modelvragen computernetwerken II, reeks C

Belangrijke algemene opmerking: alhoewel niet expliciet gevraagd, word je bij de beantwoording van de **SNMP** vragen geacht om de globale structuur en de meest typische knooppunten en objecten van volgende componenten te kennen:

- de *ip groep van MIB-II* (§4.3.1.4 en voorbeelden in §4.2.4)
- de *host resources MIB* (§4.3.3 en voorbeelden in §4.2.4)

C1.

SNMP protocol

a. Geef de ASN.1 codering van een [...] SNMP bericht, en een korte uitleg bij elke TLV.
b. Bepaal de inhoud van de (hexadecimale) dump van een [...] bericht, dat tegelijkertijd [...], [...] en [...] vraagt/meldt.

C2.

SNMP protocol

a. Geef de ASN.1 codering van een (*niet-TrapResponse*) SNMP bericht, en een korte uitleg bij elke TLV.
b. Interpreteer de inhoud van volgende (hexadecimale) dump [...] van een SNMP bericht. Geef ondermeer aan om welk soort bericht het gaat.
c. Indien het om een *response* bericht zou gaan, construeer dan de meest efficiënte oplossing voor de bijhorende *request*. Indien het om één of andere vorm van een *request* bericht zou gaan, construeer dan de bijhorende *response*.

De structuur van SNMP protocolberichten wordt vastgelegd met dezelfde ASN.1 (Abstract Syntax Notation 1) syntax als waarmee de MIB modules en objecten gedefinieerd worden. De Basic Encoding Rules (BER) van X.209 vertalen de tekstuele representatie in de gecodeerde vorm die gebruikt moet worden voor effectieve uitwisseling. De gecodeerde vorm van een SNMP bericht bestaat uit een verzameling *Tag-Length-Value* (TLV) tripletten. Deze TLV's worden door de BER van de NMSs (Network Management Stations) en de agenten vertaald in een vorm die zijn individueel begrijpen. SNMP protocolberichten hebben geen vaste formaat: het *value* veld van een triplet kan een scalaire, enkelvoudige waarde bevatten, of recursief samengesteld zijn uit een structuur of een lijst van een aantal andere TLV's. Het *length* veld bevat uiteraard de lengte van het waardeveld. Het *tag* veld bestaat uit 1 enkel byte, die verwijst naar het datatype van het *value* veld. Het NULL datatype werd ingevoerd om TLVs toe te laten zonder expliciet ingevuld *value* veld. Met elke lijn in de ASN.1 code komt een TLV triplet overeen.

SNMP biedt een verzameling eenvoudige operaties aan die interactie tussen NMS en agenten mogelijk maken, nl. de get request, de set request en de trap.

De get request of poll vormt de meest uitgevoerde basisoperatie, waardoor een NMS een agent kan vragen om specifieke informatie waarover die beschikt. Een NMS kan bv. aan de agent van een router de toestand van elk van zijn interfaces opvragen, en daarna op basis van die informatie een of andere actie ondernemen indien één van de interfaces down zou blijken

te zijn. Het periodiek opvragen van informatie, door het NMS aan de diverse agenten van het netwerk wordt *externe polling* genoemd. Indien de software van de agent periodiek de toestand van een component ondervraagt dan wordt dat *interne polling* genoemd.

De set request biedt de mogelijkheid om via de agent de toestand van bepaalde instellingen van de netwerkcomponent te wijzigen.

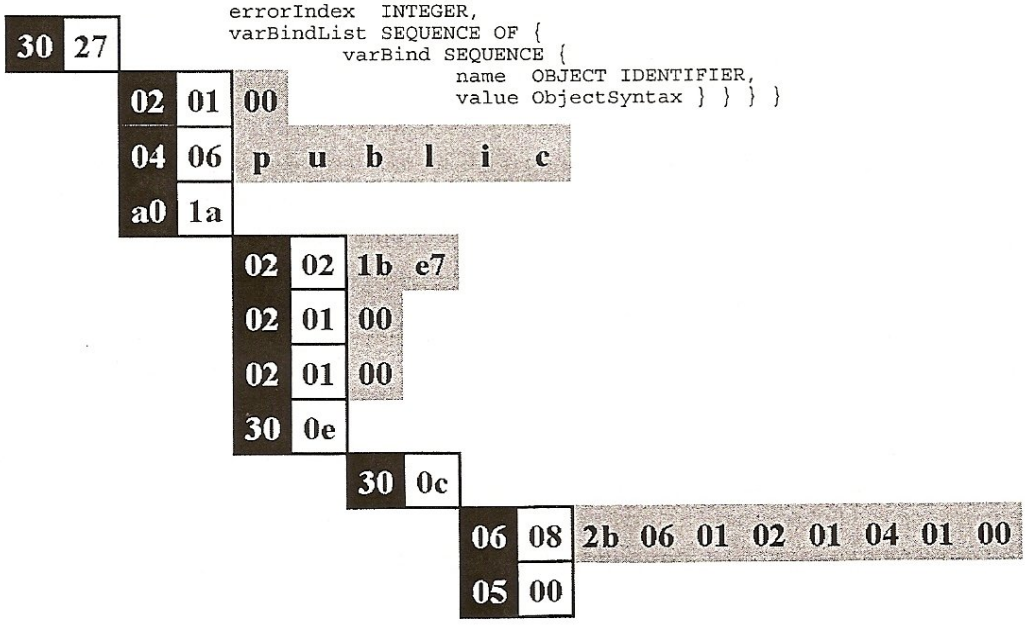
Een trap is de manier waarop een agent op eigen initiatief het NMS kan verwittigen dat er iets aan de hand is. *Traps* worden asynchroon opgestuurd, niet als reactie op ene vraag van het NMS.

C1 – C2

Het formaat van een ***GetRequest***, *GetResponse* en *SetRequest* bericht worden gedefinieerd door volgende ASN.1 code:

Een NMS neemt het initiatief om een *GetRequest* te sturen naar een agent. Het formaat van een *GetRequest* bericht wordt gedefinieerd door volgende ASN.1 code:

```
GetRequestMessage ::= SEQUENCE {
        version    INTEGER,
        community OCTET STRING,
        protocolDataUnit SEQUENCE {
                requestID    INTEGER,
                errorStatus INTEGER (0-5),
                errorIndex   INTEGER,
                varBindList SEQUENCE OF {
                        varBind SEQUENCE {
                                name   OBJECT IDENTIFIER,
                                value ObjectSyntax } } } }
```



Figuur 4.26

Het *GetRequest* bericht bestaat uit 7 scalaire TLVs:

- De *version* TLV bevat voor SNMPv1, SNMPv2c en SNMPv3, respectievelijk de waarden 0,1 en 3.
- De *community* TLV moet ingevuld worden met het niet-geëncrypteerde wachtwoord.
- De waarde in het *requestID* TLV wordt incrementeel of random gegenereerd, en is bedoeld om de SNMP antwoorden met de juiste vraag te correleren, en om eventueel gedupliceerde berichten te elimineren.
- De *errorStatus* TLV bevat in een *GetRequest* steeds de waarde 0, wat erop wijst dat geen fout is opgetreden bij het verlaten van het NMS.

- De *errorIndex* TLV wordt enkel geïnterpreteerd indien de *errorStatus* niet nul is, en toont in dit geval op welk object in de *varBindList* de fout betrekking heeft.
- De *name* TLV bevat de OID van het object. Elke comonent van een OID wordt door één byte gecodeerd, behalve de eerste twee(overeenstemmend met iso.org), die samen in 1 enkel byte gecodeerd worden. Indien de waarde van een component >= 128 is, dan wordt de component in de 7 laagste bits van 2 bytes gecodeerd. De 1ste byte krijgt hierbij een hoogste bit 1, als indicator van een afwijking. Volgende figuur toont ter illustratie de dump van de name TLV bij opvragen van het object met OID 1.3.6.1.4.311.1.3.6 (Enterprises.microsoft.software.dhcp.dhcpPar.parDhcpTotalNoOfAcks)

- De waarde van het object wordt in het TLV opgestuurd. Aangezien in *GetRequest* berichten de waarde van het object nooit gekend is, kan hier steeds een TLV met *tag* veld 05, wijzend op het NULL datatype, gebruikt worden.
- De overige TLVs structureren het bericht: zo vormt de *varBind* TLV koppels van name en *value* TLVs, terwijl aantal *varBind* TLVs gegroepeerd worden tot een *varBindList* TLV. Op deze manier kan 1 enkel *GetRequest* meerdere objecten tegelijkertijd opvragen.

Het ***GetResponse*** bericht heeft exact dezelfde structuur. Uiteraard wordt de inhoud van het *protocalDataUnit* en *value* TLVs aangepast. De agent probeert zo goed mogelijk te antwoorden. Indien de waarde van 1 van de aangevraagde objecten niet kan opgezocht worden, wordt de reden van de fout opgeslagen in de *errorIndex* en *errorStatus* TLVs. Het *value* veld van dit laatste kan in SNMPv1 volgende waarden bevatten:

noError(0) | *vraag kan beantwoord worden*
tooBig(1) | *antwoord te groot om in UDP verpakt te worden*
noSuckName(2) | *1 van de gevraagde OIDs bestaat niet*
genErr(5) | *alle overige fouten*

SNMPv1 foutboodschappen laten bijgevolg niet veel precisering toe. SNMPv2c definieert dan ook 13 aanvullende foutwaarden.

Het ***SetRequest*** bericht kan zowel gebruikt worden om de waarde van objecten te wijzigen, als om een nieuwe rij toe te voegen aan een tabelobject. De structuur van een *SetRequest* bericht is identiek aan de structuur van de *GetRequest* en *GetResponse* berichten. De agent gaat na of de aanvraag kan en mag uitgevoerd worden, en stuurt een *GetResponse* bericht terug met *errorStatus* en error Index TLVs. Volgende waarden in het *value* veld van de *errorStatus* TLV kunnen hier aanvullend optreden:

bad*Value*(3) | *waarde van object niet consistent met datatype*
readOnly(4) | *vraag tot aanpassing van een read-only object*

De structuur van een ***TrapResponse*** bericht is in SNMPv1 verschillend van de structuur van de andere berichten:

```
TrapResponseMessage ::= SEQUENCE {
        version   INTEGER (0),
        community OCTET STRING,
        protocolDataUnit SEQUENCE {
                enterpriseID OBJECT IDENTIFIER,
                agentAddress IpAddress,
                genericTrap  INTEGER,
                specificTrap INTEGER,
                timeStamp    TimeTicks,
                varBindList  SEQUENCE OF {
                        varBind SEQUENCE {
                                name  OBJECT IDENTIFIER,
                                value ObjectSyntax } } } }
```

Er zijn geen strikte regels om het *value* veld van het *enterpriseID* TLV in te vullen. Meestal gebruikt men ofwel de private OID van de constructor, ofwel de OID van de MIB module. Soms wordt een artificiële boomstructuur van foutcodes gecreëerd, en wordt het *enterpriseID* TLF op die manier gebruikt om een gedetailleerde foutboodschap te sturen. Messtal is het *specificTrap* TLV daarvoor ruimschoots voldoende.

Een ***GetNextRequest*** bericht laat toe om de exacte structuur van de MIB boom te achterhalen, en het heeft exact dezelfde structuur als een *GetRequest* bericht. En wordt net als dit beantwoord met een *GetResponse* bericht. In tegenstelling tot het *GetResponse* op een *GetRequest,* verwijst de *varBind* TLV van het *GetResponse* op een *GetNextRequest* echter niet naar het object met de aangevraagde OID, maar naar het object met de eerstvolgende OID in lexicografische volgorde . in deze volgorde worden alle objecten geordend, de SMI boomstructuur aflopende eerst in de diepte, en dan pas in de breedte, volgens oplopende RID.

Tenslotte is er in SNMPv2c nog de ***GetBulkRequest*** berichten. Deze laten toe om een deel van een tabel ineens op te halen. Een *GetBulkRequest* vraagt aan de agent om een *GetResponse* antwoord met zo veel mogelijk informatie terug te sturen. Onvolledige antwoorden zijn echter toegelaten. De structuur van een *GetBulkRequest* bericht is identiek aan dat van een *GetNextRequest* bericht:

```
GetBulkRequestMessage ::= SEQUENCE {
        version   INTEGER (0),
        community OCTET STRING,
        protocolDataUnit SEQUENCE {
                requestID      INTEGER,
                nonRepeaters   INTEGER,
                maxRepetitions INTEGER,
                varBindList    SEQUENCE OF {
                        varBind SEQUENCE {
                                name  OBJECT IDENTIFIER,
                                value ObjectSyntax } } } }
```

2 TLVs hebben echter een volledig andere interpretatie: in plaats van de *errorStatus* en *errorIndex* TLVs bevat een *GetBulkRequest* bericht *nonRepeaters* en *maxRepititions* TLVs. Het *nonRepeaters* TLV bepaalt op hoeveel *varBind* TLVs de agent dezelfde procedure moet toepassen als in een enkelvoudige *GetNextRequest* aanvraag: de overeenkomstige OIDs hebben bijgevolg meestal betrekkin op scalaire objecten, vermeld zonder .0 suffix.
Het *maxRepititions* TLV daarentegen bevat hoeveel keer de agent op de resterende *varBind* TLVs in de *varBindList* de iteratieve *GetNextRequest* procedure moet toepassen. Meestal gebruikt men dit om te verwijzen naar OIDS van kolomobjecten, en geeft *maxRepititions* TLV aan van hoeveel rijen maximaal men informatie wil bekomen. Opgelet: indien *maxRepititions* groter is dan het aantal effectieve rijen in de tabel, dan bekomt men informatie over kolommenen of andere objecten.
Het *GetResponse* antwoord op een *GetBulkRequest* bevat informatie over maximaal

*nonRepeaters + (#varBinds – nonRepeaters) * maxRepititions*

objecten. Indien kolommen van eenzelfde tabelobject opgevraagd worden, worden de cellen nu in volgorde teruggestuurd.

C3.

Overgang naar IPv6 en autoconfiguratie

a. Bespreek de belangrijkste *motieven* die uiteindelijk een overgang naar IPv6 zullen veroorzaken. (§5.1)
b. Hoe zal deze overgang worden gerealizeerd ? Van welke *technieken* zal men gebruik maken ? Je moet hierbij onder andere *tunneling* in algemene termen beschrijven, echter zonder dieper in te gaan op de praktische uitwerking ervan. (§5.7 zonder subsecties)
c. Geef de alternatieve mogelijkheden voor *autoconfiguratie* in IPv6. Bespreek hierbij de op eenvolgende stappen. (§5.6)

C3 a)

IPv4 is reeds meer dan 25 jaar, en het was oorspronkelijk ontwikkeld met als doel enkele duizenden toestellen in een internetwerk met elkaar te laten communiceren. Maar nu blijkt het nog steeds efficiënt te functioneren, ook al wordt het nu toegepast in het internet met tientallen miljoenen aangesloten stations. Naarmate echter de structuur, die bovenop IPv4 wordt gebouwd blijft doorgroeien, komen op verschillende vlakken een aantal problemen aan de oppervlakte:

- ***De beperking van de grootte van de adresruimte***. Door de adreslengte van 3é bits zijn er weliswaar 4 miljard mogelijk adressen, maar deze worden toegekend via een hiërarchisch adresseringssysteem (CIDR). Hierdoor hoeft er geen centrale autoriteit te bestaan die individuele adressen toewijst. Nadeel ervan is dan weer dat men op verschillende niveaus enorm veel toegewezen adresruimte verspilt: meer dan de helft van de beschikbare adressen zijn reeds toegewezen, maar geschat wordt dat niet meer dan 7% ervan ook effectief gebruikt wordt. Daarom wordt er nu strenger toegekeken op de toewijzing van netwerkadressen en bovendien omzeilen bedrijven het probleem door invoering van netwerkadresomzetting (NAT = Network Adress Translation). Maar in de nabije toekomst zullen er een enorm aantal bijkomende adressen moeten toegewezen worden door de web-compatibiliteit van ondermeer elektronische betaalterminals, PDA's, TV's en GSM's.
- Een ander voordeel van ***het hiërarchisch adresseringssysteem*** betreft de aggregatie van routes. Voor de meeste routes in een AS (Autonomous System) volstaan enkele routes ter aanvulling op de default route. Deze verwijst naar een ASBR(Autonomous System Border Router) van het AS, of naar een router van de ISP hoger in de hiërarchie. In principe kunnen enkel de ASBRs van de tier-1 ISPs (Internet Service Provider geen beroep doen op default routes (men noemt ze dan ook default-free routers), en moeten ze expliciet routes bevatten naar de adresblokken van alles andere tier-1 ISPs. Helaas is CIDR (Classless Inter Domain Routing) pas vanaf 1995 ingevoerd, en zijn de meeste tier-1 ISPs verantwoordelijk voor meerdere niet-aggregeerbare adresblokken. Hierdoor bevatten de default-free routers van het internet momenteel bijna 200.000 routes. Dit is zeer nefast voor het routingproces en dus voor de globale werking van het internet. Dit probleem manifesteert zich bovendien op veel kortere termijn dan de adresruimtebeperking.
- Een andere bron van performantieproblemen ***is de structuur van het IPv4 bericht***. Vooreerst werkt fragmentatie door opeenvolgende routers vertragend op het globale proces. Vervolgens worden berichtten met opties door veel routers als uitzonderingen beschouwd, en bijgevolg met lagere prioriteit verwerkt. Het routerontwerp is immers geoptimaliseerd voor IP verkeer met vaste headers. Bovendien zijn de mogelijkheden voor opties in de IPv4 header zeer beperkt. Het performantieverlies en de

implementatiebeperkingen zorgen ervoor dat opties zelden gebruikt worden. Nochtans zijn opties de enige mogelijkheid om recentere functie af te handelen.

- ***Het beveiligingsaspect***, de opvatting heerste dat beveiliging niet op de internetlaag thuishoorde: de verantwoordelijkheid voor de beveiliging van een toepassing werd in de hogere protocollagen of bij de toepassing zelf verondersteld. Bijv. SSL (Secure Socket Layer) werkt in de transportlaag en SHTTP (Secure HTTP) in de toepassingslaag werkt. Bij encryptie in de toepassingslagen blijft heel wat informatie onbeschermd en encryptie in de transportlaag vereist dan weer dat zowel de cliënt als het server gedeelte van de toepassing hiervoor aangepast wordt.
- ***Voorzieningen die beantwoorden aan de mobiliteit van computers zijn niet beschikbaar in IPv4 zelf***, maar worden gerealiseerd door protocollen uit de toepassingslaag, zoals DHCP. De connectiviteit van een toestel blijft afhankelijk van een correcte statische configuratie, hetzij lokaal, hetzij op een centraal beheerde server. Uitzondering hierop is het APIPA automatische IP-configuratie mechanisme

Voor elke van de deelproblemen bestaan bijgevolg een aantal tijdelijke oplossingen die de levensduur van IPv4 kunstmatig verlengen, maar het is reeds langer duidelijk dat IPv4 de voortdurende groei niet veel langer aankan.

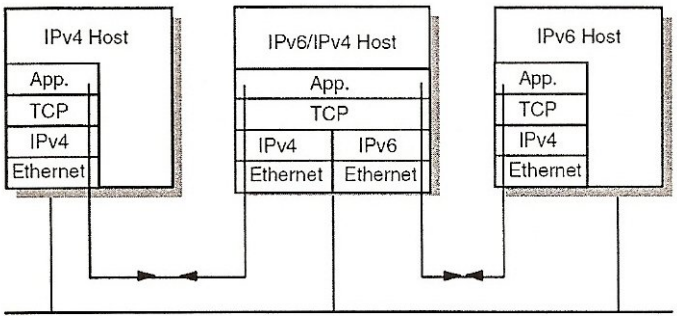
C3 b)

Door de komst van IPv6 is de nood aan de ARP en IGMP protocollen verdwenen, en moeten nieuwe versies voor ICMP en DHCP ingevoerd worden. Op subnetwerkniveau moet enkel de verwijzing naar de hogere protocollaag aangepast worden. Zo moeten Ethernet frames het Ethernet veld aanpassen van 0800 voor IPv4 naar 86DD voor IPv6. De hogere protocollagen moeten in principe enkel aangepast worden om rekening te kunnen houden met de 4voudige lengte van de IP adressen. De TCP *header* bijv bevat een *checksum* die berekend wordt op basis van het TCP segment, aangevuld met een *pseudo-header* bestaande uit de IP bron- en doeladressen. De *pseudo-header* heeft dan ook een aangepast formaat en inhoud indien TCP gebruikt maakt van IPv6.
De overschakeling naar IPv6 zal geleidelijk en betrekkelijk langzaam moeten gebeuren, rekening houdend met enorm aantal subnetwerken en knooppunten dat op het Internet is aangesloten, en met het aantal verschillende platformen waarop IPv4 draait. De protocollen en mechanismen die de overschakeling ondersteunen, moeten aan een aantal randvoorwaarden voldoen:

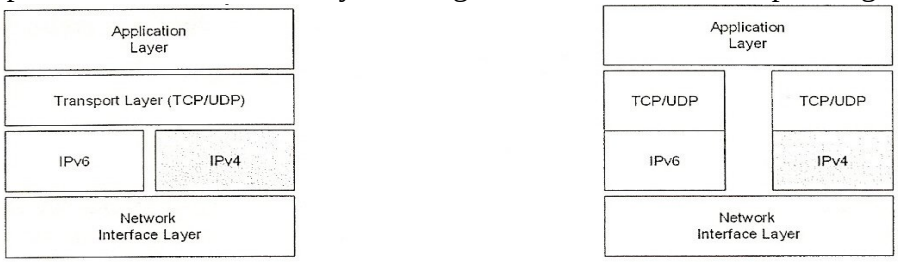
- De upgrade van IPv4 knooppunten moet op een willekeurig tijdstip kunnen uitgevoerd worden, onafhankelijk van de upgrade van andere knooppunten of routers.
- Nieuwe knooppunten, die enkel IPv6 aankunnen, in het vervolg alleen-IPv6 knooppunten genoemd, kunnen onafhankelijk van andere knooppunten of routers toegevoegd worden.
- Bestaande alleen-IPv4 knooppunten, moeten hun IPv4 adres w.x.y.z kunnen blijven gebruiken, zonder nood aan aanvullende adressen
- De systemen die een upgrade naar IPv6 hebben gehad, moeten op 1 of andere manier *interoperabel* met IPv4 systemen worden gehouden.

Ipv4 en IPv6 zullen jarenlang naast elkaar blijven bestaan, Verwacht wordt dat IPv6 op niveau van individuele ondernemingen vroeger zal worden ingevoerd, dan globaal in het Internet. Ook op schaal van de ondernemingen zelf zal de upgrade naar IPv6 eerder gebeuren in steeds groter worden eilandjes, op niveau van werkgroepen en departementen, en zal de backbone infrastructuur aan een lager tempo aangepast worden. Er zijn 3 technieken voor de overschakeling:

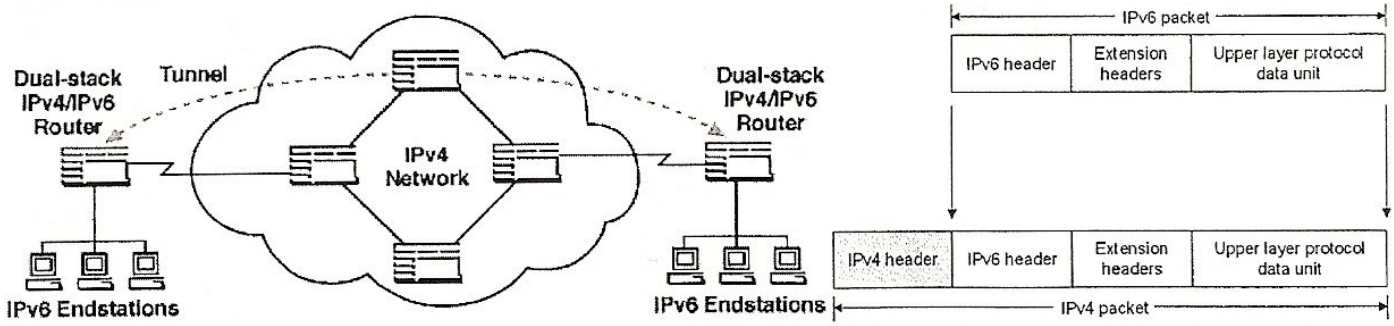
1. *aanwezigheid van een aantal Dual-IP of IPv6/IPv4 systemen*, die zowel IPv4 als IPv6 ondersteunen. IPv6/IPv4 systemen kunnen hierdoor communiceren met elk IPv4 of IPv6 knooppunt op hetzelfde subnetwerk.



Een *dual-IP* systeem moet enkel beschikken over een duale internetlaag. In veel implementaties, *dual-stack* systemen genoemd, is ook de transportlaag opgesplitst.



2. IPv6 over IPv4 *tunneling* kapselt IPv6 datagrammen, die van IPv6 eilanden door IPv4 oceanen verzonden moeten worden, in IPv4 datagrammen in, net alsof het data van een hogere protocollaag zijn. Het Protocol vel van de IPv4 *header* wordt ingevuld met de waarde 41. Het IPv6/IPv4 eindpunt van de tunnel verwijdert de IPv4 *header* en verwerkt de IPv6 *header*, ondermeer om de uiteindelijke IPv6 bestemming te achterhalen. Deze techniek is essentieel voor de 1ste fasen van de overschakelingsperiode, omdat het een *end-to-end* IPv6 dienst kan verzekeren, zonder een essentiële upgrade van de IPv4 infrastructuur, en zonder de werking van IPv4 diensten te verstoren. Ook in de latere fasen zal *tunneling* voor de connectiviteit blijven zorgen doorheen backbones die alleen IPv4 kunnen verwerken. *Tunneling* vereist dat de beide eindpunten van de tunnel *dual-stack* zijn. Tunnels moeten niet beschouwt worden als permanente verbindingen: ze bestaan enkel indien een specifieke transactie communicatie tussen de eindpunten vereist.



3. *Protocol* of *Header Translation*, uitgevoerd door IPv6/IPv4 routers, zet een IPv4 *header* in IPv6 formaat om, en omgekeerd. Hierbij worden ondermeer IPv4-gemapte adressen gekoppeld aan echte IPv6 adressen. Deze techniek zal vooral gebruikt worden in de laatste fasen van de upgrade, om de overblijvende alleen-IPv4 knooppunten te integreren in een nagenoeg compleet naar IPv6 gemigreerd Internet.*

C3 c)

IPv6 ondersteunt 2 mechanismen: *stateful* en *stateless*.
Zowel *stateful* als *stateless* autoconfigureerde adressen bevinden zich steeds in 1 van volgende toestanden;

- Tijdens de *Duplicate Address Detection* fase wordt een adres gekenmerkt als tentatief. Een knooppunt kan hierbij multicast, maar geen unicast verkeer ontvangen.
- Een als uniek bevestigd adres is geldig (*valid*) en kan unicast berichten zowel versturen als ontvangen. Een geldig adres bevindt zich hetzij in de *preferred*, hetzij in de *deprecated* toestand. In deze laatste toestand wordt van het knooppunt verwacht ofwel bij voorkeur een ander adres (in de *preferred* toestand) te gebruiken, ofwel de geldigheidsduur van het huidige adres zo vlug mogelijk te verlengen.
- Na verstrijken van de levensduur is een adres ongeldig (*invalid*) en kan het noch unicast, noch multicast verkeerd verwerken.

Stateless autoconfiguratie:

Stateless autoconfiguratie biedt aan individuele knooppunten de mogelijkheid hun IPv6 configuratie te bepalen, zonder dat ze expliciet een server moet ondervragen die informatie over elk knooppunt bezit.
Stateless autoconfiguratie biedt geen controle op wie een IPv6 adres ontvangt: elk knooppunt kan op een verbinding worden aangesloten, de prefix informatie verwerken die routers bekendmaken en een geldig IPv6 adres creëren.
Stateless autoconfiguratie is een zeer eenvoudige procedure: het *interface-id* kan berekend worden op basis van het IEEE EUI-64 formaat van een MAC adres, terwijl de netwerkprefix via *Router Discovery* kan bekomen worden.

- Het knooppunt berekent op basis van het MAC adres een tentatief *linklokaal* unicast adres. Dit geeft het knooppunt reeds de mogelijkheid om te communiceren met andere
- knooppunten op dezelfde netwerkverbinding. Vooraleer het knooppunt dit *linklokaal* unicast adres kan gebruiken, moet het via *Duplicate Address Detection* verifiëren of het adres uniek is voor het lokale subnetwerk.
- De interface wordt geïnitialiseerd met het geldig verklaarde linklokale unicast adres. Tevens wordt op de netwerkkaart het multicast MAC adres overeenkomstig het *sollicited-node* multicast adres geregistreerd. Dit multicast MAC adres bekomt men uit het MAC adres door de eerste 3 bytes te vervangen door 33-33-FF.
- De interface stuurt dan 3 opeenvolgende *Router Solicitation* berichten. Indien geen *Router Advertisement* ontvangen worden, wordt overgeschakeld op een procedure voor *stateful* autoconfiguratie. Indien wel, worden de instellingen voor de *Hop Limit*, de *Reachable Time*, de *Retrans Timer* en de aanbevolen MTU er uit afgeleid.
- Voor elke *Prefix Information* optie waarvan de *Autonomous* vlag ingesteld is, berekent men op basis van de prefix een *tentatief* unicast adres, verifieert men er via *Duplicate Address Detection* de geldigheid van, en voert men de initialisatie van de interface uit. Hierbij bepalen de *Preferred Lifetime* en *Valid Liftime* velden uit de *Prefix Information* de perioden waarin het adres als *preferred* en *deprecated* mag beschouwd worden. Indien de prefix correspondeert met een publiek subnetwerk, dan wordt de *interface-id* niet bepaald door het MAC adres, maar at random gegenereerd. Dit verzekert een bepaald niveau van anonimiteit.
- Indien de *Managemed Address Configuration* vlag in het *Router Advertisement* bericht ingesteld is, wordt een *Stateful* autoconfiguratie uitgevoerd om bijkomende adressen te bekomen. Indien de *Other Statfeul Configuration* ingesteld is, wordt een *Stateful* autoconfiguratie uitgevoerd om bijkomende configuratieparameters te bekomen.

Stateful autoconfiguratie (met behulp van DHCPv6):

Stateful autoconfiguratie houdt op een centrale server over alle knooppunten toestandsinformatie bij, die, ondersteunt door een protocol zoals DHCP, door de individuele knooppunten opgehaald kan worden. Nadelig hierbij is,vooral voor kleinere omgevingen, dat een speciaal serverprogramma moet beheerd worden.
Via stateful autoconfiguratie daarentegen kunnen enkel knooppunten worden geconfigureerd, die expliciet door de netwerkbeheerder zijn geautoriseerd. Stateful autoconfiguratie biedt eveneens een nagenoeg volledige controle over het adresseringsschema, interface-id's inclusief.
Net als elk ander protocol dat rechtstreeks met IP adressen werkt, heeft DHCP een upgrade nodig om IPv6 adressen te ondersteunen.