.168.3.1  Bcast:192.168.3.255  Mask:255.255.255.0
          inet6 addr: fe80::253:49ff:fe49:5430/64 Scope:Link
          UP BROADCAST RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:172 errors:3 dropped:0 overruns:0 frame:0
          TX packets:169 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
```

We stellen dat de interface aan de IPv4-kant geconfigureerd is met het adres 192.168.2.1 en de interface aan de IPv6-kant geconfigureerd wordt met het IPv4-vertaalde IPv6-adres van siit0. Dat wordt dan het adres ::ffff:0:c0a8:301. Dat laatste gebeurt automatisch.

Vervolgens dienen we een route toe te voegen naar de IPv4-gemapte adressen, deze moeten via siit0 gaan. Dat kan als volgt:



Figuur 5.7: SIIT: opstelling

```
[root@siitbak /root]# route add -A inet6 ::ffff:0:0/96 dev siit0
[root@siitbak /root]# ip -6 route | grep ffff
::ffff:0:0:0/96 dev eth0  proto kernel  metric 256  mtu 1500 rtt 375ms
::ffff:0:0/96 dev siit0  metric 1  mtu 1500 rtt 375ms
[root@siitbak /root]#
```

Dus nu gaan alle IPv4-vertaalde IPv6-pakketten door eth0 naar het IPv6 netwerkje, IPv4 gemapte adressen gaan door de vertalingsinterface. Omgekeerd hebben we:

```
[root@siitbak /root]# ip route
192.168.3.0/24 dev siit0  proto kernel  scope link  src 192.168.3.1
192.168.2.0/24 dev eth1  proto kernel  scope link  src 192.168.2.1
127.0.0.0/8 dev lo  scope link
default via 192.168.2.254 dev eth1
[root@siitbak /root]#
```

Pakketten die dus voor onze pool aankomen, gaan door de vertalingsinterface, pakketten voor het gewone IPv4 netwerk gaan door eth1.

Als we eens tcpdumpen[3] op siit0 zien we mooi wat er gebeurt:

```
[root@siitbak /root]# tcpdump -i siit0
tcpdump: listening on siit0
```

[3]tcpdump is een programma dat IP-verkeer op een interface bekijkt en afdrukt.

```
09:31:26.895265 192.168.1.1 > 192.168.3.2: icmp: echo request
09:31:26.895356 ::ffff:c0a8:101 > ::ffff:0:c0a8:302: frag (0|64) \
icmp6: echo request
09:31:26.903688 ::ffff:0:c0a8:302 > ::ffff:c0a8:101: icmp6: echo reply
09:31:26.903712 192.168.3.2 > 192.168.1.1: icmp: echo reply
```

Er wordt een ping gedaan naar 192.168.3.2 vanuit een ipv4-host die 192.168.1.1 als heeft. De reply komt terug en wordt vertaald. 192.168.3.2 is echter een adres uit de poule van adressen die vertaald moet worden, dus die gaat door siit0. Dan komt het antwoord met als bestemming een IPv4-gemapt adres, dus die moet weer door siit0 en die vertaalt nu naar de andere kant.

5.3.4 Network Address Translation-Protocol Translation (NAT-PT)

NAT-PT is een transitiemechanisme dat bestaat uit een combinatie van NAT en protocolvertaling. NAT in dit document heeft veel verwantschap met de NAT in IPv4, met dat verschil dat er dan niet vertaald wordt tussen lokale en globale IP-adressen, maar tussen IPv4- en IPv6-adressen.

We onderscheiden traditionele NAT-PT en bidirectionele NAT-PT. In de eerste vertrekken de sessies steeds vanuit het IPv6-netwerk; in de laatste kunnen de sessies ook vanuit het IPv4-netwerk opgezet worden.

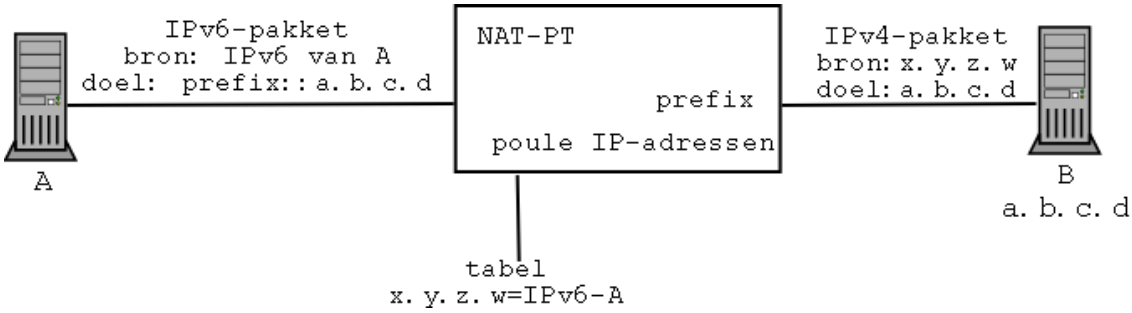
In de eerste onderscheiden we dan nog Basis-NAT-PT en NAPT-PT. Basis-NAT-PT vertaalt IP-adressen en gerelateerde velden in IP, TCP, UDP en ICMP; NAPT-PT vertaal ook nog poortnummers, wat de mogelijkheid geeft om meerdere IPv6-hosts te bedienen met één enkel IPv4-adres.

Basis-NAT-PT

Het mechanisme is vrij eenvoudig; er wordt een /96-prefix gereserveerd dat routeerbaar is in het IPv6-domein dat de NAT-PT-router moet voorbijkomen. Alle IPv6-verkeer wordt naar de NAT-PT-router gerouteerd en daar wordt het vertaald naar IPv4.

Beschouw figuur 5.8; de bron is A, de bestemming is B met IP a.b.c.d; het gereserveerde prefix is prefix. Stel nu dat IPv6-host A wil communiceren met IPv4-host B. Dan wordt het bronadres van A gelijk aan zijn IPv6-bronadres en zijn doeladres wordt gelijk aan prefix::a.b.c.d.

Als een connectie wordt opgezet, en het pakket komt voorbij de NAT-PT-router, dan wordt een IPv4-adres toegewezen aan de IPv6-host uit de poule van adressen waarover de router



Figuur 5.8: NAT-PT

beschikt. Gedurende de connectie houdt de NAT-PT-router de overeenkomst tussen het IPv4-adres en het IPv6-adres bij. Vervolgens zendt de router het pakket verder met als bronadres het adres uit de poule en als doeladres a.b.c.d. Terugkomende traffiek zal aanvaard worden als zijnde van dezelfde connectie en wordt dus in omgekeerde richting vertaald.

NAPT-PT

NAPT-PT staat voor Network Address Port Translation + Protocol Translation en laat toe dat meerdere IPv6-hosts gebruik kunnen maken van slechts één enkel IPv4-adres. Hiervoor dienen ook TCP en UDP poorten vertaald te worden. Het probleem met Basis-NAT-PT is dat als de poule van IPv4-adressen opgebruikt is, er geen nieuwe connecties meer kunnen bijkomen tot een host stopt met IPv4-connecties aan te gaan.

DNS

Tot zover was de gelijkenis met SIIT erg groot; in SIIT ontbrak echter een methode om adressen tijdelijk toe te kennen. Nu beschrijven we de interactie met de DNS-server bij het toekennen van tijdelijke IP-adressen. We beginnen eerst met inkomende DNS-aanvragen uit het IPv4-netwerk en vervolgens bespreken we de uitgaande connecties.

Adrestoekenning voor binnenkomende connecties Stel dat een IPv4-host B een DNS-aanvraag doet voor een IPv6-host A op het netwerk; de DNS-server voor dat domein bevindt zich ook op dat netwerk, dus die aanvraag moet via de NAT-PT router. De NAT-PT router moet dus de DNS-vraag en het DNS-antwoord aanpassen zonder dat de IPv4-host hier niets van merkt.

Hoe gebeurt dit nu precies? Bij naam-naar-adresvertaling, wordt het type van aanvraag

vertaald van een opzoeking naar een A-record naar een opzoeking naar een AAAA-record (of naar A6-records). Voor omgekeerde queries dient men IN-ADDR.ARPA te vervangen door IP6.INT en het omgekeerde IPv4-adres door het omgekeerde IPv6-adres. (als de overeenkomst reeds bestaat in NAT-PT-router) Indien dit laatste niet zo is, wordt er een adres genomen uit de poule.

Omgekeerd nu, wanneer het antwoord terugkomt, dan moeten AAAA- of A6-records in A-records vertaald worden. Het IPv6-adres wordt dan vervangen door het IPv4-adres dat opgeslagen is in de NAT-PT router.

Adrestoekenning voor uitgaande connecties Stel nu omgekeerd dat een IPv6-host A een DNS-opzoeking doet naar het IPv6-adres van IPv4-host B. NAT-PT zal eerst de AAAA- of A6-opzoeking doorsturen en er een A-opzoeking aan toevoegen. Indien er resultaat is voor de AAAA- of A6-opzoeking moet er geen vertaling gebeuren, in het andere geval zendt de NAT-PT router als antwoord dat het adres van B gelijk is aan PREFIX::a.b.c.d.

Protocol vertaling

Hoewel de protocolvertaling in grote mate overeenkomt met die van SIIT zijn er toch enkele details waarop we moeten letten. Bij de vertaling van versie 4 naar versie 6 wordt er gebruik gemaakt van PREFIX::IPv4-adres in plaats van IPv4-gemapte adressen. De bestemming is het echte adres en niet een IPv4-vertaald adres.

Bij vertaling van versie 6 naar versie 4 wordt het bronadres uit de tabel gehaald in NAT-PT en doeladres is de laagste 32 bit van PREFIX::IPv4-adres.

De UDP, TCP en ICMP checksums moeten ook herberekend worden.

Besluit

NAT-PT is een mooi systeem, dat even makkelijk inzetbaar als NAT op een IPv4-netwerk met private adressen; anderzijds heeft het dan ook wel de mankementjes (geen end-to-end connecties, problemen om bepaalde protocollen hierdoor te krijgen, enz.) hiervan. Door echter dit te gebruiken in plaats van gewone NAT, bekomt men dezelfde functionaliteit en is men klaar voor IPv6, want je hebt wel end-to-end IPv6-connecties. Men hoeft tevens niet meer voorzien in IPv4-connectiviteit.

Implementaties

Er zijn reeds verschillende implementaties beschikbaar van dit systeem. Dit verklaart uiteraard het kleine aantal SIIT implementaties. SIIT is eigenlijk een onafgewerkte NAT-PT transitie, maar legt anderzijds een goede basis voor NAT-PT.

Er is een implementatie van CISCO op `http://www.cisco.com`; nadat u een hoop zever en uw spamemailadres invulde op die site, kan men echter vaststellen dat deze niet meer ondersteund wordt en dat men aan een nieuwe versie werkt die gebaseerd is op de rfc in plaats van op de Internet draft. Verder is er ook nog een implementatie van Microsoft en een versie van Linux die enkel schijnt te werken op een Linux 2.4.0 beta systeem. Hier hebben we echter gekozen om eens de FreeBSD implementatie te bekijken, die ook enkel te vinden is in de KAME-snapshots en niet in de STABLE-kerneltree.

Men dient dus eerst de KAME-kernel en het KAME-basissysteem te installeren, wekelijkse snapshots zijn te vinden op `http://www.kame.net`. In de kernel compileert men volgende regel:

```
options         NATPT           # IPv6 -> IPv4 translation.
```

Men dient tevens sommige basiscommando's en headerbestanden te hercompileren en installeren; de instructies zijn te vinden bij de snapshot. Eens men dat gedaan heeft, heeft men een KAME-kernel met alle nieuwste en niet gegarandeerd stabiele IPv6-snufjes.

We dienen eerst een configuratiebestand te maken:

```
prefix 2001:6a8:1904:10::
map from any6 to 10.1.2.1
map enable
```

Hier is ons prefix gelijk aan 2001:6a8:1904:10::/96 en worden alle IPv6-adressen afgebeeld op het IPv4-adres 10.1.2.1. Vervolgens dienen we de vertaler expliciet aan te zetten met map enable. Hierbij werken enkel uitgaande connecties vanuit het IPv6-netwerk. We proberen nu een IPv4-host te bereiken, meer bepaald deze met IP 10.1.2.254. Dan moeten we pingen naar het IPv6-adres 2001:6a8:1904:10::a01:2fe of in leesbaardere vorm 2001:6a8:1904:10::10.1.2.254. Dit omzetten kan gebeuren met behulp van een aangepaste DNS-server zoals bijvoorbeeld newbie of totd. Als we dan even tcpdumpen op onze netwerkinterface krijgen we:

```
23:46:34.013077 2001:6a8:1904:2::1 > 2001:6a8:1904:10::a01:2fe: icmp6: echo request
23:46:34.013397 10.1.2.1 > 10.1.2.254: icmp: echo request
23:46:34.013880 10.1.2.254 > 10.1.2.1: icmp: echo reply
23:46:34.014042 2001:6a8:1904:10::a01:2fe > 2001:6a8:1904:2::1: icmp6: echo reply
```

Normaal zouden de IPv6-adressen en de IPv4-adressen op de interfaces van hun eigen kant moeten verschijnen; we gebruikten hier echter een router met slechts 1 interface.

5.3.5 Bump-In-the-Stack (BIS)

We hebben nu al verschillende vertalingsmechanismen gezien om IPv6 naar IPv4 te vertalen op het netwerk via routers op de grens van een IPv6- en een IPv4-domein. Maar wat vangen we aan met IPv4-applicaties die zich op een netwerk bevinden dat enkel IPv6 routeert? Hiervoor werd de BIS (Bump-In-the-Stack) techniek ontwikkeld.

Om dit te verwezenlijken heeft BIS enkele componenten nodig. Het heeft een vertaler nodig die IPv4-headers vertaalt in IPv6-headers en omgekeerd. Hoe dit kan, is beschreven in de SIIT-techniek. Voorts is er een DNS-resolver nodig; aanvragen naar AAAA-records moeten vertaald worden naar een aanvraag naar AAAA en A-records. Is er een AAAA-record, dan is er geen probleem; in het andere geval moet een IPv4-adres gebruikt worden. Hierbij komen we aan een derde nodige component; namelijk een systeem dat tijdelijk IPv4-adressen toekent (dit kunnen private adressen zijn) en die laat overeenkomen met een IPv6-adres. Dit heet de *address mapper*.

Werking bij het verzenden

De applicatie zoekt een A-record op voor de bestemming. De DNS-resolver onderschept de aanvraag en doet een aanvraag naar A- en AAAA-records aan de DNS-server. De bestemming is in dit geval IPv6; en de DNS-resolver ontvangt enkel een AAAA-record. Vervolgens doet de DNS-resolver een aanvraag aan de address-mapper om een IPv4-adres te laten overeenkomen met een gegeven IPv6-adres. De address-mapper antwoordt met een IPv4-adres, en de DNS-resolver maakt een A-record dat hij terugzendt naar de applicatie.

Vervolgens verzendt de applicatie een IPv4-pakket, dit pakket wordt opnieuw onderschept en komt terecht bij de vertaler. Nu doet de vertaler een beroep op de address mapper om een IPv6-adres te verkrijgen dat overeenkomt met het IPv4-adres; de address mapper vertaalt vervolgens en de vertaler verzendt het pakket. Als een IPv6-pakket terugkomt, wordt dat doorgegeven aan de vertaler en die geeft het vervolgens door aan de applicatie als IPv4-pakket.

Werking bij het ontvangen

Er wordt een pakket ontvangen van een IPv6-host; dit wordt doorgegeven aan de vertaler; de vertaler zoekt het corresponderende IPv4-adres op in de adress mapper en vertaalt vervolgens

het IPv6-pakket naar een IPv4-pakket, dat op zijn beurt doorgegeven wordt aan de applicatie.

5.3.6 Bump-In-the-API (BIA)

Dit mechanisme is gelijklopend met dat van BIS, enkel gebeurt de vertaling hier tussen de IPv4 en de IPv6 API's en niet in de stapels. Dit is een subtiel verschil met een gelijkaardig resultaat, maar de werking is toch wat anders.

De API-vertaler bestaat uit 3 belangrijke elementen; we hebben een name resolver, gelijkaardig aan die uit BIS. Ook de *address mapper* is dezelfde als die uit BIS. Hetgeen dan wel verschilt, is de vertaalfunctie. In plaats van de IP-headers te vertalen, vertalen we de API functies; dit gebeurt in het onderdeel *function mapper*.

Werking bij het verzenden

De IPv4-applicatie doet een DNS-opzoeking naar een A-record; hierbij wordt die opzoeking onderschept door de *name resolver*, die doet een opzoeking naar een A- en een AAAA-record voor die host. Indien er enkel een AAAA-antwoord is, vraagt de *name resolver* aan de *address mapper* voor een IPv4-adres dat overeenkomt met het verkregen IPv6-adres en maakt het een A-record aan dat teruggegeven wordt aan de applicatie.

De applicatie doet vervolgens een API-aanroep om een verbinding op te zetten; de *function mapper* onderschept die en doet een vertaling als die aanroep het resultaat is van een IPv4-applicatie. De *function mapper* vraagt nu het IPv6-adres aan de *address mapper* en voert de IPv6-API-functie uit.

Werking bij het ontvangen

Wanneer een IPv6 pakket binnenkomt, detecteert de *function mapper* dit; deze doet een opzoeking naar het corresponderende IPv4-adres bij de *address mapper* en voert een IPv4-API-functie uit voor de IPv4-applicatie.

Onderschepte functies

Volgende functies worden onderschept: bind(), connect(), sendmsg(), sendto(), accept(), recvfrom(), recvmsg(), getpeername(), getsockname(), getsocketopt(), setsocketopt(), recv(), send().

5.3.7 Dual Stack Transition Mechanism (DSTM)

Hierbij hebben we een IPv6-netwerk, met hosts die wel IPv4 spreken, maar het niet verder kunnen verzenden. Het idee hier is om een IPv4-connectie te verkrijgen via een IPv4 over IPv6 tunnel; en dit tunnelmechanisme te automatiseren.

IPv4 over IPv6 tunneling

IPv4 over IPv6 tunneling is de encapsulatie van IPv4-pakketten in IPv6-pakketten. In FreeBSD beschikken we hiervoor over de gif-interface. Laten we een tunnel leggen tussen 2001:6a8:1904:6:230:4fff:fe19:b4c4 en 2001:6a8:ff::1; de IPv4-eindpunten zijn hierbij respectievelijk 192.168.1.1 en 157.193.83.142. We leggen eerst de tunnel aan als volgt:

```
ifconfig gif0 create
gifconfig gif0 inet6 2001:6a8:1904:6:230:4fff:fe19:b4c4 2001:6a8:ff::1
```

Vervolgens benoemen we de eindpunten:

```
ifconfig gif0 192.168.1.1 157.193.83.142
```

We dienen ook de default route langs de tunnel te zenden:

```
route add default 157.193.83.142
```

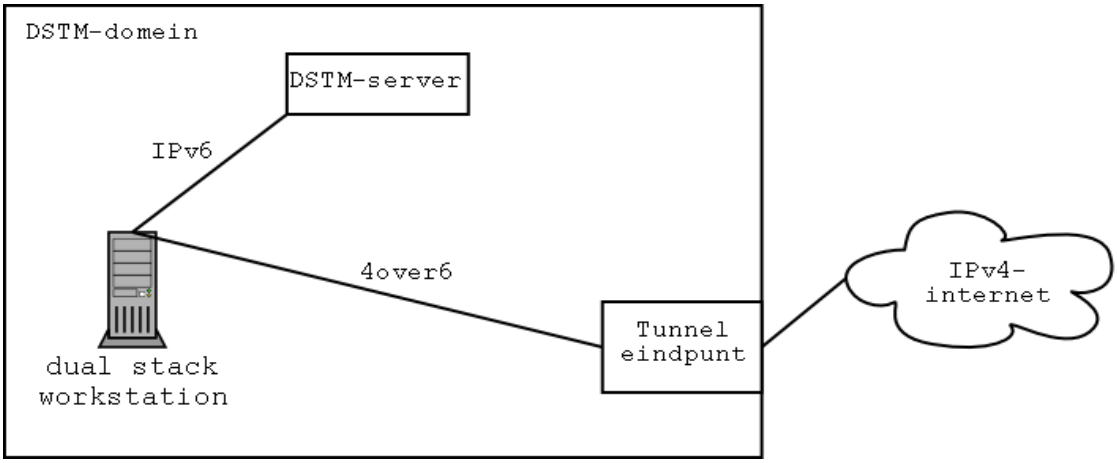
Onze interface is dan uiteindelijk configureerd:

```
gif0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1280
        tunnel inet6 2001:6a8:1904:6:230:4fff:fe19:b4c4 --> 2001:6a8:ff::1
        inet6 fe80::230:4fff:fe19:b4c4%gif0 prefixlen 64 scopeid 0x8
        inet 192.168.1.1 --> 157.193.83.142 netmask 0xffffff00
```

In dit geval moet het private adres (192.168.1.1) nog via NAT vertaald worden om het Internet op te kunnen. Er dient ook een gelijkaardige opstelling opgezet te worden aan de andere kant.

Werking van DSTM

We beschikken over een zeker domein waarop we DSTM wensen toe te passen; dat domein kan enkel IPv6 routeren en op de rand daarvan hebben we connectie met het IPv4-netwerk. In ons DSTM-domein hebben we een DSTM-server, knooppunten die als DSTM-client werken



Figuur 5.9: DSTM

en een dual stack router die als tunneleindpunt dient. Een typische opstelling is gegeven in 5.9.

Stel dat we als knooppunt in het DSTM-domein een IPv4-connectie wensen op te zetten met een host op het Internet. Het probleem is dat er geen IPv4-routing beschikbaar is. We doen een aanvraag aan de DSTM-server om een tijdelijk IPv4-adres te verkrijgen. De DSTM-server geeft ons tijdelijk een IP-adres en configureert tevens de dual stack router eindpunten. Aan de hand van het verkregen IP configureert de DSTM-client de 4 over 6 tunnel interface op de kant van de client. Op dit moment beschikt de DSTM-client over een IPv4-connectie. Daar dit een volwaardige connectie is, hebben we hier ook geen last van allerlei vertalingsmankementen.

5.3.8 6tunnel

6tunnel is een applicatie die luistert op een bepaalde poort voor IPv4; die aanvragen op die poort vertaalt naar IPv6 en deze doorstuurt naar een ander knooppunt op een bepaalde poort. Deze applicatie werd hieraan toegevoegd omdat ze niet echt thuishoort bij de besproken mechanismen, maar toch een handige en eenvoudige vertalingstoepassing is. Deze applicatie bestaat zowel voor FreeBSD, Linux als voor Windows.

De werking ervan is eenvoudig; stel dat we beschikken over een applicatie die enkel IPv4 spreekt en een server (bv. adam.rug6), die enkel via IPv6 bereikbaar is. We wensen deze server via ssh te bereiken, maar de ssh-client die we gebruiken, spreekt enkel IPv4.

```
6tunnel 2000 adam.rug6 22
ssh -p 2000 localhost
```

Dit zal ons in staat stellen adam te bereiken via ssh. We connecteren op poort 2000; 6tunnel vertaalt onze aanvraag naar IPv6 en zendt die door naar adam op poort 22; terugkerend verkeer wordt ook vertaald.

Hoofdstuk 6

Veiligheid

Een probleem dat zeker moet bekeken worden bij de uitbreiding van een bestaand netwerk naar IPv6 is de veiligheid. Met het creëren van toegang via IPv6, creëren we mogelijk nieuwe wegen voor ongewenste indringers in ons netwerk.

Veelal zijn netwerken reeds uitgerust met firewalls. Dit zijn apparaten die regelen welke pakketten ons netwerk binnen mogen en welke niet. Als die goed geconfigureerd zijn, houden die alles tegen en laten enkel het verkeer door dat we effectief wensen te hebben. Dus meestal zullen geëncapsuleerde IPv6-pakketten door een firewall tegengehouden worden. Als we die wensen toe te laten, dienen we IP-pakketten met als inhoud protocol 41 (=geëncapsuleerde IPv6-pakketten) toe te laten bij onze firewall. Deze pakketten kunnen we beter zelf ook nog eens controleren met een speciaal daarvoor aangepast programma. Dit kan met een IPv6-firewall die we later zullen bespreken.

Indien we denken geen firewall nodig te hebben, moet men in ieder geval goed uitkijken met vertalende apparaten. Een netwerk dat goed geconfigureerd is en geen interne diensten draait heeft bijvoorbeeld geen firewall nodig.

6.1 IPv6-firewalls

Laten we starten met IPv6-firewalls; onder FreeBSD hebben we IPV6FIREWALL. De andere firewall, ipfilter werkt nog niet met IPv6. Deze kunnen we gebruiken door volgende regel in de kernel te compileren:

```
options         IPV6FIREWALL            #firewall for IPv6
```

De configuratie van de firewall is gelijklopend met die van IPv4-firewalls en zullen we hier daarom niet bespreken; dit valt buiten het bestek van deze scriptie.

Onder Linux hebben we ip6tables; volgende opties dienen we in de kernel te compileren:

```
Networking options  --->
[*] Network packet filtering (replaces ipchains)
IPv6: Netfilter Configuration  --->
<*> IP6 tables support (required for filtering/masq/NAT) (NEW)
```

IPv6 ondersteuning dient uiteraard ook in de kernel te zitten.

6.2 Vertalingsmechanismen

6.2.1 Translating interfaces

TRT

Een transport relay translator kan mogelijk een groot veiligheidsgat creëren. Het is belangrijk dat we de toegang tot die server goed regelen. Stel dat we onze mailserver wensen toegankelijk te maken via IPv6 en dus poort 25 relayen. Als het relay adres routeerbaar is via het Internet hebben we een open relay mailserver, want voor de mailserver lijken de aanvragen steeds te komen van onze relayrouter en die worden dus aanvaard.

Toegang

Het is belangrijk dat we de toegang tot de vertalingsmechanismen strikt controleren; we moeten zorgen dat de vertalende routers enkel gebruikt kunnen worden door de doelgroep. Als ze dan al door derden zouden moeten gebruikt worden, dan is het om toegang te krijgen tot het netwerk waarvoor we vertalen. We moeten dus vermijden dat onze router gebruikt wordt om de afkomst van iemand te verdoezelen. Men kan bijvoorbeeld een slecht geconfiguurde Socks misbruiken om een aanval op een netwerk of host te doen. Hierbij zal dan de Socks proxy als schuldige aangewezen worden.

6.2.2 Neighbor advertisement

Indien personen die niet te vertrouwen zijn toegang krijgen tot het netwerk en ergens een IPv6-router opzetten kunnen zij interfaces laten configureren van gebruikers zonder dat deze

dit zelf weten of opmerken. Het is dus aangewezen om IPv6 af te zetten als je het niet gebruikt. Via ping6 ff02::1 (=een ICMPv6 echo verzoek naar alle knooppunten op het netwerk) door de interface van het netwerk kan men de hosts vinden die voor IPv6 geconfigureerd zijn. Als deze router advertisement pakketten aanvaarden, zijn ze direct bereikbaar van op het volledige Internet en zijn dus mogelijk slecht beveiligde diensten vatbaar voor een externe aanval.

6.3 6to4

Als we 6to4 configureren en het eindpunt als router instellen, dan kan onmiddellijk iedereen ter wereld die gebruiken als default router. Eventueel kan een firewall dit verhinderen; we zien anders echter geen mogelijkheid om dit misbruik tegen te gaan.

Hoofdstuk 7

Stand van zaken

Na deze bespreking van IPv6 en de transitiemethodes is de vraag wanneer men IPv6 zal implementeren in bedrijfsnetwerken of in hoe verre dit reeds gebeurd is. Hiervoor hebben we enkele Belgische providers aangeschreven die reeds een teken van leven tonen in het gebruik van IPv6. We hebben hen hierbij naar hun huidige implementatie gepolst; naar hun plannen naar de toekomst toe en hoe zij de evolutie van IPv6 zagen.

Als een provider al bezig is met IPv6 dan is dat in testfase en wordt het dikwijls niet officieel aan de klanten aangeboden. Laten we eerst even bekijken welke allocaties de providers reeds verkregen hebben van ripe of van de 6bone. Dit kan makkelijk achterhaald worden met whois.[1]

Belnet	3ffe:608:2::/48	Surfnet
	3ffe:80b0::/28	6bone
	2001:6a8::/35	ripe
Wanadoo/Euronet	3FFE:2501:200::/48	NLnet
	3FFE:80C0:221::/48	Stealth.net
	3FFE:8100::/28	6bone
Chello	3FFE:2501:100::/48	NLnet
	3FFE:80B0:1002::/48	Belnet
	2001:730:2::/48	Chello NL
Skynet	tunnel	IPng.nl

Wat wel blijkt is dat het probleem niet is wat men al eens durft aan te halen: Er is geen vraag, dus wij bieden het niet aan, dus er is geen vraag. Men doet dus wel degelijk inspanningen aan de kant van de providers. Het grote probleem is de moeilijkheid om het te implementeren. Cisco routers zijn nog niet klaar voor IPv6 (enkel patches van bedenkelijke kwaliteit), Linux

[1] whois is een tool waarbij je kunt opvragen aan wie een domein of een IP-adres toebehoort; daarbij wordt heel wat informatie gegeven; o.m. het gereserveerde blok voor die provider.

systemen hebben geen productiecode, om maar te zwijgen over de huidige implementaties van Microsoft. Redback servers (servers die voor ADSL worden gebruikt) kennen al helemaal geen IPv6. Ook DHCP, die tot de belangrijkste connectiemethodes behoort op dit moment, is nauwelijks gestandaardiseerd. Een ander probleem is ook dat men nog voornamelijk met IPv6 in IPv4 tunnels moet werken en die blijken een stuk minder stabiel te zijn dan gewone lijnen.

Bij Easynet en Wanadoo kunnen klanten IPv6-adressen en hosting krijgen van de provider; bij Easynet in Duitsland is men begonnen bij hun hostingsites ook reeds standaard AAAA-records toe te kennen. Wanadoo doet ook een tunnelbrokerservice waarbij eender wie een tunnel kan verkrijgen. Chello heeft deels dual stack routers en is zijn netwerk met IPv6 aan het uitrusten. Er wordt vooral nog getest, maar geen diensten aangeboden. Belnet heeft heel wat services dual stack draaien, een status is te vinden op `http://www.ipv6.belnet.be`; ze hadden een tunneldienst, maar die werkt nu niet meer en enkele IPv6-diensten werken ook niet echt super.

Hoofdstuk 8

Slotwoord

De overgansperiode van IPv4 naar IPv6 zal een lange tijd in beslag nemen; de vraag is natuurlijk hoe lang? Op dit moment ziet het er niet direct naar uit dat er een stroomversnelling komt in de overgangsperiode. Iets wat men een jaar of 2 terug wel dacht en op dat moment leek de ontwikkeling wat actiever dan nu. Eens het tekort aan IPv4-adressen groter zal worden, zal er misschien wel meer gekozen worden voor IPv6. Het grote probleem is echter dat de apparatuur en applicaties vaak nog niet klaar zijn voor IPv6. Als dat er is, denken we dat er wel snel vooruitgang zal zijn. Bedrijven zullen kiezen om ook via IPv6 toegankelijk te zijn omdat dat modern staat en een goed imago geeft.

We mogen echter ook niet vergeten dat er ook NAT bestaat in IPv4 en dat dit zeker niet in het voordeel speelt van IPv6. Er zal echter wel een moment komen dat IPv6 duidelijk interessanter wordt dan NAT (met NAT-PT kunnen we al iets gelijkaardigs verkrijgen). We denken dat NAT-PT of NAT met alles dual stack achter NAT op dit moment de interessantste oplossingen zijn. Bij NAT-PT heb je geen IPv4-administratie meer nodig; het nadeel is dat alle gebruikte applicaties IPv6 moeten ondersteunen of men heeft extra transitiemechanismes nodig. Bij NAT gecombineerd met dual stack heb je reeds de voordelen van IPv6 en kan je gewoon verder met IPv4 als voorheen.

Bijlage A

FreeBSD voor linuxgebruikers

A.1 Partionering

De installatie van FreeBSD verloopt gelijkaardig als de installatie van gelijk welk opensource-os. Dus je kunt installeren via cdrom of netwerk, enz. Het grootste verschil is de manier van partitioneren. In FreeBSD geeft men een andere betekenis aan het woord partitie. Een schijf is namelijk opgedeeld in slices (wat we elders partities noemen). FreeBSD moet zich laten installeren in een primaire slice. Eens er een slice gealloceerd is voor FreeBSD, kan die gepartioneerd worden. Daarmee wordt bedoeld dat de slice intern opgedeeld wordt in een eigen partioneringssysteem. Men dient ook een swappartitie te voorzien.

A.2 Devicenamen

Een ander groot verschil met Linux is dat de devices een andere naam hebben. Atapi harde schijven noemen /dev/ad*, scsi-schijven noemen /dev/da*. Atapi-cdroms noemen /dev/acd*, scsi-cdroms noemen /dev/cd*. Een concreet voorbeeld zal veel duidelijk maken. Bijvoorbeeld de eerste partitie in de eerste slice van de eerste schijf op de eerste controller is /dev/ad0s1a. Hierbij duidt ad op het feit dat we met een atapi disk te doen hebben, 0 op de positie van de schijf, s1 op eerste slice en a op de eerste partitie in de slice. Stel dat we een extended slice hebben die een ext2fs bevat, dan wordt die aangeduid door: /dev/ad0s5. Networkdevices hebben ook een andere naam dan in Linux, hun naam refereert naar de driver. Een NE-2000 kaartje is van het type ed. De eerste NE2000 kaart is dan ed0, de tweede ed1, enz... Een via-rhine kaart is dan vr0,... Hun namen kunnen gevonden worden in de kernelconfig (zie verder).

A.3 Kernelconfig

Net zoals in Linux kun je ook in FreeBSD je kernel compileren naar je behoeftes. Er is niet zo'n mooie gebruikersinterface zoals make menuconfig of xconfig als in Linux, maar je moet een bestand maken met alle opties die je nodig hebt. Nu staan er standaard op je systeem al het GENERIC en het LINT kernelconfiguratiebestand. Het GENERIC-kernelconfiguratiebestand bevat alle opties die meekomen met de standaard FreeBSD-kernel, dus die vormt meestal een goed vertrekpunt en van daaruit kun je dan dingen toevoegen of verwijderen. Het LINT-kernelconfiguratiebestand bevat alle mogelijke opties die je in je kernel kunt voegen en de nodige commentaar over wat welke optie doet. Deze is dus als referentie te gebruiken. Een kernel compileren onder FreeBSD gaat merkelijk sneller dan onder Linux, de eerste keer duurt het even lang, maar daarna eens je bijna alles gecompileerd hebt en enkel nog hier en daar kleine aanpassingen doet heb je binnen de kortste keren een nieuwe kernel gelinkt. De FreeBSD kernel komt ook niet op het net zoals de linuxkernel, om de 4 maand wordt gewoon een nieuwe versie van FreeBSD uitgebracht en die bevat dan de huidige kernelsource. Tussenin kun je echter steeds via cvsup (zie verder) de laatste kernelsources binnenhalen.

Hoe gaat men nu exact te werk. Je installeert de kernelsources[1], dan ga je naar de directory /usr/src/sys/i386/conf (vervang i386 door de gewenste architectuur). Daar zul je de bestanden LINT en GENERIC zien staan. Je kopieert het bestand GENERIC naar een ander bestand, een goed idee is je hostname in hoofdletters. En dan kun je je kernel tunen, veel zal kunnen verwijderd worden. Standaard zijn zowat alle netwerkkaarten ondersteund en die maken je kernel wat groter dan nodig is, voorts is het misschien interessant om IPNAT en/of IPFILTER in je kernel mee te compileren als je een router wenst op te zetten. Eens je denkt dat je configuratiebestand in orde is, voer je dit uit: `config MIJNCONFIGFILE`. Deze zal controleren of er geen fouten in configuratiebestand staan, als dat gebeurd is, dan ga je naar de directory ../../compile/MIJNCONFIGFILE en voer je `make depend && make && make install` uit. Die laatste zet je nieuwe kernel in /kernel en je oude kernel wordt /kernel.old. Deze beschermt ook die bestanden zodat die niet per ongeluk kunnen verwijderd worden, want zonder kernel is booten redelijk onmogelijk geworden. Voor FreeBSD moet je derhalve dan ook geen lilo-achtige constructies gaan doen, standaard wordt /kernel geladen, tenzij je bij het opstarten iets anders wenst op te starten, je krijgt hiervoor de mogelijkheid dit in te tikken.

[1] Dit kan aan de hand van /stand/sysinstall; dit is een tool waarmee je zowat alles kunt configureren en installeren in een menustructuur.

A.4 Werken met de ports en packages

Om programma's te installeren zijn er 2 mogelijkheden, ofwel gebruik je de ports ofwel de packages. Ik heb zelf al vastgesteld dat die 2 niet echt goed met elkaar overeenkomen en dat je die beter niet met elkaar vermengt. Er zijn echter wel voldoende tools om de anomaliën op te lossen. Packages zijn binaries die in tgz-formaat van een FreeBSD-mirror kunnen gedownload worden, met behulp van *pkg_add -r package* of in /stand/sysinstall kun je packages binnenhalen en installeren. Je hoeft je nooit zorgen te maken waar je je packages moet downloaden, FreeBSD regelt dit allemaal voor jou. Ook over afhankelijkheden hoef je je nooit zorgen te maken, die worden allemaal opgelost door FreeBSD, als een package afhankelijk is van een ander, dan wordt dat ander ook binnengehaald en geïnstalleerd. Dat is wel goed, maar enige omzichtigheid is geboden. Ik wou Evolution (een mailclient) testen; het gevolg was dat FreeBSD als afhankelijkheden Ximian (een volledige grafische omgeving) begon te installeren.

Naast de packages bestaan er nog de ports. De ports op zich is gewoon een directory /usr/ports, met daarin voor elk mogelijk programma een subdirectory. Er is natuurlijk een hiërarchische structuur: de portsdir is dus ingedeeld in categorieën en dan in programma's per categorie. In elke programma-directory staat een korte beschrijving van het programma, staan de patches die op de source moeten losgelaten worden, een Makefile en een lijst van bestanden die de port zal installeren. In de Makefile staat dan waar FreeBSD de sources vandaan moet halen. Je kunt dikwijls ook opties aan de Makefile toevoegen zodanig dat je programma dan gecompileerd wordt naar jouw behoeften. Voor vim kun je bijvoorbeeld meegeven `make "NO\_GUI=YES"` en dan wordt de grafische versie van vim (en dus ook de afhankelijkheden daarvan) niet geïnstalleerd. De ports bevatten doorgaans ook nog recentere versies dan de packages.

A.5 Sources up to date houden met cvs

A.5.1 cvsup

Om je sources up to date houden bestaat er cvsup. Deze tool wordt vooral gebruikt om de ports en de kernelsources up to date te houden. Uiteraard wordt cvsup ook gebruikt in grote projecten met vele medewerkers die zo steeds de laatste stand van zaken kunnen uppen en downen naar de cvs-server, maar dit zou ons hier te ver leiden. Hier willen we enkel dat ons systeem steeds de laatste sources kan hebben. Hiervoor moeten we eerst een supfile aanmaken die zegt welke sources wij allemaal willen updaten. Een voorbeeld van een supfile:

```
*default host=ftp2.fr.freebsd.org
*default base=/usr
```

```
*default prefix=/usr
*default release=cvs tag=RELENG_4
*default delete use-rel-suffix
*default compress
src-all
ports-all tag=.
doc-all
```

Hierbij worden de kernelsources naar de STABLE versie van de 4.x gebracht, er bestaat ook een FreeBSD CURRENT, dit is de toekomstige 5.x en is dus nog ontwikkelingscode. De ports en de docs worden naar de huidige versie geupdated. We connecteren met een ftpserver in Frankrijk.

A.5.2 anoncvs

Een andere manier om de sources up to date te hebben is anoncvs; deze methode kan ook via IPv6 in tegenstelling tot cvsup. In ssh-config zetten we:

```
Host anoncvs.no.netbsd.org
Port 11750
```

Voorts dienen we deze omgevingsvariabele te zetten

```
CVSROOT=:ext:anoncvs@anoncvs.no.netbsd.org:/cvsfreebsd
```

Met `cvs checkout ports` haal je de portstree binnen, analoog voor source en doc.

A.6 rc.conf

FreeBSD heeft één configuratiebestand waar je zowat alles kunt instellen wat dient te gebeuren bij het opstarten. Hierin zegt men welke diensten allemaal moeten gestart worden en hoe het netwerk geconfigureerd moet worden. Je kan in /etc/defaults/rc.conf al het mogelijke vinden dat in /etc/rc.conf kan. De rc-scripts die meegeleverd zijn, gebruiken dan de inhoud van rc.conf om alles goed te zetten. Heb je toch nog iets specifiek dan kun je dat nog altijd in rc.local zetten.

Bijlage B

Case: Een IPv6-netwerk op RUGnet

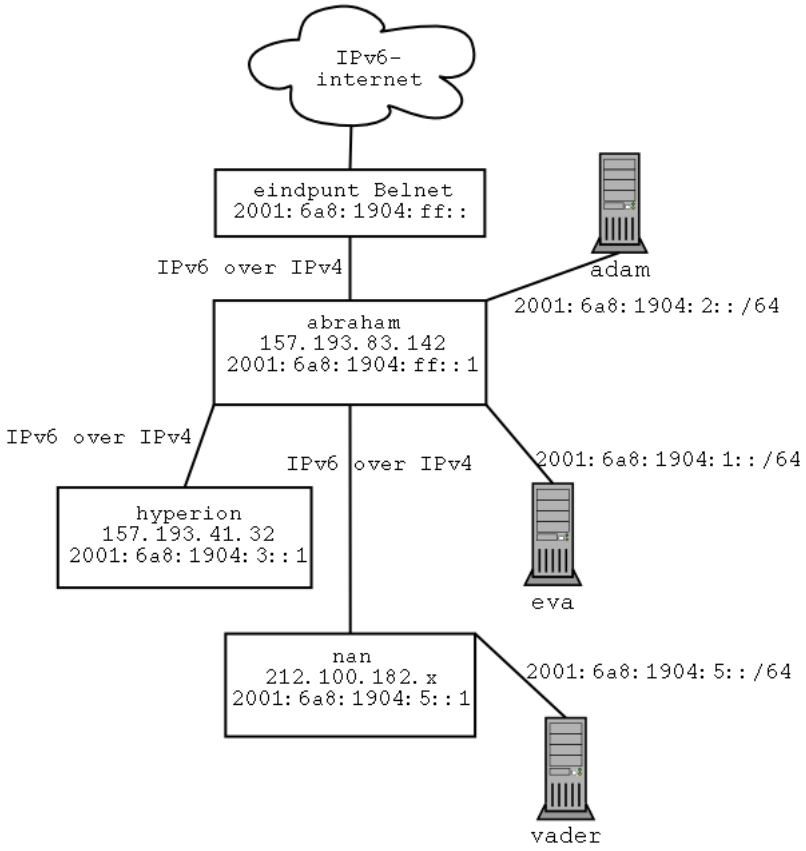
In deze bijlage wens ik de opstelling te bespreken die ik opgezet heb in het kader van mijn scriptie. Ik ga overlopen wat de structuur is van het IPv6-netwerk, welke vertaaltechnieken ik heb gebruikt en hoe ik de vpnclient van Cisco wist te omzeilen.

Het volledige netwerk in 1 keer neerpoten zou iets te veel van het goede zijn; een stuk ervan is te vinden in figuur B.1. Ik zal beginnen met de belangrijkste router te bespreken. Dat is abraham.elis.rug.ac.be met IPv4-adres 157.193.83.142. Deze is gestart met een FreeBSD 4.4-RELEASE besturingssysteem en vervolgens meerdere malen upgedated tot nu een FreeBSD 4.5-STABLE. Ze bevat 3 interfaces; 1 interface op het elisnet en 2 interfaces voor lokale netwerken.

Op dit moment interesseert ons de externe interface. Daar het elisnet geen IPv6 spreekt, moest abraham dus als tunneleindpunt ingesteld worden. Hiervoor dienden enkele aanpassingen te gebeuren aan de elis firewall opdat geëncapsuleerde IPv6-pakketten het netwerk zouden binnenkunnen. Deze pakketten mochten echter niet van om het even waar komen, maar slechts van die knooppunten of netwerken die we vertrouwden.

In eerste instantie was er IPv6-connectiviteit met Freenet6 via een IPv6 in IPv4 tunnel aan de hand van het tspc-protocol[1]. Hiervoor diende de firewall 206.123.31.114 (tsps1.freenet6.net) toe te laten voor protocol 41. Ook was er connectiviteit via een 6to4-router aan de hand van de 6to4 methode. Ook hiervoor werd de firewall aangepast. Deze 2 providers lagen echter nogal ver van het elisnet (pingtijden van meer dan 200ms).

[1]TSPC is een protocol dat automatisch een IPv6tunnel configureert met behulp van een eigen server en client



Figuur B.1: IPv6 op rugnet(1)

Geruime tijd na mijn aanvraag wist Belnet toch nog te reageren en kreeg ik een tunnel waarbij het andere eindpunt bij de spreekwoordelijke deur ligt. Laten we vanaf hier dan nog enkel de situatie beschouwen waarbij de default IPv6-router bij Belnet ligt. Er werd een statische IPv6 in IPv4 tunnel opgetrokken tussen abraham (157.193.83.142) en Belnet (193.190.197.234). De IPv6 eindpunten zijn hierbij respectievelijk 2001:6a8:ff::1 en 2001:6a8:ff::. Van Belnet kreeg ik het prefix 2001:6a8:1904::/48; dit is een SLA-delegatie van Belnet die de delegatie 2001:6a8::/35 kreeg van RIPE.

Aan abraham hangen 2 subnetten met op die 2 subnetten elk 1 host namelijk adam en eva. Daar vrouwen altijd eerst gaan werden de subnetten met eva en adam respectievelijk benoemd met de prefixen 2001:6a8:1904:1::/64 en 2001:6a8:1904:2::/64. De subnetten spreken ook nog IPv4; deze kregen respectievelijk de private adressen 10.1.1.0/24 en 10.1.2.0/24 met netmask 255.255.255.0. Als er iets zou misgaan met de ene adresbenoeming, dan kunnen we nog steeds op de andere terugvallen. In dit geval was dit echter geen garantie want FreeBSD heeft blijkbaar last met vaststellen welk type media aangesloten is op de D-link kaartjes. Daarom

moest expliciet het media type meegegeven worden bij de configuratie.

Adam draait een FreeBSD 4.4-KAME kernel en basissysteem; eva draait een FreeBSD 4.4-RELEASE systeem. Adam en eva zijn voor IPv6 geconfigureerd met router advertisement; voor IPv4 kregen ze statische private IP-adressen.

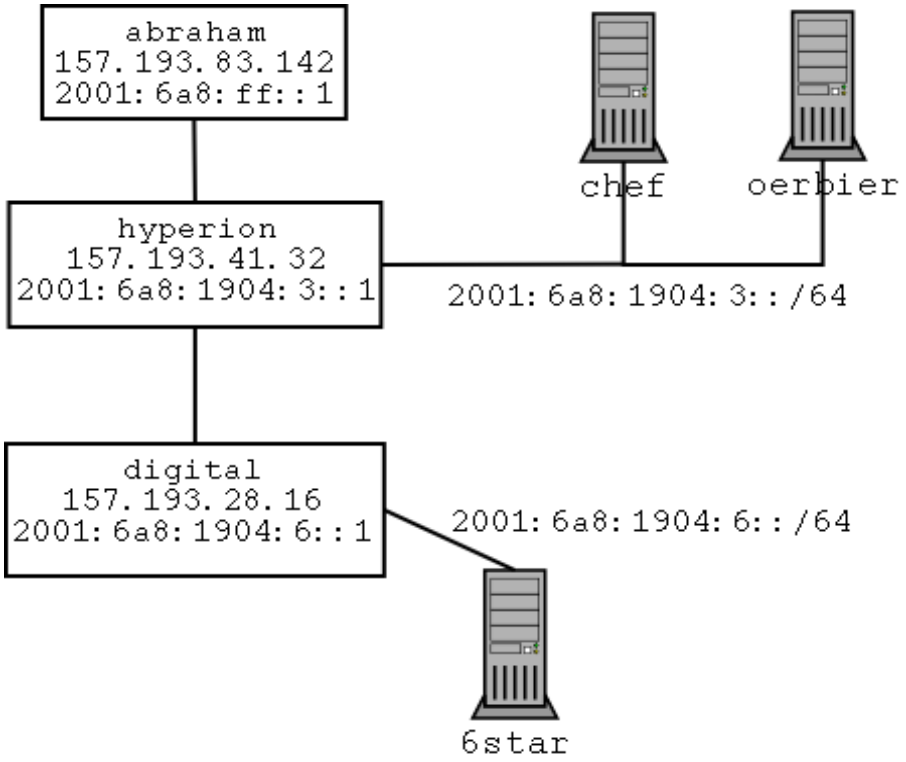
Voorst vertrekken uit abraham nog 2 IPv6 in IPv4 tunnels. Een tunnel naar hyperion (157.193.41.32) over rugnet en nog een tunnel met een variabel eindpunt, maar dat eindpunt ligt steeds in het netwerk 212.100.182.0/24. Deze laatste is een Easynet ADSL go account en heeft als hostname nan met een variabel IPv4-adres.

Laten we even de tunnel naar nan beschouwen. nan is een multiboot machine met de besturingssystemen Linux debian, FreeBSD 4.5-STABLE en Windows XP. nan is ingesteld als dual stack router en daarachter staat nog een Linux debian unstable; dat is het netwerk 2001:6a8:1904:5::/64. De IPv6-eindpunten van deze tunnel zijn onbenoemd. Deze linuxmachine krijgt zijn IP via router advertisement.

Voorts hebben we een tunnel naar hyperion. Beschouw figuur B.2. Hyperion is een dual stack router met 2 interfaces. Aan de ene kant hangt het interne netwerk van zeus; deze heeft het prefix 2001:6a8:1904:3::/64. Het zeusnetwerk verkrijgt ook IPv6-adressen via router advertisement. Netwerk 2001:6a8:1904:4::/64 wordt gerouteerd naar een vertalende interface op hyperion.

Vanuit hyperion vertrekt er dan nog een IPv6 in IPv4 tunnel naar digital; digital is een router die via vpn connecteert op RUGnet en heeft een pseudovast-IP-adres. In die zin dat het ding zijn IP behoudt zolang de VPN-router niet plat gaat. Inderdaad, mijn machines zijn stabieler dan die dingen van het ARC... Bij het schrijven van deze tekst hebben we al 3 weken hetzelfde IP-adres en dat zullen we dan ook maar gebruiken, nl. 157.193.28.16. De vpnclient die door Cisco geleverd wordt, heeft echter de eigenschap dat ze standaard het lokale netwerk uitschakelt. Uiteraard kan men in de instellingen instellen dat de vpnclient dat niet mag doen, maar Cisco heeft het blijkbaar niet zo op zijn instellingen begrepen. Na wat testen stel ik vast dat echter enkel het IPv4-verkeer niet werkt op het netwerk achter digital, IPv6 werkt wel. Dit was een gelukkig toeval, want ik zat nu in een situatie waarbij mijn workstation 6star die achter digital staat enkel via IPv6 het netwerk op kon. Het netwerk tussen digital en 6star kreeg het prefix 2001:6a8:1904:6::/64, ook hier werd gebruik gemaakt van router advertisement. Deze keer met de Linux client omdat digital een Linux debian testing is met kernel 2.4.13 en het workstation 6star is een FreeBSD 4.5-STABLE. Op deze laatste draait er nog een vmware met daarin een Linux uitgerust met USAGI 2.2.20 kernel. Deze was niet verbonden met Internet en diende enkel om de Linux SIIT implementatie uit te testen. Tot zover dus de netwerkopstelling.

Alle machines zijn uitgerust met een sshd die zowel op IPv4 als op IPv6 luistert. Abraham



Figuur B.2: IPv6 op rugnet(2)

is uitgerust met een webserver (apache+ipv6-patch 1.3.22) en draait een nameserver bind9 (versie 9.1.3). Aan de webserver is php gekoppeld; met daarop een php script dat de tunnel naar Easynet kan aanpassen. De nameserver is voor het domein abraham.elis.rug.ac.be. In de nameserver van elis is daarom het volgende record opgenomen:

```
abraham          IN      A       157.193.83.142
                         NS      abraham.elis.rug.ac.be.
```

De versie werd gemaskeerd in het named.conf configuratiebestand; de 3 belangrijkste zones zijn:

```
zone "abraham.elis.rug.ac.be" {
        type master;
        file "db.abraham.elis.rug.ac.be";
};

zone "rug6" {
        type master;
        file "db.rug6";
```

```
};

zone "4.0.9.1.8.a.6.0.1.0.0.2.IP6.INT" {
        type master;
        file "db.2001.6a8.1904";
};
```

Hierbij zijn rug6 en abraham.elis.rug.ac.be dezelfde domeinen; rug6 is om wat minder tikwerk te hebben. Voorts is er een ook een zone voor de adres-naar-naam-opzoekingen voor het prefix 2001:6a8:1904::/48. Hiervoor is er echter geen pointer van bij Belnet en werken die omgekeerde opzoekingen enkel als abraham.elis.rug.ac.be als nameserver gebruikt wordt (al dan niet rechtstreeks).

Hier is het zonebestand voor abraham.elis.rug.ac.be:

```
kristof@abraham:/usr/local/etc/namedb9$ cat db.abraham.elis.rug.ac.be
$TTL 86400
@               IN      SOA     abraham.elis.rug.ac.be. kverhenn.elis.rug.ac.be. (
                                        2001122500 ; Serial
                                        28800 ; Refresh every 8 hours
                                        7200 ; Retry every 2 hours
                                        604800 ; Expire after a week
                                        86400) ; Minimum ttl of 1 day
                IN      NS      abraham.elis.rug.ac.be.
                IN      NS      ns4.abraham.elis.rug.ac.be.
                IN      NS      ns6.abraham.elis.rug.ac.be.
@               IN      A       157.193.83.142
                IN      AAAA    2001:6a8:ff::1
                IN      AAAA    2001:6a8:1904:1::1
                IN      AAAA    2001:6a8:1904:2::1
ns6             IN      AAAA    2001:6a8:ff::1
ns4             IN      A       157.193.83.142
ipv6            IN      AAAA    2001:6a8:ff::1
eva             IN      AAAA    2001:6a8:1904:1:280:c8ff:fe54:ed6c
adam            IN      AAAA    2001:6a8:1904:2:280:c8ff:fe54:fcc6
oerbier         IN      AAAA    2001:6a8:1904:3:250:fcff:fe44:8d4
hyperion        IN      AAAA    2001:6a8:1904:3::1
chef            IN      AAAA    2001:6a8:1904:3:250:fcff:fe44:8e3
zeus            IN      AAAA    2001:6a8:1904:4::0a01:01fe
nan             IN      AAAA    2001:6a8:1904:5::1
vader           IN      AAAA    2001:6a8:1904:5:210:5aff:feda:367b
digital         IN      AAAA    2001:6a8:1904:6::1
6star           IN      AAAA    2001:6a8:1904:6:230:4fff:fe19:b4c4
```

Het bestand voor de omgekeerde opzoekingen:

```
$TTL    3600
@               IN      SOA     abraham.elis.rug.ac.be. kverhenn.elis.rug.ac.be. (
                                2001122500 ; serial
                                8H ; refresh
                                2H ; retry
                                1W ; expire
                                1D) ; Minimum TTL
                IN      NS      abraham.elis.rug.ac.be.

1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.0 IN  PTR  abraham.elis.rug.ac.be.
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.0.0 IN  PTR  abraham.elis.rug.ac.be.
c.6.d.e.4.5.e.f.f.f.8.c.0.8.2.0.1.0.0.0 IN  PTR  eva.abraham.elis.rug.ac.be.
6.c.c.f.4.5.e.f.f.f.8.c.0.8.2.0.2.0.0.0 IN  PTR  adam.abraham.elis.rug.ac.be.
4.d.8.0.4.4.e.f.f.f.c.f.0.5.2.0.3.0.0.0 IN  PTR  oerbier.abraham.elis.rug.ac.be.
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.3.0.0.0 IN  PTR  hyperion.abraham.elis.rug.ac.be.
3.e.8.0.4.4.e.f.f.f.c.f.0.5.2.0.3.0.0.0 IN  PTR  chef.abraham.elis.rug.ac.be.
e.f.1.0.1.0.a.0.0.0.0.0.0.0.0.0.0.4.0.0.0 IN  PTR  zeus.abraham.elis.rug.ac.be.
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.5.0.0.0 IN  PTR  nan.abraham.elis.rug.ac.be.
b.7.6.3.a.d.e.f.f.f.a.5.0.1.2.0.5.0.0.0 IN  PTR  vader.abraham.elis.rug.ac.be.
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.6.0.0.0 IN  PTR  digital.abraham.elis.rug.ac.be.
4.c.4.b.9.1.e.f.f.f.f.4.0.3.2.0.6.0.0.0 IN  PTR  6star.abraham.elis.rug.ac.be.
```

In principe is dat iets vreselijks om in te tikken, maar de omgekeerde zones heb ik gegenereerd met ip6_int.

Voorts heb ik mij bij kreynet (irc-netwerk) als IPv6-gebruiker laten registreren en kan ik via IPv6 op ipv6.krey.net via het adres 2001:6a8:ff::1. Een client als irssi ondersteunt IPv6. Op abraham draait ook nog rtadvd, dat is een programmatje dat router advertisement pakketten uitzendt voor de 2 netwerken die eraan hangen.

Hyperion is een andere belangrijke router in het netwerk, hierop draait ook een router advertisement daemon om het zeusnetwerk van IPv6 te voorzien. Voorts is hyperion uitgerust met een WWWoffle proxyserver. Via deze server kan men op een IPv4-machine aanvragen doen naar IPv6-sites. Deze http-proxy is zowel via IPv4 als via IPv6 bereikbaar en kan aanvragen doen voor zowel IPv4 als IPv6. Daar hyperion achter de firewall van het arc staat moeten aanvragen voor IPv4 sites die niet op rugnet gelegen zijn verder aangevraagd worden aan een andere proxy (bv. proxywww.rug.ac.be of wina.rug.ac.be).

Hyperion is ook uitgerust met faithd, de vertalende interface voor het TRT-mechanisme. Deze werd in het bijzonder gebruikt om pc's op het zeusnet die niet rechtstreeks van buiten rugnet beschikbaar zijn en die enkel IPv4 spreken toch via IPv6 beschikbaar te maken. Nu

dienen we bij TRT toch kenbaar te maken welke services we wensen te delen. De belangrijkste waren ssh, http en imap over ssl; dus poorten 22, 80 een 993. Het gevolg is hier dat alles dat achter de firewall staat van het arc en normaal enkel van binnen rugnet bereikbaar is nu ook bereikbaar is van buiten rugnet voor die poorten.

Nu met een faithd alleen heeft men steeds een hoop rekenwerk om het IP-adres te berekenen, daarom draait op hyperion ook nog de trick-or-treat daemon (totd). Dit is een nameserver die de gerelayde adressen berekent. Bijvoorbeeld voor zeus met faithd en totd op krijgen we:

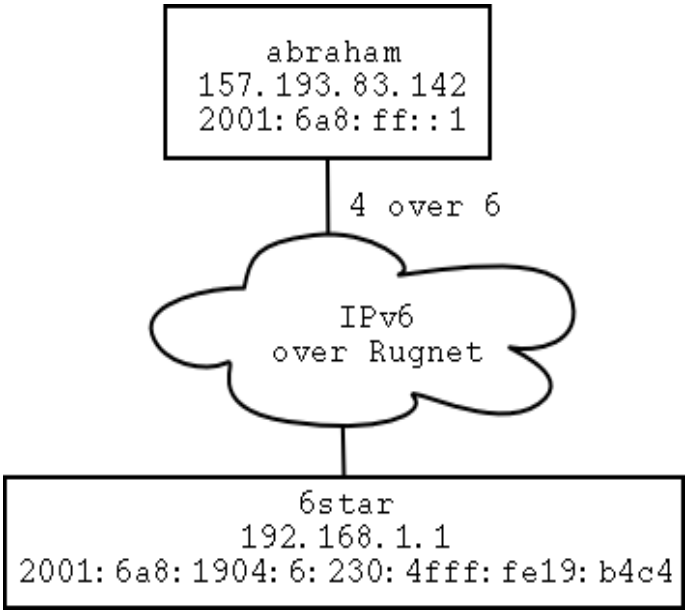
```
bash-2.05a$ telnet zeus.rug.ac.be 22
Trying 2001:6a8:1904:4::9dc1:2926...
Connected to zeus.rug.ac.be.
Escape character is '^]'.
SSH-1.5-OpenSSH-1.2.3
```

Voorts hebben we nog digital en 6star. Op ons IPv6-workstation proberen we zoveel mogelijk IPv6-programma's te draaien. Sommige programma's werken nu echter eenmaal niet met IPv6. Om te browsen en te ftp'en, kunnen we gebruik maken van onze proxyserver. Ssh werkt allemaal reeds via IPv6, onze mail kunnen we binnenhalen via imap over ssl langs de vertaler hyperion. Onze IPv4-programma's zijn te paaien via een 6tunnel constructie. Zij connecteren lokaal via IPv4 en 6tunnel connecteert via IPv6. Via de faith interface kunnen we zelfs zo twee IPv4-programma's met elkaar laten spreken die elkaar enkel via IPv6 zouden kunnen bereiken. Nu hebben we wel nog het probleem dat we geen mail kunnen verzenden, want we wensen geen open relay te creëren door faithd 25 te draaien; of we moesten controleren wie de faithd allemaal mag gebruiken. Om daarvoor een oplossing te vinden, heb ik IPv4 over IPv6 laten transporteren.

Er werd een IPv4 over IPv6 tunnel gelegd tussen abraham en 6star. IPv4 over IPv6 is op dit moment enkel ondersteund in BSD en hiervoor diende ook de kernel opnieuwe compileerd te worden. Het probleem is dat er dubbele encapsulatie gebeurde op abraham en FreeBSD laat standaard slechts enkele encapsulatie toe. Door meerdere encapsulaties toe te laten loopt men het risico in een encapsulatie-loop terecht te komen wat zou leiden tot een kernel crash.

6star heeft een lokaal IP-adres gekregen en abraham werd als IPv4-NAT-router ingesteld. Op 6star hadden we volgende tunnel, waarbij de default IPv4-route naar 157.193.83.142 liep.

```
gif0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1280
        tunnel inet6 2001:6a8:1904:6:230:4fff:fe19:b4c4 --> 2001:6a8:ff::1
        inet6 fe80::230:4fff:fe19:b4c4%gif0 prefixlen 64 scopeid 0x8
        inet 192.168.1.1 --> 157.193.83.142 netmask 0xffffff00
```



Figuur B.3: 4 over 6

Op abraham hadden we de volgende tunnel met een route naar 192.168.1.1 door die tunnel.

```
gif3: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1280
        tunnel inet6 2001:6a8:ff::1 --> 2001:6a8:1904:6:230:4fff:fe19:b4c4
        inet6 fe80::280:c8ff:fe54:fcd0%gif3 prefixlen 64 scopeid 0xd
        inet 157.193.83.142 --> 192.168.1.1 netmask 0xffff0000
```

De vraag hierbij is nu of er een mogelijkheid bestaat voor derden om langs deze weg ongevraagde toegang te verkrijgen tot het elisnet. Daar we niet geëncrypteerd werken is dit mogelijk. Men plaatst zich ergens op rugnet waar de IPv6 over IPv4 tunnel passeert en men verzendt valse pakketten. 6star is echter niet te misbruiken omdat die geen IPv4 routeert.

Daar we niet vertaalden voor poort 25 en het elisnet ook niet in gevaar willen brengen, hebben we nog steeds geen afdoende oplossing om mail te verzenden. Gelukkig is er nog iets als een ssh-tunnel. Hierbij forwarden we een lokale poort naar een mailserver op poort 25. Dat kan bijvoorbeeld als volgt:

```
ssh -L 2525:eduserv.rug.ac.be:25 kristof@digital
```

En dan verzenden we mails op localhost poort 2525. Als we dit laatste als root doen kunnen we ook localhost poort 25 nemen. Maar als root op het netwerk gaan doet men dus niet...

Uiteindelijk is dit op zijn minst gezegd omslachtig om netwerkdiensten te verkrijgen, er werd een veelheid van trucjes en technieken toegepast. Een eenvoudige manier wordt wat belemmerd doordat digital een Linux-systeem is en eenvoudige overgangsmethodes daarop dus nauwelijks geïmplementeerd zijn. Een nat-pt implementatie of een 4 over 6 tunnel naar digital zou de ideale situatie zijn.

Bijlage C

Lijst van afkortingen

API	Application Program Interface
BIA	Bump-In-the-API
BIS	Bump-In-the-Stack
CGI	Common Gateway Interface
CIDR	Classless Inter-Domain Routing
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DSTM	Dual Stack Transition Mechanism
FQDN	Fully-Qualified Domain Name
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IMAP	Internet Message Access Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
MTU	Maximum Transfer Unit
NAT	Network Address Translation
NAT-PT	Network Address Translation-Protocol Translation
NAPT-PT	Network Address Protocol Translation-Protocol Translation
NLA	Next Level Aggregator
OSPF	Open Shortest Path First
POP3	Post Office Protocol 3
RIP	Routing Information Protocol
SIIT	Stateless IP/ICMP Translation Algorithm
SLA	Site Level Aggregator
SSH	Secure Shell
TCP	Transmission Control Protocol
TLA	Top Level Aggregator
TRT	Transport Relay Translator
TSCP	Tunnel Setup Protocol Client
UDP	User Datagram Protocol

Bibliografie

[1] Piet de Meester, "Computer Networks", 2001.

[2] Andrew S. Tanenbaum, "Computernetwerken", 1997.

[3] Evi Nemeth, Garth Snyder, Scott Seebass en Trent H. Hein, "Unix System Administration Handbook", 2001.

[4] Silvano Gai, "Interworking IPv6 with Cisco Routers", 1998.

[5] S. Deering en R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 1883, December 1995.

[6] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas en L. Jones, "SOCKS Protocol V5", RFC 1928, April 1996.

[7] R. Callon en D. Haskin, "Routing Aspects Of IPv6 Transition", RFC 2185, September 1997.

[8] R. Hinden, "IP Version 6 Addressing Architecture", RFC 2373, Juli 1998.

[9] R. Hinden, M. O'Dell en S. Deering, "An IPv6 Aggregatable Global Unicast Address Format", RFC 2374, Juli 1998.

[10] R. Hinden, "Proposed TLA and NLA Assignment Rules", RFC 2450, December 1998.

[11] S. Deering en R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[12] T. Narten, E. Nordmark en W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)", RFC 2461, December 1998.

[13] S. Thomson en T. Narten, "IPv6 Stateless Address Autoconfiguration", RFC 2462, December 1998.

[14] A. Conta en S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 2463, December 1998.

[15] G. Armitage, P. Schulter, M. Jork en G. Harter, "IPv6 over Non-Broadcast Multiple Access (NBMA) networks", RFC 2491, Januari 1999.

[16] B. Carpenter en C. Jung, "Transmission of IPv6 over IPv4 Domains without Explicit Tunnels", RFC 2529, Maart 1999.

[17] D. Borman, S. Deering en R. Hinden, "IPv6 Jumbograms", RFC 2675, Augustus 1999.

[18] E. Nordmark, "Stateless IP/ICMP Translator (SIIT)", RFC 2765, Februari 2000.

[19] G. Tsirtsis en P. Srisuresh, "Network Address Translation - Protocol Translation (NAT-PT)", RFC 2766, Februari 2000.

[20] K. Tsuchiya, H. Higuchi en Y. Atarashi, "Dual Stack Hosts using the "Bump-In-the-Stack"Technique (BIS)", RFC 2767, Februari 2000.

[21] R. Rockell en R. Fink, "6Bone Backbone Routing Guidelines", RFC 2772, Februari 2000.

[22] M. Crawford en C. Huitema, "DNS Extensions to Support IPv6 Address Aggregation and Renumbering", RFC 2874, Juli 2000.

[23] R. Gilligan en E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers", RFC 2893, Augustus 2000.

[24] B. Fink, "6BONE pTLA and pNLA Formats (pTLA)", RFC 2921, September 2000.

[25] R. Hinden, S. Deering, R. Fink en T. Hain, "Initial IPv6 Sub-TLA ID Assignments", RFC 2928, September 2000.

[26] A. Durand, P. Fasano, I. Guardini en D. Lento, "IPv6 Tunnel Broker", RFC 3053, Januari 2001

[27] B. Carpenter en K. Moore ,"Connection of IPv6 Domains via IPv4 Clouds", RFC 3056, Februari 2001

[28] C. Huitema, "An Anycast Prefix for 6to4 Relay Routers", RFC 3068, Juni 2001.

[29] H. Kitamura, "A SOCKS-based IPv6/IPv4 Gateway Mechanism", RFC 3089, April 2001.

[30] J. Hagino en K. Yamamoto, "An IPv6-to-IPv4 Transport Relay Translator", RFC 3142, Juni 2001.

[31] Internet Draft: An overview of the introduction of IPv6 in the Internet

[32] Internet Draft: 6to4 and DNS

[33] Internet Draft: Dual Stack Hosts using "Bump-in-the-API"(BIA)

[34] Internet Draft: Dynamic Host Configuration Protocol for IPv6 (DHCPv6)

[35] Peter Bieringer, "Linux IPv6 Howto"

[36] `http://www.ipv6.wanadoo.be/`

[37] `http://www.freenet6.net/`

[38] `http://www.xs26.net/`

[39] `http://www.iana.org`

[40] `http://www.microsoft.com/WindowsXP/pro/techinfo/administration/ipv6/ipv6ipv4.asp`

[41] `http://hs247.com/`

Index