

Notities Netwerken II - 2021

Robin De Bock

Hoofdstuk 1: Lagen van de internet protocol stack	6
OSI model	6
Encapsulation/decapsulation	7
IP: ICMP, Fragmentation (labo 1)	7
Hoofdstuk 2: applicatielaag	8
Sectie 2.1 (Communicatie) Principes van netwerkapplicaties	8
Poortnummers opgelijst	8
Sectie 2.7/3.2 Sockets programmeren en de-/multiplexen	9
Voorbeeld socket (in Python)	9
Dynamic server process creation	9
Sectie 2.2-2.2.3 HTTP	10
HTTP : Request Message	11
HTTP: response message	12
HTTP evolution and versions	12
Principle of web server configuration	12
Sectie 2.2-2.2.3 Persistente vs. niet-persistente HTTP	13
Non-persistent HTTP	13
Non-persistent HTTP: Response Time	13
Persistent HTTP	13
Commando's	14
netstat: Monitoren van socket informatie op een host	14
telnet - Interactie via TCP-socket naar server	14
netcat - Zwitsers zakmes voor netwerken	15
Sectie 2.2.4 User-Server Interaction: Cookies	15
Sectie 2.2.5 Web Caching	17
Sectie 2.2.6 The Conditional GET	17
Sectie 2.4 DNS	18
DNS Resource Records (RR)	18
Hoe werken DNS servers samen	19
Recursive/Iterative DNS query	19
DNS resolver	21
DNS registratie	22
Interactie tussen e-mail en DNS (voorbeeld)	25
DNS server	26
Reverse DNS	30
DNS importance & security	31
Zelftest DNS	33
Hoofdstuk 4: Netwerklaag - datalevel	34
Sectie 4.3.3 IPv4 addressing	34
Hoe werken IP adressen	34
IPv4 addressing, speciale netwerken	34
Efficiënt gebruik (opdelen) van IP adresblokken	34
Forwarding in een IP router	35

Voorbeeld IP forwarding	35
Aggregatie van IPv4 adressen	36
Response college	38
Sectie 4.3.3 IPv4 Dynamic Host Configuration Protocol	42
DHCP (Dynamic Host Configuration Protocol)	42
DHCP renewal & relay	45
DHCP client	48
DHCP server	49
Sectie 4.3.4 Network Address Translation Protocol (NAT)	51
H4*: Netwerklaag - IPv6	54
IPv6 address space & notation	54
Adres verkorten	54
IPv4 adres opnemen in IPv6	54
Poortnummers gebruiken met IPv6 adres	54
IPv6 address scope	55
Scope	55
IPv6 ULA (Unique Local Address) scope	57
IPv6 address type	58
Multicast adressen	58
Anycast	58
Preserved addresses	59
Response college	60
Oefening: configureren van klein netwerk. (local, global IPv6)	60
Oefening 2: routeringstabel router	61
Oefening 3: geldige IP adressen en scope/type van adressen	61
Oefeningensessie	63
IPv6 address resolution	68
Solicited-node multicast	68
Address Resolution IPv6 (NS & NA)	70
IPv6 Duplicate Address Detection (DAD)	72
IPv6 StateLess Address AutoConfiguration (SLAAC)	73
IPV6 - DHCPv6	74
H5: Netwerklaag	75
Sectie 5.1 Introductie	75
Routing <> Forwarding	75
Router architecture overview	75
Globale routeringsproces	76
Routing Protocol Scope	77
Routing Protocol Type	78
Routing Protocol Overview	78
Sectie 5.2 Routeringsalgoritmen: distance vector routing	79
Werking:	79
Bijzondere karakteristieken:	80
Count to infinity (traag slecht nieuws)	80

RIP (Routing Information Protocol) (intra-AS)	81
RIP routing daemon	81
Sectie 5.3 Intra-AS routering: Link-state routing & OSPF	82
Link-State Packets	82
Link-State Routing Protocol Overview	83
Link-State Routing Protocol – OSPF (Open Shortest Path First)	83
OSPF routing daemon	85
OSPF messages	85
Link-state advertisement (LSA)	85
OSPF “advanced” features	85
Hierarchical OSPF (2-levels)	86
Hierarchical OSPF takeaway	87
Link-State vs. Distance Vector Routing	87
Uitleg zelftest vanaf slide 27	88
Sectie 5.4 Routing among the ISPs: structuur van het internet	89
Sectie 5.4 Routing among the ISPs: BGP	91
BGP principles (3 principes)	91
Principle 1: Advertisement of reachability	93
Principle 2: Propagation	93
III. Route selection	96
Response college	98
BGP in practice	101
Sectie 5.5 Programming the network	102
Sectie 5.5 Software Defined Networking (SDN): Programmability + Software Defined Networks	104
Het probleem	104
De evolutie	104
Klassieke router architectuur	104
Openflow (protocol)	105
Openflow forwarding abstraction	106
OpenFlow data plane	106
Control mechanisms - flow insertion	107
SDN architectuur componenten	109
Applications	109
Sectie 5.5 Programming the network: Netwerkvirtualisatie	110
Datacenter components	110
Datacenter components - Virtual switch	110
Virtual networking & tunneling	112
L2 Ethernet network over IP	112
Network recovery in the modern datacenter	114
Sectie 5.6 Internet Control Message Protocol (ICMP) (control level)	115
Sectie 5.7 Netwerkbeheer en SNMP (control level)	118
FCAPS (acroniem)	118
Network management architecture	119
Korte geschiedenis van netwerk management (niet zo belangrijk)	120

Simple Network Management Protocol (SNMP)	121
1. Management Information Base (MIB)	121
2. Structure of Management Information (SMI): data definition language	121
3. SNMP protocol	124
4. SNMP security and administration	125
Net-SNMP	125
Netwerk automatisering met Ansible	126
Ansible execution modes	126
Typische use cases van Ansible	126
Ansible architecture	127
Key Ansible features	129
H6 Linklaag	129
Sectie 6.4 Local-Area netwerken met switches	130
ARP (Address Resolution Protocol)	130
ARP cache management	131
Ethernet switching	131
Forwarding	133
Virtual Local Area Network (VLAN)	136
Gratuitous ARP	138
Response college	139
H8: beveiliging	141
8.9 Firewall and IDS	141
Waarom firewalls (onzichtbaar)	141
1) Stateless packet filters	141
Stateful packet filtering	143
Application gateways	144
Limitations of Firewalls, Gateways (onzichtbare slide)	144
Intrusion Detection Systems (IDS)	146
Oefening instellen packet filtering (firewall)	147

Hoofdstuk 1: Lagen van de internet protocol stack

OSI model

L5	Application layer
L4	Transport layer
L3	Network layer
L2	Link layer
L1	Physical layer

5) Application layer

Maakt webapplicaties mogelijk. *Een abstractielag die communicatieprotocollen en koppelvlakken specificeert, waarmee computers met elkaar kunnen communiceren over telecommunicatienetwerken en computernetwerken.*

Protocollen: FTP, HTTP, SMTP

4) Transport layer

Dient voor process - process communicatie. Het bepaalt via welke poorten toepassingen op het doelsysteem kunnen worden aangestuurd. Met andere woorden worden datapakketten toegewezen aan een bepaalde toepassing.

Protocollen: TCP, UDP

3) Network layer

Dient voor routering en de controle van de gegevensstroom.

Maakt gebruik van logische adressering aka IPv4/IPv6 adressen.

Protocollen: IP, routeringsprotocollen

2) Link layer

Dient voor voorkomen van overdrachtsfouten en gegevensoverdracht tussen naburige netwerk elementen.

Maakt gebruik van hardware adressering aka MAC adressen

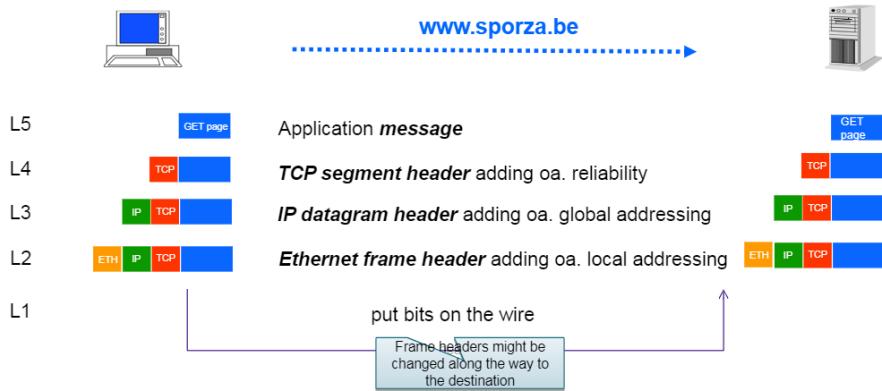
Protocollen: Ethernet, PPP, 802.11 (WiFi)

1) Physical layer

Bits "op het kanaal" als een fysiek signaal.

Protocollen: DSL, ISDN, Bluetooth, USB

Encapsulation/decapsulation



IP: ICMP, Fragmentation (labo 1)

ICMP (Internet Control Message Protocol)

De werking van IP steunt hierop. In IP zitten ook zaken om de **werking van de netwerklaag** na te gaan, zonder de applicatielaag, zonder de transportlaag (TCP/UDP). Hierin zitten bepaalde functionaliteiten zoals 'ping', maar ook 'destination host unreachable' → niet gekend in een router, probleem in een router.

In ICMP zit ook **Time To Live Expired**. Dit kan misbruikt worden door de TTL op 1 te zetten en naar de router te sturen. Die router stuurt een 'TTL exceeded'. De eerste hop router is dus geïdentificeerd. Zo verder doen om alle hop routers te identificeren. Dit doet 'Traceroute' op Windows of 'traceroute' op Linux.

IP fragmentation

IP segmenten moeten passen op de **MTU (Maximum Transmissible Unit)** van de **datalink laag**. IP pakket is te groot dus wordt het opgedeeld in meerdere stukken zodat het op een bepaalde uitgaande link kan verzonden worden. Hierbij wordt de IP header gedupliceerd over meerdere pakketten → oorspronkelijke header + 'het is gefragmenteerd', met dan de payload die verdeeld is. Dit komt niet zoveel voor omdat TCP zich probeert af te stemmen op de MTU van de link.

Hoofdstuk 2: applicatielaag

Sectie 2.1 (Communicatie) Principes van netwerkapplicaties

Wat is er nodig voor communicatie tussen netwerkapplicaties?

De eerste is het **adresseren van externe processen**. We maken gebruik van **IP adressen** om nodes (ofwel interfaces) te kunnen lokaliseren. Daarnaast is er een mechanisme om op die host de juiste **applicatie** te vinden, een **poortnummer**.

Ten tweede gaan we kijken naar het **type connectiviteit**. Dit wordt voorzien door de onderliggende **transportlaag**. Enerzijds de betrouwbare byte stream service **TCP** (Transmission Control Protocol), anderzijds de 'best-effort' datagram service **UDP** (User Datagram Protocol).

Voorbeeld TCP: browsen van het web.

Webservers wachten passief tot clients een request sturen op poort 80. Web browser kunnen dus nu HTTP berichten sturen naar poort 80 op het IP adres van de webserver.

Poortnummers van clients zijn niet vast, maar worden tijdelijk toegekend door het OS, toegelaten nummers starten vanaf poortnummer 1024 (afhankelijk van OS).

Voorbeeld UDP: DNS

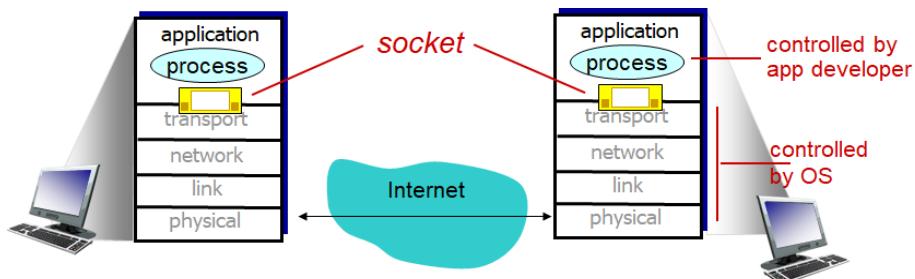
Ook hier luisteren DNS servers op een poort, poort 53.

Poortnummers opgelijst

Application/protocol	Usual server port	Transport Layer Protocol
FTP	20,21	TCP
SSH	22	TCP
Telnet	23	TCP
SMTP (email transfer server)	25	TCP
DNS	53	UDP/TCP
HTTP (web)	80, 443 (secure)	TCP
POP (email server)	109, 110	TCP

Sectie 2.7/3.2 Sockets programmeren en de-/multiplexen

Een socket is een toegangspoort waardoor een applicatie kan communiceren met een andere. Een socket is een **Operating System Call for network programming** (netwerk programmering). De meest bekende API (Application Programming Interface) is gebaseerd op **Berkeley sockets**. Zo'n toegangspoort of socket wordt gedefinieerd door 5 zaken (5-tuple): het IP en poortnummer van bron en bestemming, alsook het gebruikte transport protocol.



Voorbeeld socket (in Python)

TCP server	<pre>from socket import * s = socket(AF_INET, SOCK_STREAM) s.bind(('127.0.0.1', 9999)) s.listen(5) while True: c, addr = s.accept() g = bytes("Hello %s" % addr[0]) c.send(bytes(g,'utf-8')) c.close()</pre> <p>→ AF_INET = IPV4, SOCK_STREAM = TCP → zal maximaal 5 inkomende verbindingen toestaan → c = nieuw socket object, want we beschikken over alle informatie</p>
TCP client	<pre>from socket import * s = socket(AF_INET, SOCK_STREAM) s.connect(('127.0.0.1', 9999)) s.send(b'Hello, world') data = s.recv(1024) s.close() print 'Received', data</pre> <p>→ Actief verbinding maken met het serverproces → poortnummer wordt gekozen door OS, parameter is buffergrootte</p>

Dynamic server process creation

Dit is het koppelen van een thread aan het afhandelen van een clients. De threads specifiek voor deze taak worden **request handlers** genoemd en zitten samen in een **thread pool**. Terwijl de server hetzelfde IP adres en poortnummer hanteert, maakt elke client gebruik van een ander IP adres en poortnummer. Omdat poortnummers worden bepaald door het besturingssysteem kan het voorkomen dat twee clients hetzelfde poortnummer hebben. Dit is geen probleem aangezien sockets worden bepaald door zowel IP adres als een poortnummer.

Sectie 2.2-2.2.3 HTTP

HTTP (Hypertext Transfer Protocol) is een typisch voorbeeld van een **client-server protocol**. Een webserver biedt objecten aan die opgevraagd kunnen worden met het HTTP protocol. Ieder object dat kan opgevraagd worden heeft een **URL (Uniform Resource Locator)**: <domein/hostnaam/IP> adres (:poortnummer) / <bestandspad of object>. Dit HTML bestand is de basis en kan **referenties** bevatten naar andere objecten (met elk hun eigen URL).

HTTP maakt gebruik van **TCP** en zijn **3-way-handshake**.

Als eerste luistert de server (open socket, standaard poort 80).

Nadien zal de cliënt de TCP connectie initiëren (**3-way-handshake** stap 1) (eigen socket creëren met poortnummer > 1024) met de server op poort 80.

De server zal de connectie accepteren van de client (**3-way-handshake** stap 2).

De server stuurt een bevestiging (**3-way-handshake** stap 3).

HTTP berichten (**request/response** messages) worden uitgewisseld tussen browser (HTTP client) en Web server (HTTP server). Afhankelijk van de HTTP versie zal er steeds een nieuwe TCP connectie opgesteld worden of kan de connectie hergebruikt worden.

Nadien wordt de connectie gesloten (**3-way-handshake**).

HTTP is **stateless**, de server houdt **geen toestandsinformatie** van requests bij die eerder van een client kwamen. Dit was doelbewust, omdat bij stateful protocollen er aparte mechanismen moeten zijn om problemen op te vangen bij het crashen van nodes (de state van server en client kan dan verschillen).

Eenmaal dat de TCP connectie is opgesteld maakt HTTP gebruik van twee soorten berichten: **request messages** en **response messages**. Alle berichten worden in plain ASCII tekstformaat over de socket verzonden (< HTTP versie 2).

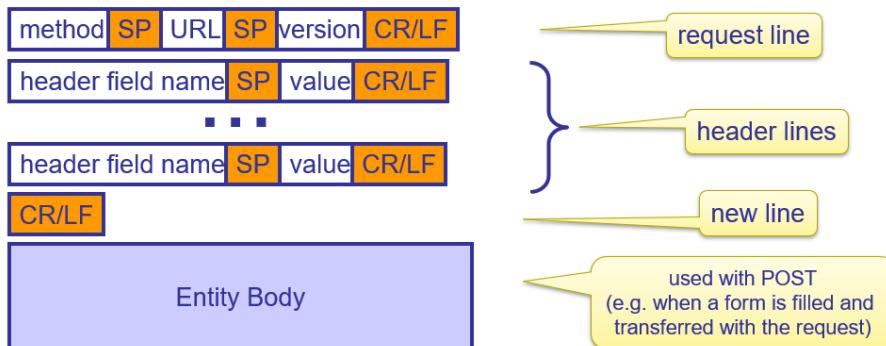
HTTP : Request Message

De berichten gaan van **client** naar **server**.

Request methods:

- HTTP/1: GET, POST, HEAD
- HTTP/1.1: PUT, DELETE

Diagram request message



Voorbeeld: request message in tekstvorm

Header velden laten toe extra informatie mee te geven zodat de webserver de response hierop kan afstemmen. Afgesloten door twee controle karakters: carriage return en line feed.

```
GET / HTTP/1.1
Connection: keep-alive
User-agent: Mozilla/5.0 Chrome/69.0...
Accept: text/html,image/webp,image/apng
Accept-language: en,en-GB,nl
```

- **Connection: keep-alive:** huidige TCP connectie openhouden en hergebruiken voor volgende request naar dezelfde webserver
- **User-agent:** browser (normaal maar 1 waarde gespecificeerd)
- **Accept:** (in voorbeeld) zowel tekst als figuren kunnen gelezen worden in de response

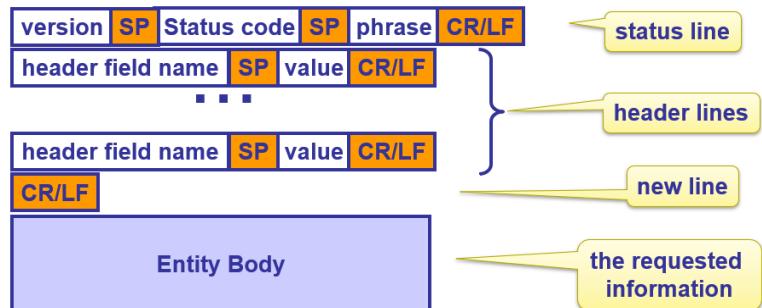
Uploading form input

Hiervoor zijn er twee manieren, de eerste manier maakt gebruik van de **POST method** en geeft de ingevoerde gegevens door in de **body** van de request. De tweede manier maakt gebruik van de **GET method** en geeft de ingevoerde gegevens door in de **URL** van de request.

HTTP: response message

Het zijn berichten van **server** naar **client** die **status codes** bevatten om verslag te geven.

Diagram response message



Voorbeeld: response message in tekstvorm

```
HTTP/1.1 200 OK
Date: Fri, 21 Sep 2018 13:49:54 GMT
Server: Apache/2.4.25 (Debian)
Last-Modified: Fri, 22 Sep 2017 13:05:41 GMT
ETag: "3b2-559c6ddb38ba8-gzip"
Accept-Ranges: bytes
Content-Length: 634
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

- **Date:** tijdveld, dit geeft het moment aan dat de response werd gegenereerd. Niet te verwarren met het veld last-modified, wat aangeeft wanneer de content het laatst werd gewijzigd.
- **Content-length**
- **Content-type**
- **Server:** type webserver
- **ETag:** fungeert als een soort hash van het gevraagd object en kan door **caches** gebruikt worden zodat geen nieuwe content moet opgevraagd worden.
- **Accept-Ranges:** webserver geeft aan dat hij toelaat om partiële objecten door te sturen met een gegeven byte range.
- **Keep-alive:** instellingen voor TCP, timeout en hoeveel requests er maximaal binnen de connectie zullen worden toegelaten.

HTTP evolution and versions

Vanaf versie 2 werd het een **binair protocol** in plaats van tekst gebaseerd. Het laat ook toe om verdere compressie te doen, wat de latency verder verlaagt.

Principle of web server configuration

Veel gebruikte webserver implementaties zijn Apache of NGINX, daarmee kan je eenvoudig websites opzetten bestaande uit statische HTML pagina's en bijbehorende objecten. Je kan ook een complexe dynamische webapplicatie maken bovenop een framework zoals bv. Python. De webserver fungeert dan als een soort **entry point**, ook wel een **reverse proxy** genoemd, die luistert op poort 80 naar binnengkomende HTTP requests. Eenmaal geparsed kunnen die doorgestuurd worden naar de webapplicatie via een protocol zoals uWSGI. Die zal dan een webpagina of object genereren die terug naar de webserver gestuurd wordt via de uWSGI interface.

Sectie 2.2-2.2.3 Persistente vs. niet-persistente HTTP

Non-persistent HTTP

Voorbeeld:

- 1) **HTTP client** initiates TCP connection to HTTP server (process) at idlab.technology on port 80
- 2) **HTTP server** at host www.ibcn.intec.UGent.be waiting for TCP connection at port 80 "accepts" connection, notifying client
→ three way handshake
- 3) **HTTP client** sends HTTP request message (containing URL) into TCP connection socket. Message indicates that client wants object infrastructure/virtual-wall/index.php
- 4) **HTTP server** receives request message, forms response message containing requested object, and sends message into its socket
- 5) **HTTP server** closes TCP connection.
- 6) **HTTP client** receives response message containing html file, displays html and closes TCP connection. Parsing html file, finds 10 referenced .jpg objects
- 7) Steps 1-6 repeated for each of 10 jpg objects

Non-persistent HTTP: Response Time

Definitie van **RTT (Round Trip Time)**: tijd die nodig is om een klein pakket te versturen van client naar server en terug.

De totale tijd zal hier bestaan uit **2 RTT + transmissietijd**, dat laatste is de tijd die nodig is om de data te ontvangen (verzenden vanaf de server).

De eerste RTT is steeds om de (eerste twee stappen van) **TCP connectie** op te zetten en de tweede is de **HTTP request, response**. De data wordt meegestuurd met deze tweede.

De RTT gaat over tijd en niet over aantal pakketten of de grootte, daarom wordt de HTTP request en response met data gezien als 1 RTT en een bepaalde transmissietijd.

Persistent HTTP

HTTP versie 1 maakt gebruik van niet-persistente verbindingen, vanaf versie 1.1 kunnen **meerdere objecten per TCP sessie** verzonden worden. Dit elimineert de nood aan een nieuwe TCP connectie per HTTP request.

Persistent zonder pipelining

Kan pas een nieuwe request sturen als de **vorige is ontvangen**.

→ 1 RTT voor elk object dat wordt opgevraagd.

Persistent met pipelining:

Versturen zonder te wachten op responses.

→ overhead kan beperkt worden tot 1 RTT voor alle request zijn. HTTP versie 2 gaat nog verder dan 1.1 in dat de responses in een **andere volgorde** kunnen toekomen als dat ze verzonden zijn.

Commando's

netstat: Monitoren van socket informatie op een host

```
[alf@server ~]$ netstat -atu
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp      0      0 0.0.0.0:ssh              0.0.0.0:*
tcp      0      0 server:ssh              192.168.1.48:59476  ESTABLISHED
tcp6     0      0 [::]:mysql             [::]:*
tcp6     0      0 [::]:ssh               [::]:*
udp      0      0 0.0.0.0:slingshot       0.0.0.0:*
udp      0      0 0.0.0.0:7091           0.0.0.0:*
```

-a list all ports, also listening

-t list tcp ports

-u list udp ports

-n list numbers instead of domains or applications (e.g., 22 instead of ssh)

-p bijhorend programma (pad)

telnet - Interactie via TCP-socket naar server

De telnet-client is een 'generieke' TCP-client. Zendt alles wat u typt naar de TCP-socket. Print wat terugkomt via de TCP-socket. Handig voor het testen van TCP-servers (op ASCII gebaseerde protocollen). Voorbeelden: Putty (Windows) en netstat CLI commando (Linux)

```
$ telnet www.UGent.be 80
Trying 157.193.40.33...
Connected to sangoku.ugent.be.
GET / HTTP/1.1

HTTP/1.1 200 OK
Date: Fri, 18 Feb 2000 15:46:11 GMT
Server: Apache/1.3.6 (Unix)
Last-Modified: Tue, 21 Dec 1999 13:44:47 GMT
ETag: "a-8ae-385f844f"
Accept-Ranges: bytes
Content-Length: 2222
Connection: close
Content-Type: text/html

<html>
  <head>
    <title>UNIVERSITEIT GENT - UNIVERSITY OF
    GHENT</title>
    <style type="text/css">
      ...
    </style>
  </head>

  <body bcolor="#000066" link="#cccccc"
    vlink="#cccccc" alink="#666666">
    ...
  </body>
</html>
Connection closed by foreign host
```

The diagram illustrates the sequence of events during a telnet session. It shows the client's request to connect to port 80 of the UGent website, followed by the server's response with an HTTP 200 OK status and various headers. The server then sends the HTML content of the homepage. Finally, the connection is closed. Brackets on the right side group the output into five distinct phases: 'connection set-up', 'request to get UGent homepage', 'reply header', 'HTML document UGent homepage', and 'close connection'.

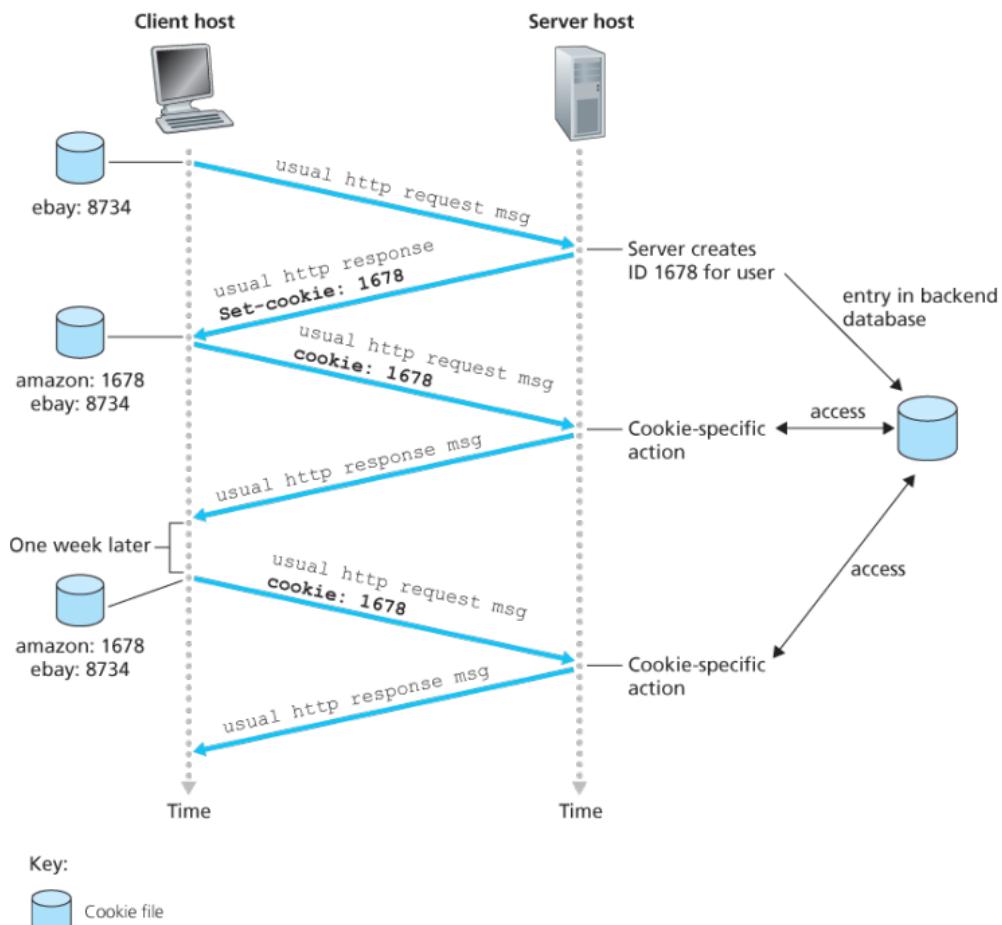
netcat - Zwitsers zakmes voor netwerken

```
C:\Users\USERNAME\Desktop\Netcat>nc -h
[v1.11 NT www.rodneybeede.com/]
connect to somewhere: nc [-options] hostname port[s] [ports] ...
listen for inbound:  nc -l -p port [options] [hostname] [port]
options:
-d          detach from console, background mode
-g gateway  source-routing hop point[s], up to 8
-G num     source-routin
-h          this cruft
-i secs    delay interval for lines sent, ports scanned
-l          listen mode, for inbound connects
-L          listen harder, re-listen on socket close
-n          numeric-only IP addresses, no DNS
-o file    hex dump of traffic
-p port    local port number
-r          randomize local and remote ports
-s addr    local source address
-u          UDP mode
-v          verbose [use twice to be more verbose]
-w secs    timeout for connects and final net reads
-z          zero-I/O mode [used for scanning]
port numbers can be individual or ranges: m-n [inclusive]
```

Sectie 2.2.4 User-Server Interaction: Cookies

Het is vaak wenselijk dat een website gebruikers **identificeert**, hetzij vanwege de server de gebruikerstoegang wil beperken of omdat het inhoud wil aanbieden in functie van de gebruikers identiteit.

Cookie technologie bestaat uit vier componenten: (1) een **cookie header line** in de **HTTP-request**. (2) een **cookie header line** in de **HTTP-response**. (3) een **cookie bestand** dat wordt bewaard op het eindsysteem van de **gebruiker** en beheerd wordt door de browser van de gebruiker. En (4) een **back-end database** op de **website**.



Sectie 2.2.5 Web Caching

Een **webcache**, ook wel **proxyserver** genoemd, is een netwerk entiteit die voldoet aan HTTP-verzoeken **namens de oorspronkelijke webserver**. Een cache is zowel een **server** (voor de client) als een **client** (voor de server).

Voorbeeld:

Browser stelt TCP connectie op met de Web Cache (Proxy server) en stuurt een HTTP request voor een object in de Web cache.

Web cache controleert of het object reeds lokaal aanwezig is.

Indien ja, stuurt deze die terug met een HTTP response bericht.

Indien neen, opent de proxy een TCP connectie naar de server voor het object en krijgt deze met een HTTP response. Bij ontvangst wordt het object opgeslagen in het geheugen en een kopie teruggestuurd naar de client via een HTTP response.

Web caching wordt gebruikt voor twee grote redenen:

De eerste is dat het de **responsijd** dramatisch kan verminderen voor een client request.

Vooral als de bandbreedte tussen de client en de origin server veel kleiner is dan tussen de client en de cache.

Ten tweede kan een cache het **verkeer met het internet** van een instelling dramatisch verminderen. Hierdoor moet de instelling de bandbreedte naar buiten (het internet) niet upgraden, wat zeer duur is.

Verder kunnen caches het **globale internetverkeer** verminderen, wat positief is voor iedereen.

Door het gebruik van **Content Distribution Networks (CDN's)** spelen web caches steeds vaker een belangrijke rol op internet. Een CDN-bedrijf installeert veel **geografisch verspreide caches voor het internet**, waardoor een groot deel van het verkeer wordt gelokaliseerd.

Sectie 2.2.6 The Conditional GET

Caching introduceert een nieuw probleem, het object in de cache verouderd zijn. In HTTP is een mechanisme ingebouwd om dit te verhelpen, **Conditional Get**. Een HTTP request is een conditional Get als het de **GET** methode bevat en een header line 'If-Modified-Since:' .

Als de cache een object in geheugen heeft en er komt een GET request voor dat object, kan het zijn dat dat object in de tussentijd is geüpdatet. Het stuurt een **Conditional Get** met als waarde voor 'If-Modified-Since' header line, dezelfde waarde als voor de 'Last-Modified' header line van het object op de server. Zo laat de cache de server weten enkel het bestand te verzenden indien het gewijzigd is sinds dat tijdstip. Dus zijn er twee opties:

De eerste is dat het object niet is gewijzigd, 'HTTP/1.1 304 Not Modified' met een lege body.
De tweede is dat het gewijzigde object wordt teruggestuurd.

Sectie 2.4 DNS

Domain Name System (DNS) is een **client-server protocol** in de **applicatielaag**. Het maakt gebruik maakt van een **gedistribueerde database** dat de **mapping tussen domeinnamen en IP adressen** bijhoudt.

DNS Resource Records (RR)

In een DNS slaan we meerdere dingen op. Een lijst van wereldwijde root servers, lijst van namen (host, name server, mail server, ...) en hun corresponderende IP adressen. Verder nog alias namen en hun canonical naam. Laatste voorbeeld is een lijst van IP adressen en hun corresponderende namen (voor reverse-lookup).

Deze data wordt opgeslagen aan de hand van een **Resource Record (RR)**.

[name], [TTL], [class], record-type, record-data

→ **name** : name to be resolved

→ **TTL** : hoe lang record **gecached** mag worden. Afweging tussen **consistentie** en **netwerk load**.

→ **class** : IN (voor Internet)

→ **record-type** : bv. NS, A, MX, CNAME

→ **record-data** : bv. IP address

Record types/ Record-data

- **A** (IPv4) of **AAAA** (IPv6): de naam is de **hostname** en de record-data is het **IP adres**
 - *webserv1.intec.ugent.be IN A 157.193.22.45*
- **NS** : de naam is een **domein** en de record-data is de **hostnaam van een server** die de **IP-adressen** in dat domein weet te bemachtigen (**authoritative name server**).
 - *ugent.be IN NS ugdns1.ugent.be* → (authoritative name server for *ugent.be*)
 - *intec.ugent.be IN NS dns1.intec.ugent.be* → (authoritative name server voor *intec.ugent.be*)
- **CNAME** : de naam is een **alias voor een hostname** en de record-data is de bijhorende **canonical name**
 - *www.intec.ugent.be IN CNAME webserv1.intec.ugent.be*
- **MX** : de naam is een **domeinnaam** en de record-data is de bijbehorende servernaam van een **mailserver** (MTA). **'preference'** duidt de primaire, secundaire, ... mailservers aan voor het domein.
 - *intec.ugent.be IN MX preference=10 mail-tech.intec.ugent.be*
preference=20 mail4.intec.ugent.be
preference=30 cedar.ugent.be
- **PTR** : de naam is een **aangepast IP adres** en de record-data is de bijbehorende **servernaam. (reverse DNS)**
 - *2.212.193.157.in-addr.arpa. IN PTR tacitus.ugent.be* → (reverse DNS entry)

Hoe werken DNS servers samen

Indien een instelling een domeinnaam wil gebruiken, moet deze de domeinnaam registreren bij **DNS registrar**. Deze houdt bij welke domeinnamen nog beschikbaar zijn.

Nadien moet de instelling een DNS server instellen om de mapping te voorzien tussen het IP adres en de domeinnaam, de **authoritative DNS server** (authoritative name server) van het domein. Dit kan de instelling zelf doen of overlaten aan een ISP.

Resolver: een lokaal programma dat op de pc van de client draait en kan interageren met de DNS infrastructuur. Het stuurt een mapping request naar een Local Name Server.

Local Name Server: een DNS server die een netwerkbeheerder of ISP heeft ingesteld als eerste aanspreekpunt (**Default Name Server**). De client leert het adres van deze DNS server meestal via het **DHCP protocol**.

Root Name Server: houden een overzicht bij van welke andere DNS servers je request kunnen beantwoorden. Ze doen dit op een **hiërarchische** manier. In eerste instantie houden ze adressen bij van DNS servers die verantwoordelijk zijn voor top-level domeinen (.be; .com; ...). Op hun beurt houden ze weer bij welke DNS server verantwoordelijk zijn voor welke subdomeinen van het gegeven top-level domein.

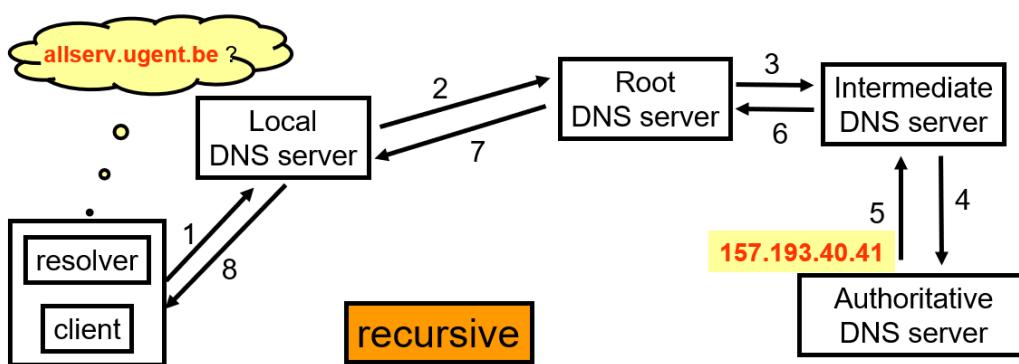
Recursive/Iterative DNS query

Het **proces van een DNS query** kan **recursief** of **iteratief** proces doorlopen. We hebben in de voorbeelden die volgen steeds ondersteld dat geen enkele geraadpleegde DNS server op de hoogte was van de gevraagde mapping. In de praktijk wordt vaak gecached in alle DNS servers. → sneller beantwoorden van queries en netwerk minder belasten.

DNS kan ook gebruikt worden als **load balancing**, deze ene keer wordt het IP adres van server 1 gegeven, de andere keer die van server 2 (cache verloop tijd is hier minuten ipv dagen).

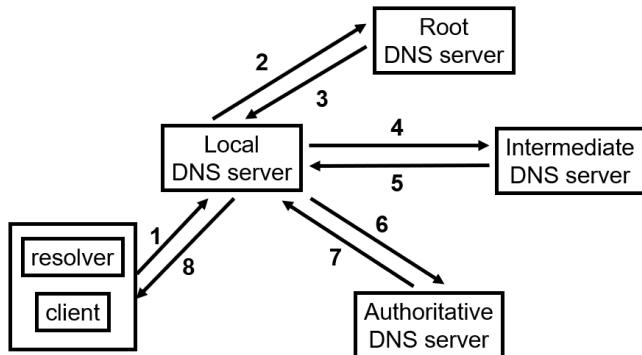
Recursief:

Bij recursief zal de client zijn lokale DNS server aanspreken, bij een lege cache stuurt deze zijn request door naar een root server. De root server stuurt op zijn beurt het request door naar een tussenliggende DNS server (typisch een top-level domein server). Dit wordt herhaald tot het aankomt bij een authoritative DNS server. Deze antwoordt met het IP adres en wordt het recursief teruggestuurd, tot het terechtkomt bij de resolver.



Iteratief:

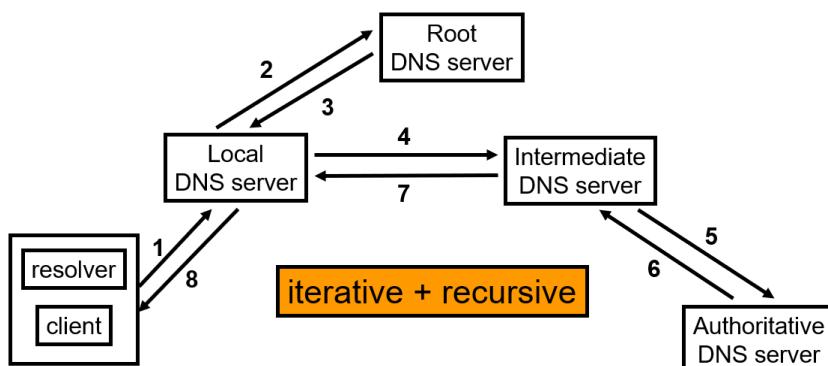
Query wordt niet doorgestuurd als een DNS server deze request niet kan beantwoorden. Het IP adres wordt gegeven van een DNS server die je mogelijk wel het juiste IP adres zou kunnen geven. Op die manier worden tussenliggende, in het bijzonder root servers, niet langer belast met oproepen die ze niet zelf kunnen beantwoorden.



iterative

In de praktijk

Er wordt vaak een soort hybride proces gevuld, waarbij voornamelijk de **root servers** ontlast worden aan de hand van een **iteratief** proces. De andere tussenliggende DNS servers de vraag wel doorsturen en die **recursief** beantwoorden.



DNS resolver

Het is een **lokaal programma** dat een mapping request stuurt om het bijbehorende IP adres bij een domeinnaam te achterhalen. Typisch gaat een DNS resolver een DNS request **doorsturen** naar een **Local Name Server** en neemt zelf geen deel in het proces, daarom wordt dit stuk software ook wel een '**stub resolver**' genoemd.

Het maakt deel uit van het **OS** en voorziet een **API** voor andere programma's. Hoe de resolver precies werkt is **afhankelijk van het besturingssysteem**.

Op Linux zijn alle DNS resolver configuraties terug te vinden in **/etc/resolv.conf**.

Alle configuraties zijn hier in IPv4, maar kunnen even goed voor IPv6 gebruikt worden.

Voorbeeld /etc/resolv.conf:

```
nameserver 157.193.40.24  
nameserver 157.193.40.42  
domain ugent.be  
options rotate
```

→ Twee nameservers om uit te kiezen. Als je een eigen nameserver wilt hosten moet 'localhost' worden toegevoegd.

→ Optie '**rotate**': bij iedere query zal de andere nameserver gecontacteerd worden

→ **domain 'ugent.be'**: unqualified names (bij queries) zullen steeds vervolledigd worden met 'ugent.be'. Bv. 'yeet' → 'yeet.ugent.be'

Voorbeeld /etc/hosts:

Dit kan worden gebruikt om zonder DNS toch hostnames te resolven (statisch).

```
127.0.0.1      localhost  
157.193.43.5    ugent.be  
157.193.238.45 mail xmail.ugent.be
```

Commando's DNS resolver

host → type a record en mogelijks nog meer

nslookup → meer opties en interactief mogelijk

dig → beste optie

Voorbeeld dig:

Goed om mee te experimenteren, zeker als je de 'trace' flag aanzet.

```

.% dig ugent.be any
;; Truncated, retrying in TCP mode.

; <>> DiG 9.10.6 <>> ugent.be any      Gebruikte header velden
; global options: +cmd
; Got answer:
; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 44719
; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL:
1

; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512      Query/request details
; QUESTION SECTION:
;ugent.be. IN ANY

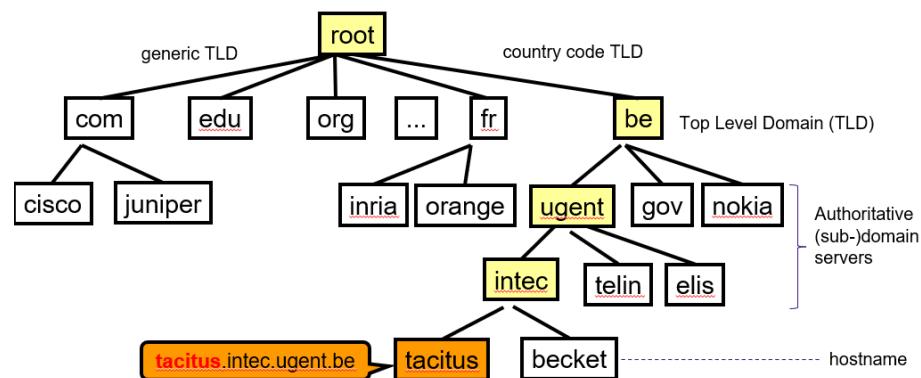
;ANSWER SECTION:
ugent.be. 24466 IN NS ugdns1.ugent.be.
ugent.be. 24466 IN NS ugdns2.ugent.be
ugent.be. 25799 IN A 157.193.43.50      Gevonden records voor het domain
ugent.be. 24466 IN NS ns.belnet.be.
ugent.be. 24466 IN NS ugdns3.ugent.be.

; Query time: 36 msec
; SERVER: 10.8.1.1#53(10.8.1.1)
; WHEN: Fri Feb 12 11:02:46 CET 2021
; MSG SIZE rcvd: 140      Response data inclusief
                           local name server

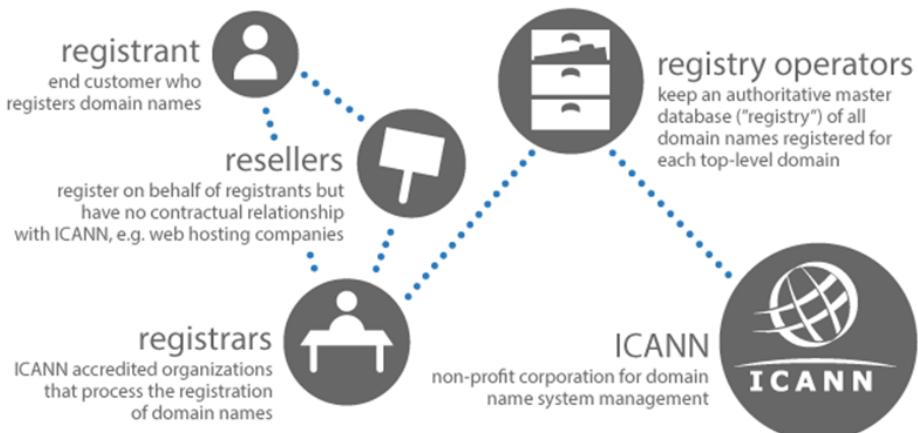
```

DNS registratie

Hoe zorgen we ervoor dat we in deze hiërarchie kunnen gebruik maken van een domeinnaam?



→ TLD domain name servers bevatten pointers naar authoritative (sub-)domain name servers, zodat DNS werkt wanneer men start bij een root DNS server.



→ resellers hebben als nadeel dat ze geld kunnen vragen voor bijvoorbeeld het domein 'corona.be'.

Registry operators: deze zijn verantwoordelijk voor alle domeinnamen die vallen onder een top-level domein zoals '.be'. Die lijst van alle gebruikte domeinnamen onder zo'n top-level domein wordt de **registry** genoemd.

Generieke top-level domeinen zoals '.com' of '.org' vallen onder de **ICANN organisatie**. Zij voeren een globaal beleid over welke domeinnamen wel of niet gebruikt mogen worden. Beheer van country codes zoals '.be' **verschilt van land tot land**.

Registrars: om te voorkomen dat registry operators een bottleneck vormen bij domeinnaam aanvragen, maakt men gebruik van **registrars**. Dit zijn bedrijven die **toegang krijgen tot de registry** om aanvragen af te handelen.

Registrant: de **aanvragers** zelf van domeinnamen.

Verificatie tijdens registratie

De data die bij zo'n registratie wordt bijgehouden is voornamelijk **administratief** en **juridisch**, eerder dan een technische noodzaak voor het DNS systeem. Welke data wordt bijgehouden is afhankelijk van de specifieke registry. Meestal identificatie zoals naam, email, ... maar ook een status of het eventueel wordt overgedragen naar een andere eigenaar.

EPP (Extensible Provisioning Protocol): voorkomt dat meerdere registrars **hetzelfde domein** op hetzelfde moment registreren bij een registry.

RDAP protocol

Registry operators maken een deel van de domeinregistratie gegevens beschikbaar via een **RDAP protocol**, ook wel '**whois**' genoemd (voorloper).

Voorbeeld domein registreren:

Eerste stap is het domein registreren bij een **DNS registrar**, met als gegevens de **namen** en **IP adressen** van de **authoritative name servers** die hiervoor verantwoordelijk zullen zijn. Ideaal geef je een primaire en secundaire authoritative server op (redundantie). Hierna zal de **registrar**, die verantwoordelijk is voor het top level van het domein '.com', vier entries invoegen.

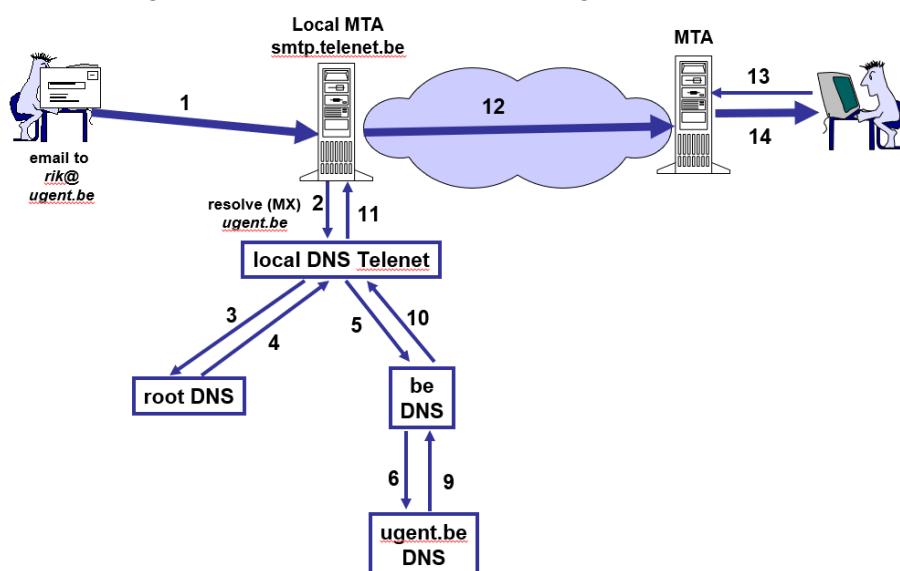
Twee entries voor de nameservers die kunnen gecontacteerd worden, meestal aan de hand van hun hostnames. Ook twee type A entries om hun IP adressen te resolven ('waar staan deze servers'). Dit voorkomt dat de 'dns1.networkutopia.com' geresolved zou moeten worden door een andere DNS server. Daarom worden deze records ook wel **glue records** genoemd, omdat ze de connectie vormen tussen het domein dat geresolved moet worden en hun nameservers.

(**networkutopia.com**, **dns1.networkutopia.com**, NS)
(**dns1.networkutopia.com**, **212.212.212.1**, A)
(**networkutopia.com**, **dns2.networkutopia.com**, NS)
(**dns2.networkutopia.com**, **212.212.215.1**, A)

Nu moet nog de nameserver worden opgezet en geconfigureerd worden (zie verder).

Interactie tussen e-mail en DNS (voorbeeld)

De e-mail client maakt gebruik van het SMTP protocol om de mail af te leveren bij de lokale SMTP server van het netwerk (1). De local MTA (Message Transfer Agent) wenst de e-mail door te sturen naar de mailserver van de bestemming en inspecteert daarom het email adres van deze bestemming. De server probeert het gevonden domein te **resolven** aan de hand van zijn **local name server** (2). De local name server stuurt de request door naar een **root server** (3), die antwoord met het IP adres van de **top-level domain server** (4). De **local name server** gaat het request doorsturen naar de **top-level domain server** (5), die het op zijn beurt doorstuurt naar de **authoritative name server** (6). Deze laatste weet het correcte adres van de **mailserver** behorende tot het domein en antwoord deze terug (9). **Recursief** (10) komt het antwoord terug bij de **local name server** die het op zijn beurt aflevert bij de **local mailserver** (11). Nu het IP adres van de volgende mailserver bekend is, kan de e-mail doorgestuurd worden over het internet tot bij de mailserver van de bestemming (12), ook hiervoor wordt SMTP gebruikt.



Oefening: reconstructie aan de hand van een web e-mail client zoals Gmail.

Hier is de takeaway dat de client gewoon connecteert met een HTTP server, maar deze server dan een connectie maakt met de mail server. Dus niet de client met de mailserver.

DNS server

DNS servers / nameservers worden geconfigureerd aan de hand van zone-files.

Bij het opzetten van een DNS server moet de keuze worden gemaakt tussen een **caching DNS server** en een **Authoritative server**. De twee zouden gecombineerd kunnen worden, maar dit wordt in de praktijk afgeraden omwille van beveiligingsmaatregelen.

Caching nameserver: enkel en alleen om netwerkgebruikers een **betere netwerkkwaliteit** aan te bieden. Deze server is dan vaak de local name server die eindgebruikers in je netwerk instellen, bijvoorbeeld via DHCP.

→ sneller afhandelen van **meerdere requests voor dezelfde domeinnaam**, minder DNS verkeer noodzakelijk over het internet.

Caching duur van opgevraagde resource records wordt bepaald door de **TTL**.

Authoritative server: wanneer je als instelling eigenaar bent van een domein die is aangevraagd via de registratieprocedure, dan moet je ook twee Authoritative Name Servers opzetten die in staat zijn DNS query's te beantwoorden voor domeinnamen die onder jouw verantwoordelijkheid vallen. Gaan normaal geen query's kunnen beantwoorden waarvoor ze niet verantwoordelijk zijn. Ze gaan de queries dus niet recursief beantwoorden, maar wel een IP adres teruggeven van een DNS server die dat wel kan (iteratief).

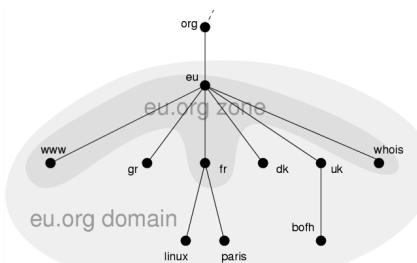
DNS werkt op het principe van hiërarchisch delegeren. Root servers delegeren request naar een top-level domain server. Top-level domain servers delegeren naar authoritative DNS servers.

Een delegation is dus een administratieve grens waar de verantwoordelijkheid van een gegeven domein wordt doorgegeven aan een andere Name server.

Domain/zone:

Met een **domein** verwijzen we naar een **subtree** binnen de hiërarchie.

Met een **zone** verwijzen we naar een deel binnen zo'n subtree dat **beheert** wordt door **eenzelfde DNS server**. De RR (Resource Records) voor een zone worden opgeslagen in zone files.



DNS server basics:

De daemon 'named' luistert op UDP en TCP poort 53. Binnenkomende **DNS requests** gaan via **UDP**, **zone transfers** via **TCP**.

Een zone transfer is het proces waarin een **primaire** DNS server een **zone file** kopieert naar een **secundaire** DNS server.

Voorbeeld: running a caching nameserver:

De server zal alle requests forwarden, maar cached de resultaten.

De configuratie gebeurt in **/etc/named.conf**. Hier zijn er twee delen '**options**' en **de zones**.

Per zone moet een pointer naar een zonebestand gegeven worden die resource records bevat voor die zone. Hier is de enige zone die nodig is die van de **root** (aangeduid met een punt).

```

options {
    recursion yes;
    // other
}

zone "." {
    type hint;
    file "db.cache";
}

```

Voorbeeld root hints bestand:

Zelf downloaden van internet en invullen.

```

.
3600000      NS   A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET. 3600000      A    198.41.0.4
A.ROOT-SERVERS.NET. 3600000      AAAA  2001:503:ba3e::2:30
;
; FORMERLY NS1.ISI.EDU
;
.
3600000      NS   B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET. 3600000      A    199.9.14.201
B.ROOT-SERVERS.NET. 3600000      AAAA  2001:500:200::b

```

→ Voor het domein root '.' hebben we verschillende root servers.

→ Daaronder A en AAAA type record met bijbehorende IP adressen voor de root DNS servers. Voor elke query waarbij er geen data in de cache aanwezig is, zal een van die root servers gecontacteerd worden.

Access control list

Without any extra measures, your DNS servers will be attacked (DNS amplification attack)

Add the following to **/etc/named.conf**

```

acl goodclients {
    192.0.2.0/24;
    localhost;
    // other
};

options {
    // other options
    allow-query { goodclients; };
}

```

Voorbeeld: draaien van een (primaire) authoritative name server:

De authoritative name server is verantwoordelijk voor een aantal DNS zones.

Configuratie in **/etc/named.conf**

Wanneer je verantwoordelijk bent voor een DNS domein, moet je minstens 2 authoritative DNS servers opzetten: een primaire (master) en een secundaire (slave).

```

options { // see earlier };

zone "ugent.be" in {
    type master;           // primary name server
    file "db.ugent.be";   // zone file for the domain
};

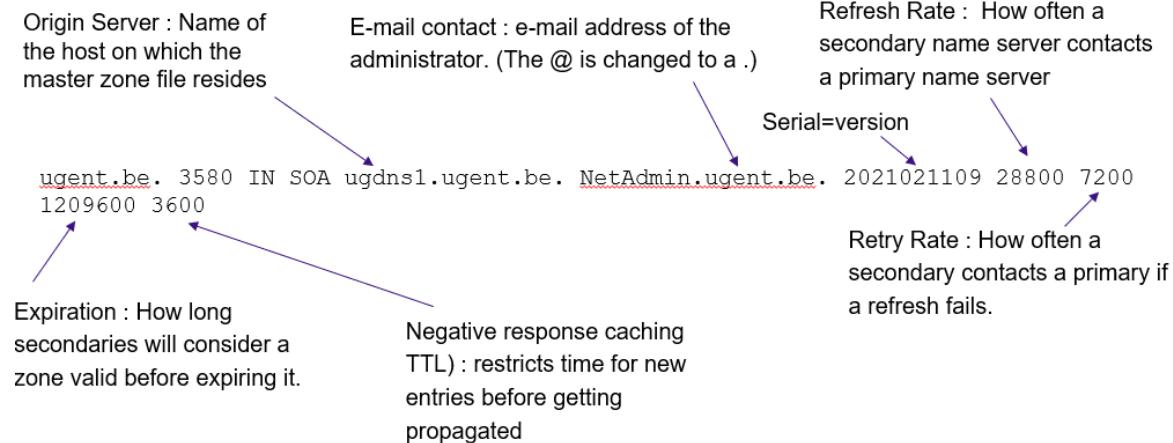
zone "193.157.in-addr.arpa" in {
    type master;           // reverse DNS zone
    file "db.157.193";   };
};

zone "..." { }; // info for other zones
zone "." in { // see earlier };

```

Zone file van een authoritative server:

Een **SOA (Start Of Authority) record**, dit definieert de standaard eigenschappen voor het hele domein waarvoor we verantwoordelijk zijn.



Opmerkingen zone files:

Een '.' punt op het einde van een domeinnaam in een zone file, refereert dat het een Fully Qualified Domain Name (FQDN) is. Zonder dat wordt de domeinnaam van de zone (=origin) geprepend aan alle domeinnamen (relatief).

- *mail.ugent.be. 20035 IN CNAME xmail.ugent.be.*
- *mail 20035 IN CNAME xmail.ugent.be.*

Wanneer in een resource record de **domeinnaam** niet wordt vermeld, dan wordt de naam van het **voorgaande record** gebruikt.

→ zelfde naam voor meerdere servers, **load balancing**.

DNS security Extensions:

Vastgelegd in RFCs, voornaamste doel is om toe te laten dat de **bron van zone data geauthenticeerd** kan worden. Op die manier kan niet iedereen zomaar claimen de authoritative DNS server van een domein te zijn. Daarnaast zijn er ook mechanismen om de **integriteit van DNS data** te verifiëren.

Richtlijnen DNS server architectuur:

Authoritative server nooit alleen opzetten, altijd met een **master** en een **slave**, liefst meerdere van elk. Master heeft schrijfrechten naar de zone files en de slaves kopiëren deze.

Een DMZ (Demilitarized Zone): een netwerksegment die zich typisch bevindt tussen interne en externe deel van het netwerk, meestal met een firewall.

Reverse DNS

Het omgekeerde proces, bij een gegeven IP adres wil je achterhalen wat de bijbehorende domeinnaam is. Dit is onder andere nuttig op vlak van **beveiliging**. Het laat toe om te controleren dat e-mails niet toekomen van spammende bots, snel opgezette tijdelijke machines krijgen meestal geen domeinnaam toegekend.

Verder is het handig voor **troubleshooting**, zoals bij 'traceroute' via welke tussenstations en via welke eigenaars je pakketten passeert.

Het maakt **geen** gebruik van bestaande DNS databases met een reverse lookup. Hiervoor is het niet ontworpen (geen index), dus zouden alle nameservers overlopen moeten worden om een match te vinden.

Daarom maakt het gebruik van een **PTR** resource record.

Om een reverse DNS query te doen werd een nieuw top-level domein aangemaakt, 'arpa'.

Het bevat een subdomein 'in-addr' voor IPv4 en 'IP6' voor IPv6, om adressen te resolven.

Het reverse DNS mechanische vereist dat je een **resource record** toevoegt **voor een IP adres**, net zoals je dat normaal zou doen voor een domeinnaam. Op die manier kan een reverse DNS query hetzelfde iteratieve of recursieve proces volgen als een normale DNS query.

Zo is er ook een **hiërarchische structuur**. De eerste octetten behoren meestal toe aan de regional internet registries (**RIR**). Daaronder de ISP's of de universiteiten. Dit is makkelijk als de octet grenzen worden gerespecteerd. Er wordt echter vaak gebruik gemaakt van CIDR, wat die grenzen niet respecteert.

Reverse zones in BIND DNS

SOA en NS records zijn hetzelfde als bij een normale zone. Er moet wel nog een **extra PTR record** worden toegevoegd. Om toe te laten dat jouw verantwoordelijke name server daadwerkelijk gecontacteerd wordt bij een query voor een IP adres met bv. een prefix 157.193, moet je contact opnemen met de beheerder van het bovenliggende domein. Zo neemt die beheerder het IP adres van jouw DNS server op in zijn database.

14.49.193.157.in-addr.arpa. 32400 IN PTR cedar.ugent.be.

Commando reverse DNS lookup:

dig -x

```
% dig -x 157.193.49.14

; <>> _DIG_ 9.10.6 <>> -x 157.193.49.14
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49646
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;14.49.193.157.in-addr.arpa. modified IP address into a domain name

;; ANSWER SECTION:
14.49.193.157.in-addr.arpa. 32400 IN PTR cedar.ugent.be.

;; Query time: 36 msec
;; SERVER: 10.8.1.1#53(10.8.1.1)

```

Found entry and domain name

DNS importance & security

Internet DNS server statistics

Een enorm groot deel van de DNS records wordt bijgehouden door een beperkt aantal servers. **Gebruik meer dan 1 DNS server** en probeer deze secundaire nameserver(s) in-house te houden.

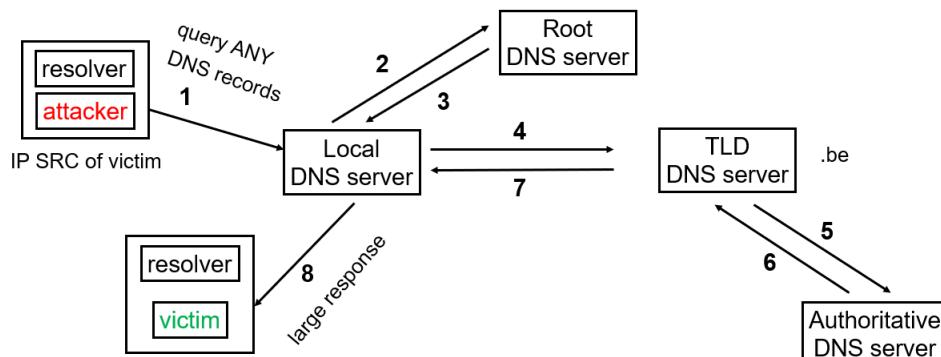
Status = NXDOMAIN

Status **NXDOMAIN** ofwel status code 3, betekent dat de gezochte naam niet resolved kan worden.

Dit kan misbruikt worden door de ISP's door het IP adres van een eigen versie te retourneren wanneer ze een NXDOMAIN onderscheppen, waarna ze ads tonen en gegevens achterhalen. → **DNS hijacking**.

Chrome verhielp dit door random queries te sturen op startup en zo deze hijacking te detecteren.

Open name servers & dns amplification attacks



Kleine DNS query die **heel veel DNS verkeer** teweegbrengt bij het slachtoffer.

→ **Toegang tot nameserver beperken via Access Control List (ACL)**.

De local name server moet werken met de ACL, waardoor er controle gebeurt van wie de queries juist komen. Om zo queries te weigeren van een IP adres die van een heel ander subnet komt dan bijvoorbeeld het subnet die je zelf beheert als ISP. Op die manier kan een

attacker zich niet meer voordoen als een bepaald slachtoffer, omdat het verkeer afkomstig is van een extern netwerk, waardoor de local name server nooit zal reageren op een DNS query die van een remote netwerk lijkt af te komen.

Zelftest DNS

Vraag 4: "dig +trace @8.8.8.8 idlab.ugent.be A"

a) welke DNS server was het eerste aanspreekpunt (geef het IP adres)

Vrij obvious, met de '@' duid je de nameserver aan die je wenst te gebruiken.

b) welke DNS server heeft uiteindelijk het IP adres van idlab.ugent.be bezorgd (geef het IP adres)

c) wat is het IP adres van de gezochte server?

d) hoeveel tussenliggende DNS servers waren er nodig om dit antwoord te komen (incl. je eerste aanspreekpunt)

e) hoe kan de command line tool "dig" de trace van tussenliggende DNS servers bekomen:

a.d.h.v. een "iteratieve DNS query" of a.d.h.v. een "recursieve DNS query"?

```
> dig +trace @8.8.8.8 idlab.ugent.be A
; <<>> DIG 9.10.6 <<>> +trace @8.8.8.8 idlab.ugent.be A
; (1 server found)
;; global options: +cmd
. 86656 IN NS a.root-servers.net
. 86656 IN NS b.root-servers.net
. 86656 IN NS c.root-servers.net
. 86656 IN NS d.root-servers.net
. 86656 IN NS e.root-servers.net
. 86656 IN NS f.root-servers.net
. 86656 IN NS g.root-servers.net
. 86656 IN NS h.root-servers.net
. 86656 IN NS i.root-servers.net
. 86656 IN NS j.root-servers.net
. 86656 IN NS k.root-servers.net
. 86656 IN NS l.root-servers.net
. 86656 IN NS m.root-servers.net
;; Received 525 bytes from 8.8.8.8#53(8.8.8.8) in 50 ms
be. 172800 IN NS a.ns.dns.be
be. 172800 IN NS b.ns.dns.be
be. 172800 IN NS c.ns.dns.be
be. 172800 IN NS y.ns.dns.be
be. 172800 IN NS b.ns.dns.be
be. 172800 IN NS z.nsset.be
;; Received 871 bytes from 192.36.148.17#53(i.root-servers.net) in 15 ms
ugent.be. 86400 IN NS ugdns3.ugent.be
ugent.be. 86400 IN NS ns1.belnet.be
ugent.be. 86400 IN NS ns2.belnet.be
ugent.be. 86400 IN NS ns3.belnet.be
ugent.be. 86400 IN NS ugdns2.ugent.be
;; Received 801 bytes from 194.0.43.1#53(c.ns.dns.be) in 179 ms
idlab.ugent.be. 86400 IN NS dns2.idlab.ugent.be
idlab.ugent.be. 86400 IN NS dns1.idlab.ugent.be
idlab.ugent.be. 86400 IN NS ugdns1.ugent.be
;; Received 150 bytes from 193.190.198.14#53(ns1.belnet.be) in 14 ms
idlab.ugent.be. 86400 IN A 157.193.43.50
idlab.ugent.be. 86400 IN NS dns2.idlab.ugent.be
idlab.ugent.be. 86400 IN NS ugdns1.ugent.be
idlab.ugent.be. 86400 IN NS dns1.idlab.ugent.be
;; Received 150 bytes from 157.193.214.4#53(dns2.idlab.ugent.be) in 18 ms
```

[DNSSEC entries zijn eruit geknipt]

a) Eerste aanspreekpunt = local name server

b) DNS server gaf antwoord

c) IP adres van gezochte server

d) 5 intermediate DNS servers

e) Via iteratieve query, kan telkens IP adres van tussenliggende DNS servers gevonden worden

Hoofdstuk 4: Netwerklaag - datalevel

Sectie 4.3.3 IPv4 addressing

Hoe werken IP adressen

Een **interface** is een connectie tussen een host/router en een fysieke link.

IPv4 addressing, speciale netwerken

Eerste adres (0) van een subnet is **netwerkadres**, om het netwerk te identificeren. Het laatste adres (255) is om te **broadcasten**.

Het **0.X.Y.Z/8** adres identificeert de **host** op het netwerk (gebruikt voor booten). Enkel toegelaten als source adres.

127.X.Y.Z is de **loopback Interface** (debugging), vooral 127.0.0.1 wordt gebruikt.

169.254.0.0/16 is voor **link-local adressering**. Enkel geldig op link, niet routeerbaar. Enkel geldig als destination, geen forwarding toegelaten.

Private netwerken: **10.X.Y.Z; 172.16.X.Y; 192.168.X.Y.**

Efficiënt gebruik (opdelen) van IP adresblokken

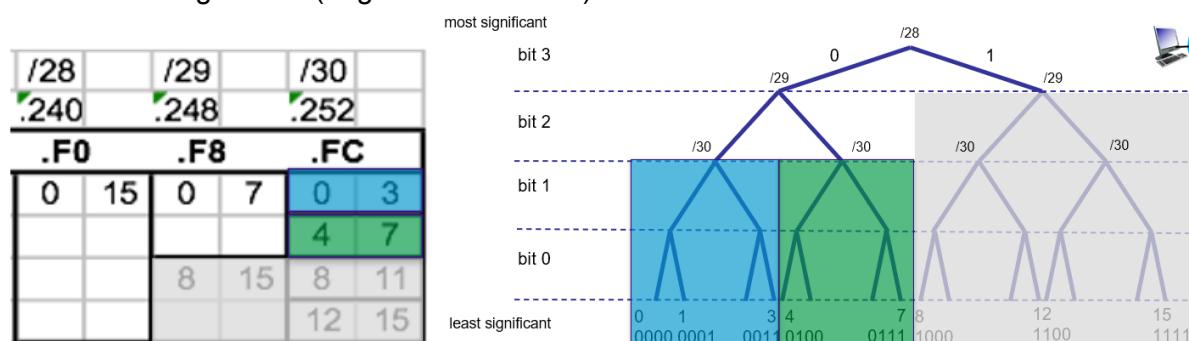
/28 voorbeeld

Optie 1, alles adressen, aantal beschikbare adressen: $2^{(32-28)} - 2 = 2^4 - 2 = 14$.

Optie 2, 2 /29 adresblokken, aantal beschikbare adressen: $2^{(32-29)} - 2 = 2^3 - 2 = 6$

Optie 3, 4 /30 adresblokken, aantal beschikbare adressen: $2^{(32-30)} - 2 = 2^2 - 2 = 2$

In de praktijk zijn er vaak subnetten die niet dezelfde grootte hebben / niet hetzelfde aantal interfaces. **Per subnet** bepalen hoeveel IP adressen nodig. Stel dat er een subnet is met 2 interfaces en 1 met 6 interfaces. Dan gaan we op zoek naar de kleinst mogelijke adresblokken om elke interface in de gegeven subnetten een adres te kunnen geven. Voor die met 2 interfaces is er een /30 adresblok nodig. Het eerste beschikbare adresblok is helemaal links en begint op 0. Voor een subnet met 5 interfaces hebben we minimaal een adresblok met /29 nodig (1 adres over). In de boomstructuur is nu duidelijk welke adresruimte nog over is (nog 1 /30 adresblok).



Forwarding in een IP router

Subnetten zijn interfaces die met elkaar direct kunnen communiceren via de **Link Layer**. **Routers** verbinden **verschillende subnetten** met elkaar. De interface van de **router** die toegang geeft tot het **subnet** van de **bestemming** is de **gateway interface**. Er is geen plaats in de IP header om het adres van router er in op te nemen, er is enkel plaats voor de source en destination adressen. Om de **gateway** te bereiken kan de afzender gebruik maken van de onderliggende **Link Layer**, die laat toe om alle rechtstreekse verbonden toestellen met elkaar te laten communiceren. De **Link Layer** zal een **extra header** toevoegen zodat het naar de **gateway** gestuurd kan worden **binnen hetzelfde subnet**. De router gaat dan het pakket opslaan in zijn buffer, vervolgens gaat die de IP header parsen en analyseren om te zien welke bestemming het pakket heeft. Aan de hand van die bestemming zal hij een lookup doen in zijn forwarding tabel en de interface kiezen die bij het subnet van de bestemming hoort. Vervolgens wordt het pakket doorgestuurd naar een gegeven uitgaande interface die het naar de volgende hop stuurt, wat mogelijks weer een gateway is.

Bijkomende handelingen IP router

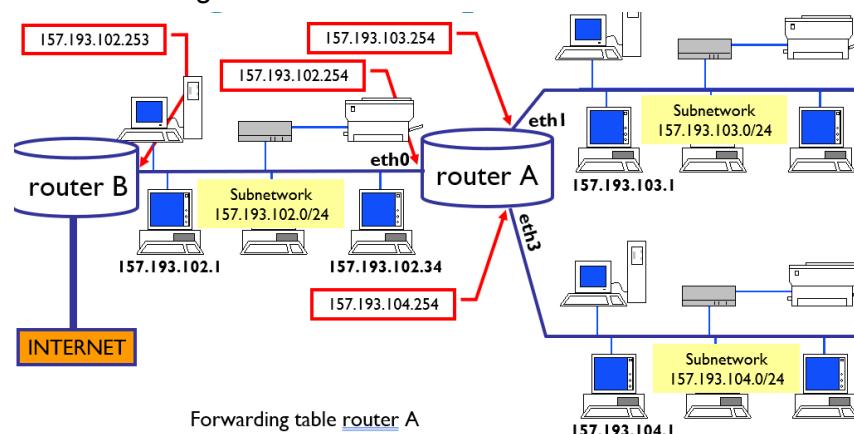
Eerst doet de router een aantal **sanity checks**, waaronder het **versienummer** controleren en een **checksum** om de header te controleren om er zeker van te zijn dat er geen header velden corrupt zijn. De tweede check zal de **TTL decrementeren**, indien de TTL 0 wordt dropt hij het pakket. Dit **voorkomt netwerk loops**.

De lookup in de forwarding table maakt gebruik van **longest prefix matching**. De router zal steeds de entry kiezen waarbij de prefix van de bestemming **maximaal** overeenkomt met een entry in de forwarding tabel.

Bij het uitsturen van het pakket wordt een uitgaande link geselecteerd. De **header checksum** wordt opnieuw berekend en toegevoegd aan het pakket. Dit komt omdat de TTL en fragmentatie vlag mogelijks de inhoud van de header hebben aangepast. De MTU van de uitgaande link zal bepalen of het pakket moet gefragmenteerd worden.

Voorbeeld IP forwarding

'eth0, eth1, ...' zijn veelgebruikte namen voor router interfaces, het geeft aan dat de Link Layer gebruik maakt van Ethernet. Vaak worden die het grootst mogelijke IP adres binnen dat subnet toegekend.



Destination	Prefix length	Gateway	Interface	Interface
127.0.0.0	/8	127.0.0.1	lo0	127.0.0.1
0.0.0.0	/0	157.193.102.253	eth0	157.193.102.254
157.193.102.0	/24	0.0.0.0/*	eth0	157.193.102.254
157.193.103.0	/24	0.0.0.0/*	eth1	157.193.103.254
157.193.104.0	/24	0.0.0.0/*	eth3	157.193.104.254

→ In de tabel overlappen de laatste twee kolommen, beide kunnen gebruikt worden om een interface aan te spreken.

→ De eerste lijn is om te testen.

→ De tweede regel is de **default route entry**, matcht eerder welk adres aangezien de prefix 0 is. Deze lijn wordt gebruikt als geen enkel andere entry matched, aangezien dit de *longest matching prefix* is. Vaak naar het internet. In deze oefening wordt het gestuurd naar router B zodat die het naar het publieke internet kan sturen.

→ Vervolgens zien we drie entries in de forwarding tabel. 1 voor elk rechtstreeks verbonden subnet in router A.

Lokale routing tabel inspecteren

"Nuttige oefening"

netstat -rn

→ r: routing table

→ n: name resolution, IP adres gebruiken

Destination	Gateway	Flags	etif
default	192.168.1.1	UGScl	en0
127	127.0.0.1	UCS	lo0
127.0.0.1	127.0.0.1	UH	lo0
169.254	link#4	UCSI	en0
192.168.1	link#4	UCSI	en0

→ Flags:

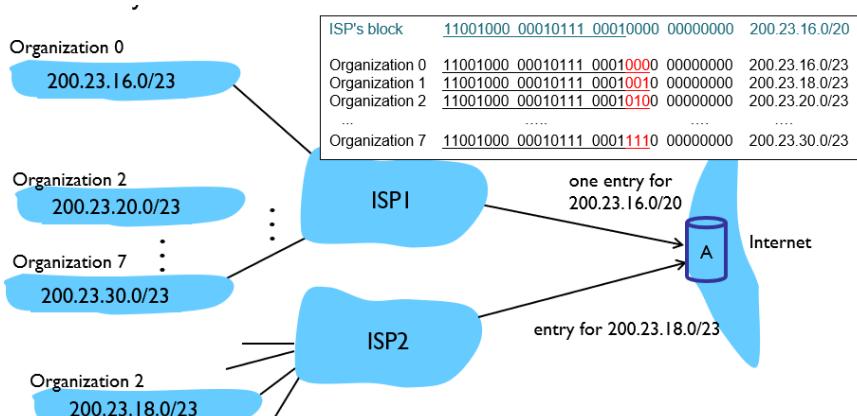
→ U: route is up

→ G: gateway zal gebruikt worden bij route

→ H: directe verbinding met host

Aggregatie van IPv4 adressen

Bij grote routers (die van het internet) bevatten de lookup tables enorm veel entries. Om het aantal entries in die tabellen minimaal te houden, maken we gebruik van **adres aggregatie** die mogelijk is door de hiërarchische manier waarop IP adressen zijn georganiseerd.

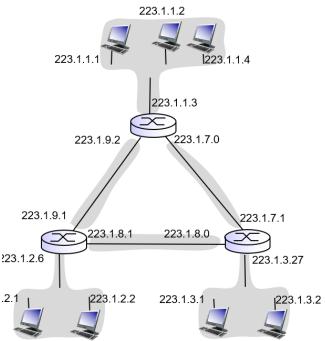


Router A zal enkel entries bevatten per ISP block. Er wordt gebruik gemaakt van ***longest prefix matching*** om specifieke route toe te voegen.

Response college

Subnets tellen

Routers kunnen enkel verbonden worden door apart subnet. Op figuur aparte grijze zones tellen.



Hoeveel hosts zijn er in het netwerk?

157.193.122.96 subnetmask: FF.FF.FF.E0

E0 = 14 0 = 1110 0000 → $2^5 = 32$ → -2 voor 255 en 0 = 30 hosts

De -2 voor **netwerkadres** en **broadcastadres**

157.193.122.240 is a host in a network with 62 hosts

What is the (sub)networkaddress and the subnetmask ?

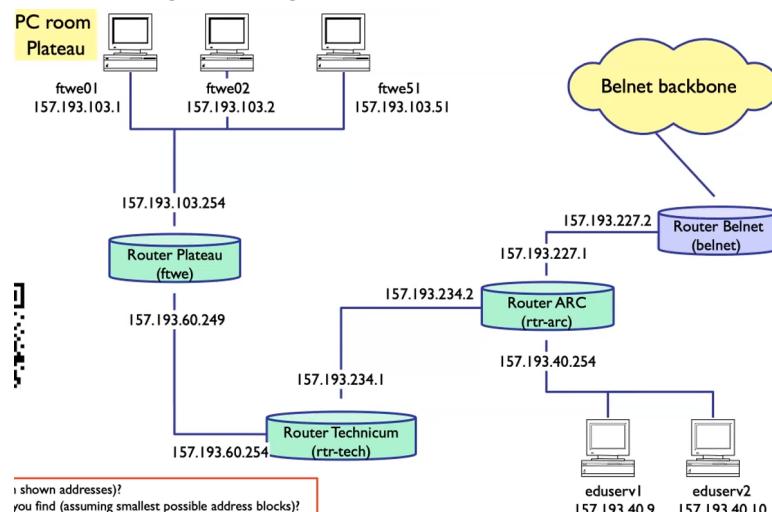
62 hosts → $\lg(62 + 2) = 6$

$32 - 6 = 26 \rightarrow /26$

6 nullen → subnetmask: FF:FF:FF:C0 , 255:255:255:192

Netwerk adres: subnet mask **bitwise AND** host address: 157.193.122.192

IP subnetting oefening

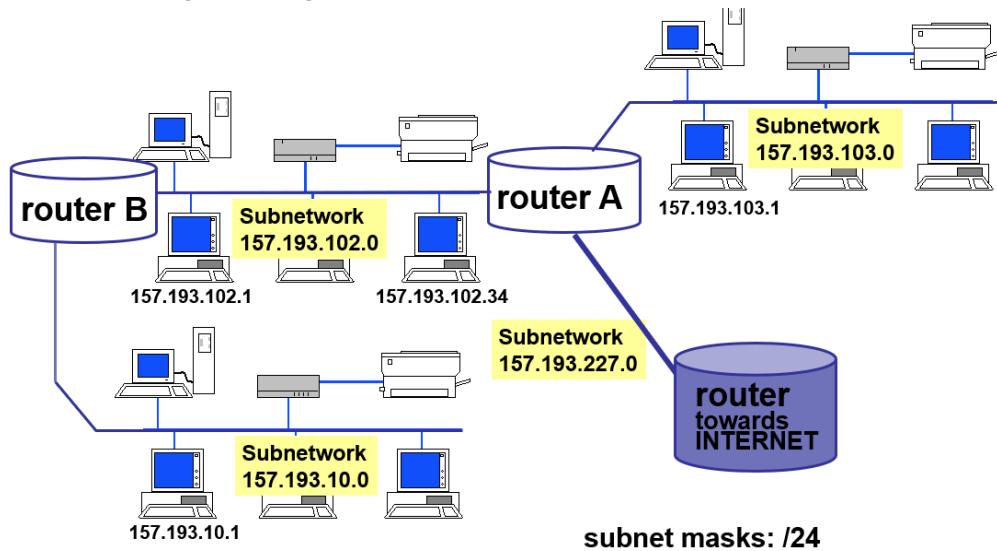


- Hoeveel subnets
- Prefix lengtes (kleinste adresruimte)
- Kleinste subnetten en geassocieerde prefixen
- Aantal adressen per subnet

- 5
- +c)

1 → 254:	$32-8 = 157.193.103.0/24$	→ 254 adressen
249 → 254:	$32-3 = 157.193.60.248/29$	→ 6 adressen
1 → 2:	$32-2 = 157.193.234.0/30$	→ 2 adressen
9 → 254:	$32-8 = 157.193.40.0/24$	→ 265 adressen
1 → 2:	$32-2 = 157.193.227.0/30$	→ 2 adressen

Zelftest routing oefening



- a) Hoeveel interfaces moeten nog een IP adres krijgen zodat een routing tabel kan worden opgesteld?

De gateways om de bestemmingen te bereiken, die van router B. Dan die van de router naar het internet. Vervolgens ook IP adressen van router A, om verkeer van de interface naar de uitgaande bestemming te sturen. → $2 + 3 = 5$

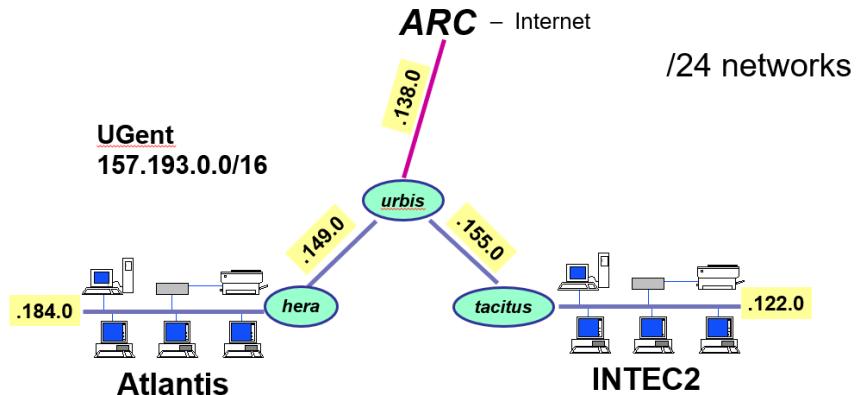
- b) Hoeveel entries in routing tabel?

Zeker 1 entry voor ieder geconnecteerd netwerk, aka 3. Voor subnetwerk 157.193.10.0 ook een entry toevoegen. Ook nog een default route.

→ 5 routing entries

	Destination	Subnet Mask	Gateway	Interface
Direct connected	157.193.102.0	/24	-	157.193.102.254
	157.193.103.0	/24	-	157.193.103.254
	157.193.227.0	/24	-	157.193.227.254
Networks, not in default Always: - "default" entry	157.193.10.0	/24	157.193.102.253	157.193.102.254
	0.0.0.0	/0	157.193.227.253	157.193.227.254

Oefening A



a) Routing tabel bepalen URBIS

Destination	Mask	Gateway	Interface
.138.0	/24	-	.138.254
.149.0	/24	-	.149.254
.155.0	/24	-	.155.254
.122.0	/24	.155.253	.155.254
.184.0	/24	.149.253	.149.254
default	/0	.138.253	.138.254

Geef .149.0 en .155.0 vrij voor andere doeleinden.

Herverdeel 122.0/24 adresblok zodat:

1. Grootst mogelijk (sub)adresblok kan gebruikt worden voor INTEC2 subnet
2. Subnetten tussen urbis, hera en tacitus nieuwe adressen krijgen
3. Bepaal nieuwe routing table van Urbis

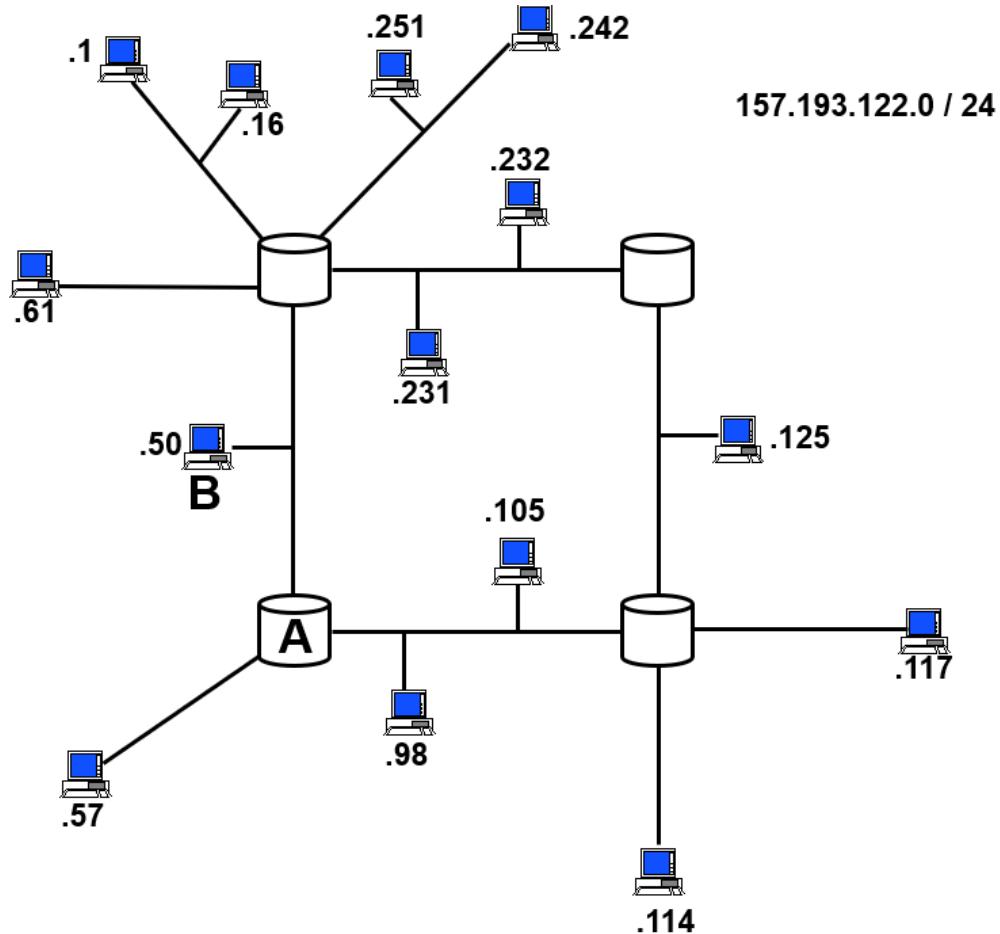
Destination	Mask	Gateway	Interface
.138.0	/24	-	.138.254
.122.128	/30	-	.122.129
.122.132	/30	-	.122.133
.122.0	/25	.122.134	.122.133
.184.0	/24	.122.130	.122.129
default	/0	.138.253	.138.254

Bepaal vrije, ongebruikte adresblokken

Free subnets :

.122.136 /29
.122.144 /28
.122.160 /27
.122.192 /26

Oefening IP



- Bepaal netwerkadressen/prefixen (zo spaarzaam mogelijk)
- Ken aan alle routers interfaces IP adressen toe
- Neem IP adressen zo hoog mogelijk binnen het adresblok.
- Geef de vrije subnetten weer
- Routingstabellen A en B (zo klein mogelijk, gebruik kortste pad)

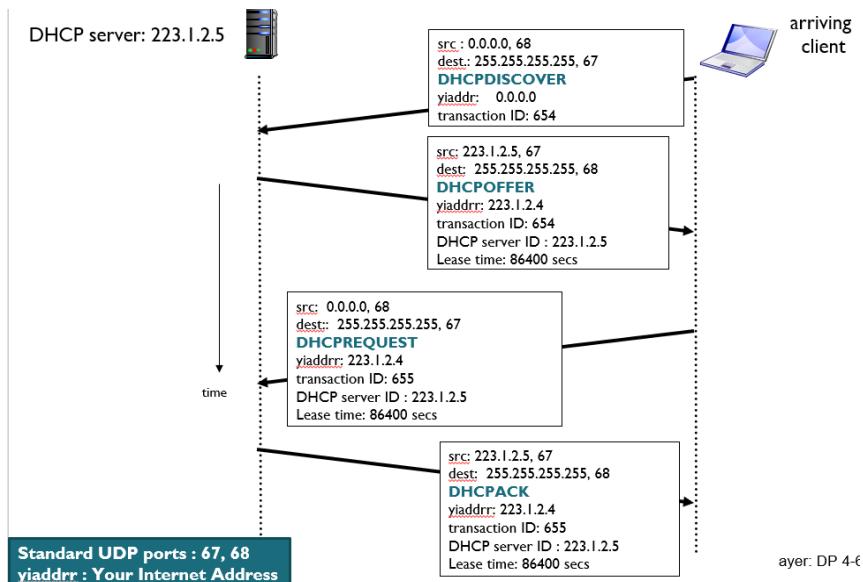
Sectie 4.3.3 IPv4 Dynamic Host Configuration Protocol

DHCP (Dynamic Host Configuration Protocol)

Er zijn twee manieren om hosts te voorzien van IP adressen.

De eerste is waarbij de systeembeheerder **manueel** de hosts gaat instellen.

De tweede is **dynamisch** en **plug-en-play** die een IP adres gaat uitdelen aan hosts wanneer de host het IP adres heeft van de DHCP server.



Hosts ontdekken de DHCP server door een **DHCP discover**. Ze doen een DHCP discover broadcast met **source IP-adres 0.0.0.0** en **destination 255.255.255.255** (broadcast adres). Het maakt gebruik van **UDP** en luistert op poort 67 aan de kant van de server en 68 aan de kant van de client. Ook wordt een **yiaddr** (Your Internet Address) veld 0.0.0.0 en een **random transaction ID** meegegeven. Het laatste is zodat de DHCP server verschillende requests uit elkaar kan houden.

De DHCP server houdt een lijst van beschikbare IP adressen bij en zal het eerste teruggeven met een **DHCP offer** en reserveren. Bovendien wordt het ook **gebroadcast** in **hetzelfde netwerk**. Dit request heeft **dezelfde transaction ID** als dat van de host. Het **IP adres** dat de DHCP server aanbiedt wordt meegegeven alsook de **verlooptijd** van het aanbod.

Omdat er meerdere offers kunnen gemaakt worden aan de DHCP host kiest de host een offer en antwoordt met een **DHCP Request**.

De server antwoordt hierop met een **DHCP ACK**.

Als de **lease time** verlopen is, moet de host een **nieuw IP adres** aanvragen. Hierdoor wordt DHCP ook wel een **soft-state** protocol genoemd. Het voordeel is dat adressen kunnen **hergebruikt** worden door andere hosts wanneer de lease niet verlengd wordt.

Broadcasting wordt gebruikt voor alle berichten. Dit laat enerzijds toe om de servers te ontdekken, maar het maakt het ook mogelijk dat andere hosts of DHCP server op de hoogte blijven van de reeds uitgestuurde offers, requests en ACKs.

Andere functionaliteiten DHCP

DHCP kan ook het IP adres van de default first-hop router / **default gateway**, **DNS server**, alsook de **network mask** / prefix lengte meesturen.

Autoconfiguratie link-local

Wanneer er geen DHCP server beschikbaar/bereikbaar is en dat het DHCP request tot niks leidt. In dat geval kan de DHCP client beslissen om een **link-local IP adres** te gebruiken **169.265/16** met random gegenereerde laatste twee octetten. Wanneer alle hosts op het netwerk dit zo doen, beschikken ze elk over een eigen IP adres in **hetzelfde subnet** en kunnen ze toch met elkaar communiceren.

Om te voorkomen dat de random gegenereerde adressen **reeds in gebruik** zijn, kan een **ARP request** over het netwerk gestuurd worden. Wanneer een host hierop antwoordt, moet een nieuw random IP adres gegenereerd worden.

Het gaat hier over link-local, dus ze kunnen **niet met hosts op andere subnetten** communiceren, die bevinden zich achter de router en deze zal geen forwarding instellen voor link-local adressen.

Tekortkomingen

Wanneer de DHCP server opnieuw online komt zullen de hosts een nieuw IP-adres worden toegewezen, maar zullen ze niet meer beschikbaar zijn op de **vorige link-local adressen**. Dit kan er voor zorgen dat applicatie hun communicatie stilvalt.

Een tweede aandachtspunt is dat **DHCP** in vele organisaties een **single-point-of-failure** is, wat het een goed doelwit maakt.

DHCP maakt gebruik van **broadcasting** in de **link-laag**, wat zorgt voor netwerk overhead. Hierdoor werkt IPv6 met andere technieken.

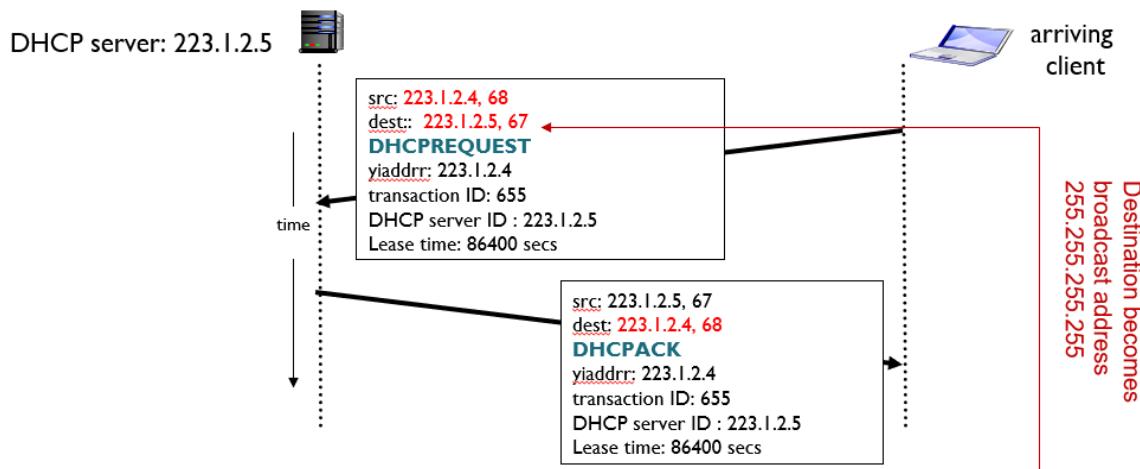
DHCP renewal & relay

De **initialisatie** gebeurt meestal wanneer de host opstart.

Wachten tot de lease verlopen is om dan een nieuwe DHCP request te doen zou betekenen dat alle communicatie verbroken wordt, omdat de host een nieuw IP adres krijgt toegekend. Het **DHCP renewal** proces laat toe de host zijn IP adres te laten vernieuwen zonder dat hij bestaande connecties hoeft te verliezen. Bovendien verloopt het proces **efficiënter** voor het netwerk, aangezien **discover** en **offer** messages worden overgeslagen en er **geen broadcast** berichten uitgewisseld worden.

Om te voorkomen dat een host zijn adres verliest, gaat deze een **DHCP Request** sturen **halverwege de leasetijd**. Als alles goed gaat zal de DHCP server hierop antwoorden met een **DHCP ACK** rechtstreeks naar de client. Hier zijn er maar **twee unicast berichten** nodig om de connectie te vernieuwen.

Mocht de DHCP server niet meer bereikbaar zijn, kan een andere DHCP server geprobeerd worden aan de hand van het **broadcast DHCP adres**. Mogelijks wordt dit proces herhaald met een exponentieel toenemende wachttijd, indien nodig. Als na drie pogingen nog geen antwoord wordt ontvangen, zal de client zijn IP adres moeten deactiveren.



DHCP in een ander subnet

Soms zijn subnetten te klein om een afzonderlijke DHCP server voor op te zetten.

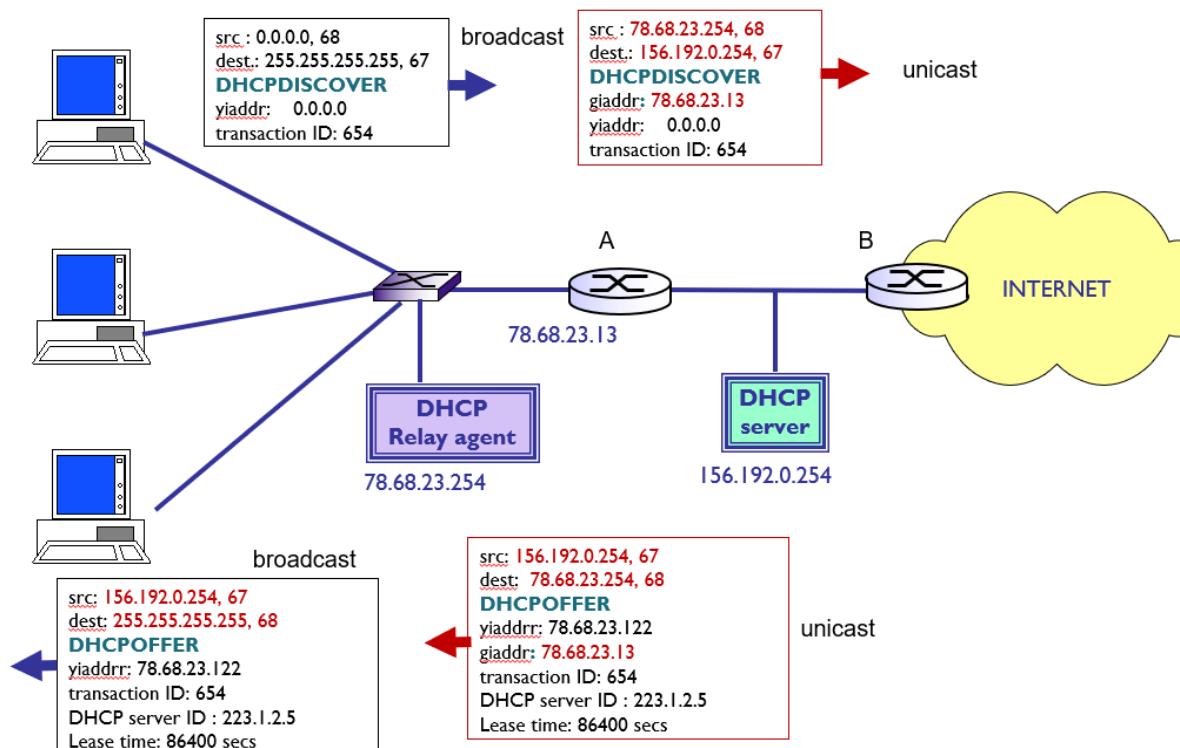
Broadcast berichten worden niet doorgestuurd via routers, er zijn twee manieren om dit probleem op te lossen.

De eerste oplossing is door een **nieuwe DHCP server** te installeren in het subnet, enkel en alleen voor de enkele hosts en de router interface. Dit is een verspilling van resources en vaak is het wenselijk alle DHCP activiteiten centraal te beheren vanuit 1 server.

Daarom kan men een DHCP relay agent in het netwerk introduceren. In het voorbeeld kan men bijvoorbeeld een **DHCP relay agent** connecteren als aparte server in het linker subnet. In de praktijk wordt de DHCP relay vaak geïntegreerd met de router. De enige functie van zo'n relay bestaat er uit om alle **broadcast berichten door te sturen** naar de DHCP server in een **ander subnet**. Naar welke server dit gaat moet geconfigureerd worden in de relay.

De relay gaat een DHCP **broadcast** onderscheppen en deze via een **unicast** doorsturen naar het andere subnet. Zo veranderen de destination en source adressen van het discover pakket en wordt het DHCP veld ingevuld met het adres van de relay agent in het **Gateway IP Adress (giaddr)** veld. Als dit pakket toekomt ziet de server dat dit niet gaat over een client in zijn eigen subnet, maar uit het **subnet van het gateway IP adres**. Zo kan hij makkelijk zien of hij een nieuw IP adres kan aanmaken binnen het bereik van dat subnet.

Zo ja wordt het IP adres aangeboden met een **unicast DHCP offer** naar de relay agent. Als het DHCP offer toekomt bij de relay agent, zal deze het **broadcasten** alsof het van een DHCP server afkomstig is. De daarop volgende DHCP berichten volgen hetzelfde proces, met de DHCP relay agent die DHCP broadcasts omzet naar unicast berichten naar een DHCP server in een ander subnet. **DHCP unicast berichten uit het andere subnet** worden omgezet naar **broadcast** berichten.



DHCP client

De DHCP client is de software op de host die verantwoordelijk is voor het versturen van DHCP berichten om een IP adres te verkrijgen.

Eenmaal het DHCP ACK bericht is ontvangen zal de **DHCP client** enkele velden aanpassen: het **IP adres** van de client, de **subnet mask**, de **default gateway** in de routeringstabell, de **DNS resolver** informatie (/etc/resolv.conf), de **naam** van de machine.

Commando's

sudo dhclient -d → DHCP client informatie

```
Listening on LPF/eth0/08:00:27:c2:be:11
Sending on  LPF/eth0/08:00:27:c2:be:11
Sending on  Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 3  (xid=0x56400537)
DHCPREQUEST of 10.0.2.15 on eth0 to 255.255.255.255 port 67  (xid=0x37054056)
DHCPoffer of 10.0.2.15 from 10.0.2.2
DHCPACK of 10.0.2.15 from 10.0.2.2
RTNETLINK answers: File exists
cp: EOF on /tmp/tmp.fEV0gezo6J which is empty
bound to 10.0.2.15 -- renewal in 36071 seconds.
```

cat /var/lib/dhcp/dhclient.leases → lease informatie

```
lease {
    interface "eth0";
    fixed-address 10.0.2.15;
    option subnet-mask 255.255.255.0;
    option dhcp-lease-time 86400;
    option routers 10.0.2.2;
    option dhcp-message-type 5;
    option dhcp-server-identifier 10.0.2.2;
    option domain-name-servers 10.0.2.3;
    option domain-name "telenet.be";
    renew 3 2021/02/24 01:39:56;
    rebind 3 2021/02/24 12:38:45;
    expire 3 2021/02/24 15:38:45;
}
```

DHCP server

In kleinere netwerken (thuis) is het meestal de wireless access router die de hosts voorziet van een IP adres aan de hand van een **ingebouwde DHCP server**. Deze DHCP server zal IP adressen geven uit de private IP address range zoals 192.168.0.0/24. In grotere netwerken maakt men meestal gebruik van een **aparte DHCP server** die de netwerkbeheerder kan finetunen zoals hij wil.

Een DHCP server luistert op **UDP poort 67** naar berichten van **clients of relay agents**.

Het Internet System Consortium (ISC) is een non-profit organisatie die een aantal referentie-server-implementaties van internetprotocollen aanbiedt. Zo bij de verantwoordelijk voor de Bind9 DNS server, maar ook voor een van de meest gebruikte DHCP servers: **ISC DHCP**.

Opzetten van een DHCP server

Aan de DHCP server moeten **een of meerdere scopes** worden toegewezen. Een DHCP server kan verantwoordelijk zijn voor meerdere subnetten, zie DHCP relays. Een scope is in deze context een **range van IP adressen** waarvoor die DHCP gemachtigd is om toe te kennen aan hosts. Dat kan een **subnet adresblok** zijn (bv. 192.168.1.0/24), maar kan ook een **deel van een adresblok** zijn (192.168.1.15 - 192.168.1.123).

Toekennen van adressen

Er zijn meerdere manieren waarop een DHCP server een adres kan toekennen aan een host:

De eerste mogelijkheid is dat de DHCP server het IP adres toekent op basis van het **MAC adres** waarvan de **DHCP discover** afkomstig is. Dat zal enkel gaan wanneer er **geen relay** gebruikt wordt.

Een daaraan gekoppelde mogelijkheid is dat er een bepaald aantal IP adressen wordt **geserveerd** voor een bepaald aantal hosts. Zodat de host met een bepaald MAC adres steeds hetzelfde IP adres zal toegekend krijgen.

De derde mogelijkheid is dat de DHCP server een **vrij adres** uit een **lijst van adressen** toekent.

Er kunnen ook nog andere zaken worden meegezonden: de name server (blokkeren van torrent sites), domeinnaam, default gateway, ...

Voorbeeld bestanden te configureren bij ISC DHCP

/etc/dhcpd.conf → configuratie bestand

```
lease-file-name "/var/lib/dhcpd/dhcpd.leases";

subnet 192.168.1.0 netmask 255.255.255.0 {
    option domain-name           "cnet2.ugent.be";
    default-lease-time          86400;   # 24 hours
    max-lease-time               172800;  # 48 hours

    option routers                 192.168.1.1;
    option subnet-mask            255.255.255.0;
    option broadcast-address      192.168.1.255;
    option domain-name-servers   192.168.1.1;
    option ntp-servers            192.168.1.1;
    range   192.168.1.101     192.168.1.200;

    host cnet2server {
        hardware ethernet 00:ca:20:cc:54:89
        fixed address 192.168.1.105
    }
}
```

→ Eerste de locatie van het lease-bestand

→ Dan een blok per subnet

 → een optie domeinnaam die aan alle clients zal uitgedeeld worden

 → daaronder komen de andere meest-gebruikte opties bij DHCP offers (option router= default gateways, kunnen een of meerdere)

 → daaronder welke range binnen het subnet mag beheerd worden

 → daaronder voor een gegeven host met MAC adres een vast IP adres.

/var/lib/dhcp/dhcpd.leases

```
lease 192.168.1.108 {
    starts 0 2021/01/30 08:02:54;
    ends 5 2021/02/04 08:02:54;
    hardware ethernet 00:50:04:53:D5:57;
    uid 01:00:50:04:53:D5:57;
    client-hostname "laptop-wouter";
}
```

→ In het lease-bestand staat de historie van de geleasede IP adressen, niet alleen de actieve

→ Hier staat 1 lease beschreven als entry

Sectie 4.3.4 Network Address Translation Protocol (NAT)

De meeste toestellen op het internet fungeren slechts als client in een client-server architectuur. Op die manier hoeven ze niet zoals een server op een **vast IP adres en poort te luisteren naar binnenkomende verbindingen**, maar maken ze zelfs steeds eerst verbinding met een server. Bovendien bevinden veel toestellen zich achter een gateway router van en netwerk. NAT poogt **alle toestellen achter 1 gateway router** te bereiken via **1 publiek IP adres**, in plaats van alle individuele toestellen een eigen publiek IP adres te geven.

Aangezien elk privaat netwerk IP adressen binnen 192.168.1.0/24 kan gebruiken, kunnen we deze toestellen niet bereiken op hun IP adressen via hun routers. Routers op het internet weten niet naar welk privaat IP adres ze hun pakketten moeten sturen. De router heeft een publiek IP adres gekregen. Om zo'n router met een private gateway adres, met een private range adres toch toegang te geven tot het internet wordt er gebruik gemaakt van **NAT**.

NAT laat toe om toestellen achter de private gateway router initiatief te laten nemen om te communiceren met een server op het internet. De NAT functie in de router gaat een **vertaling** maken. Eerst gaat het na welke **poortnummer** op het **publieke IP adres** van de NAT router nog vrij is en gaat alle pakketten van de client **richting de webserver** vertalen. Het **source IP adres** in de IP header is gelijk aan dat van de NAT router en de **source application port** de toegekende poort wordt. Iedere vertaling wordt aan de NAT tabel toegevoegd op het moment dat de **client** uit het thuisnetwerk verbinding probeert te maken met een server op het internet. De **applicatie poort** wordt in de **header** van het gebruikte **transport protocol** vertaald, dit kan zowel in de TCP als in de UDP header zijn. Op die manier lijkt het pakket dat aankomt bij de server alsof het werd verstuurd vanaf de NAT router zelf.

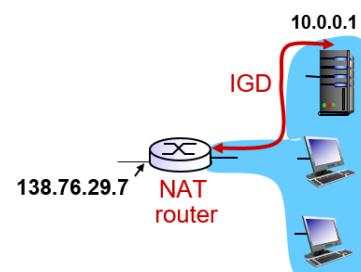
De response wordt naar die NAT router gestuurd die opnieuw het binnenkomende pakket zal vertalen in de **omgekeerde zin** volgens de regel die hij zelf had toegevoegd.

De situatie wordt anders wanneer een **apparaat in het thuisnetwerk** als **server** wilt fungeren. Wanneer een client vanop het internet verbinding probeert te maken met de server lukt dit enkel wanneer er een regel is toegevoegd die de poort nummer linkt aan het IP adres van het apparaat in het thuisnetwerk. Zolang er geen vertaling regel is toegevoegd in de NAT tabel is de server in het private netwerk niet bereikbaar en wordt verkeer gedropt bij de NAT router. Op die manier is NAT niet alleen een efficiënte manier om meer toestellen internet connectiviteit te bieden, het werkt ook als een soort **firewall**. Dit laatste kan ook wel als een probleem aangezien worden en wordt het **NAT traversal probleem** genoemd.

NAT traversal problem oplossingen

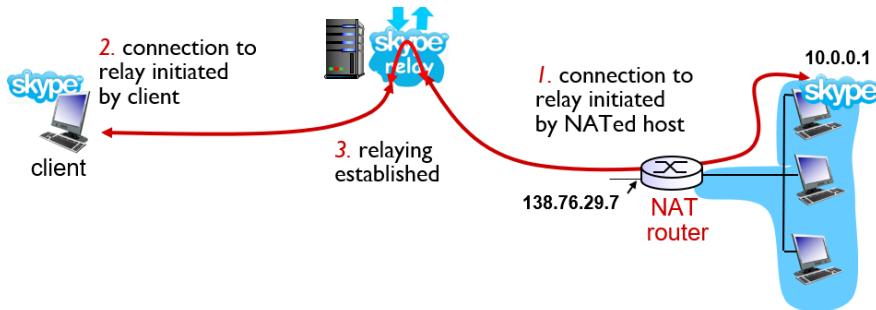
De eerste meest voor de hand liggende oplossing is om **statisch** een **regel** toe te voegen aan de NAT functie. Dat kan vaak aan de hand van een CLI of een webportal.

Een tweede oplossing werkt **slimmer en dynamisch**. Het **Internet Gateway Device (IGD) Protocol** laat toe om een server in het private netwerk **automatisch** te laten leren welke het publiek IP adres is van de NAT router en **dynamisch regels** toe te voegen die ervoor zorgen dat binnenkomend



verkeer op een gegeven poort doorgesluisd wordt naar de server binnen het private range netwerk.

Een derde oplossing werkt met een **relay** (bv. Skype). In plaats van een server op te zetten in het private range netwerk, maakt het toestel binnen het private range netwerk verbinding met een **publieke relay server**. Op die manier komt het initiatief voor de verbinding opnieuw van de client uit het private netwerk en wordt op basis hiervan automatisch een regel toegevoegd in de NAT functie van de router. De andere partij maakt op zijn beurt ook verbinding met de relay server, waarna de relay **beide verbindingen** aan elkaar kan **koppelen** om communicatie tussen de eindpunten mogelijk te maken.



NAT: voor -en nadelen

Meerdere toestellen kunnen aan de hand van 1 IP adres toch met het internet communiceren, er zijn dus **minder publieke IP adressen nodig**.

De IP adressen in het private netwerk **kunnen wijzigen** zonder dat iemand daarbuiten er van op de hoogte moet zijn.

Je kan van **ISP veranderen**, zonder dat adressen in het lokale netwerk hoeven mee te veranderen.

De NAT router kan fungeren als een soort **firewall** een pluspunt.

Gebruiken van transportlaag poorten om pakketten naar het juiste apparaat te leiden in de netwerklaag een **inbreuk op de verschillende lagen**. De router zou maar tot aan laag 3 (netwerklaag) moeten opereren. Bovendien is dit enkel mogelijk zolang er **poorten vrij** zijn op het publieke IP adres van de NAT router.

Het **NAT traversal probleem** zorgt voor een aantal **extra configuratieproblemen** om ervoor te zorgen dat **server toepassingen** mogelijk zijn in het private netwerk.

Als laatste moeten **applicatieontwikkelaars** rekening houden dat NAT hun toepassingen kan dwarsbomen en dus bv. werken met **relay servers**.

Large scale

NAT wordt zelfs door ISP's gebruikt, USP's kregen slechts 1 publiek IP adres ter beschikking. Bij **NAT444** of Carrier Grade NAT wordt NAT twee keer achter elkaar gebruikt. Er worden 3 IPv4 address ranges gebruikt. Eén private range in het klantennetwerk, één private range in het ISP netwerk en één publieke IPv4 adresruimte.

Het laat opnieuw toe **meer toestellen** tot het internet toe te laten met minder beschikbare IP adressen. Dit wordt voornamelijk gebruikt in landen waar IP adresruimte schaars is. Er zijn wel extra problemen met deze oplossing: klanten netwerken en ISP netwerken mogen **geen gebruik** maken van **dezelfde adressen**. Daarnaast kan communicatie tussen toestellen van verschillende klanten netwerken niet noodzakelijk het **kortste pad** nemen, aangezien ze enkel bereikbaar zijn via de NAT van de service provider.

H4*: Networklayer - IPv6

IPv6 address space & notation

IPv6 heeft 2^{128} aantal mogelijke adressen.

Het wordt genoteerd als **8 velden** van telkens **2 bytes** die door een ":" worden gescheiden.

Die 2 bytes, 16 bits, een **hextet**, worden voorgesteld door **4 hexadecimale tekens**.

"xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx", bv. "2001:0660:30F3:AC01:0000:0000:6d43:210f"

Het bestaat uit twee gelijke delen. Het eerste deel van 64 bits is vast en is het netwerkdeel, dit wordt de **(sub-)network prefix** genoemd. Hier zijn geen klassen of een flexibele netmask. Er is ook een tweede deel, een host gedeelte, ofwel **interface identifier (IID)**.

xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx

Interface identifier (IID)

De interface identifier wordt niet random toegekend, er zijn twee manieren om het **zelf te genereren**. De eerste optie is dat het gebaseerd wordt op het **MAC adres** (48-bit) van de **host**. Om van het 48-bit MAC adres over te gaan naar 64-bit, wordt gebruik gemaakt van het **EUI-64** formaat. **Splits** het MAC adres in twee delen en vul het midden op met **ff:fe**. Nadien de **7e bit togglen** om het **IID** te bekomen. MAC → EUI-64 → IID

Opmerking: MAC adressen worden typisch genoteerd in groepjes van twee hexadecimale getallen genoteerd. Een EUI-64 wordt in groepjes van 4 hexadecimale getallen genoteerd.

bv. MAC: 00-08-b3-1e-83-29

→ EUI-64: 0008:b3 ff:fe 1e:8329

→ IID: 0208:b3ff:fe1e:8329

Deze methode brengt een **privacy issue** met zich mee, want als het IPv6 adres steeds het MAC adres bevat ben je wereldwijd traceerbaar door je MAC adres. Hiervoor bestaat er nog een tweede methode, je voegt een MD5 stap toe, een hashfunctie. Hierdoor komen de bits er randomized uit en dat je MAC adres niet meer traceerbaar is.

Hierbij verschilt IPv4 van IPv6, in **IPv4** kan je het host gedeelte niet zelf bepalen. Dat gebeurt ofwel **manueel** ofwel via **DHCP**.

Adres verkorten

De notatie is erg lang dus zijn er enkele regels om het adres korter te noteren.

Leading zeros binnen een hextet kunnen we skippen:

2001:0660:30f3:0001:0000:0000:6d43:210f → 2001:660:30f3:1:0:0:6d43:210f

Opeenvolgende nullen vervangen door een dubbelpunt. Dit kan maar **eenmalig** gebeuren.

2001:660:30f3:1:0:0:6d43:210f → 2001:660:30f3:1::6d43:210f

bv. 2001:db8::1:0:0:1 is al correct

IPv4 adres opnemen in IPv6

Dotted decimal notatie gebruiken om het **32-bit gedeelte** (IPv4 adres) toe te voegen.

bv. 2001:0660:30f3:0001::101.67.33.15

Poortnummers gebruiken met IPv6 adres

Om een poortnummer toe te voegen moet het IPv6 adres tussen **brackets**.

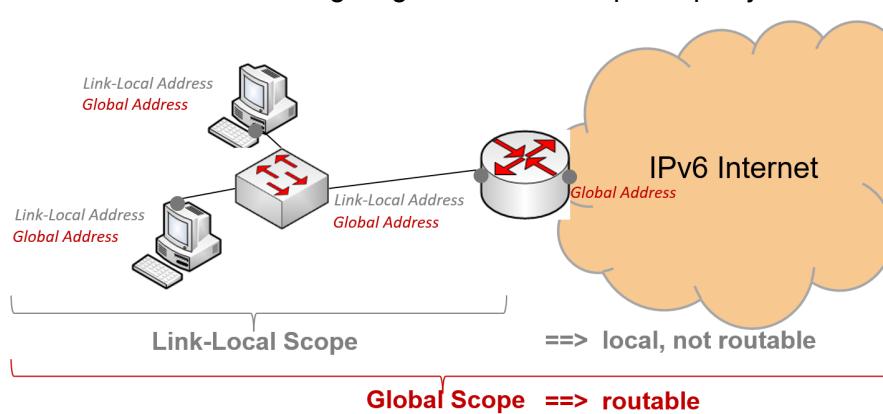
bv. http://[2001:660:30f3:2:a00:20ff:fe18:964c]:443/

IPv6 address scope

IPv6 adressen komen in meervoud. Bij IPv4 was het zo dat je voor 1 interface, 1 IPv4 adres hebt. Bij IPv6 is het omgekeerd, daar heb je typisch **per interface meerdere IPv6 adressen**. Die adressen verschillen van elkaar in **scope** (lokaal/globaal). De scope bepaalt tot waar een adres geldig is. Adressen kunnen ook verschillen in **type** (unicast/multicast).

Scope

Met het ene adres heb je een adres om te communiceren binnen het **lokaal Local Area Network (LAN)**, bv. om een router te gaan vinden. Het **globale adres** kan gebruikt worden om met **heel de wereld** te communiceren. Bij een router wordt de routeringstabell iets complexer want de interfaces gaan meer IPv6 adressen hebben. **Link-local adressen** zijn **niet routeerbaar**, dus de geldigheid van de scope stopt bij de router.



Link-local (inet6 addr)

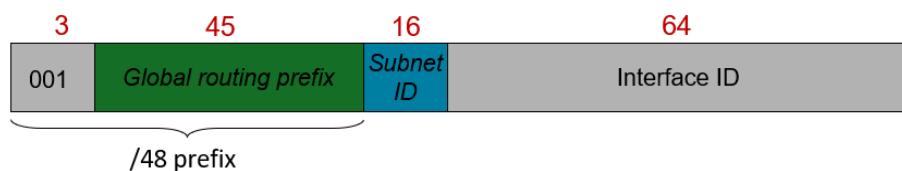
De eerste 64 bits komen neer op **fe80::/64**.

10	54	64
1111 1110 10	0 0	Interface ID

Binnen een LAN zal elke host zijn interface ID hebben, op die manier kan er steeds link-local gecommuniceerd worden. De Interface IDs worden steeds automatisch gegenereerd. Als dat gebaseerd is op het MAC adres heeft iedereen een uniek link-local adres, want iedereen heeft een uniek MAC-adres. Het grootste voordeel hiervan is de mogelijkheid tot **link-local communicatie zonder nood aan configuratie**, zijnde manueel of DHCP.

Global

Binnen de volledige IPv6 ruimte is **2000::/3** de globale adresruimte.



Deze globale adressen worden niet random toegekend. Een bedrijf krijgt een globale prefix toegekend van een ISP die bestaat uit **48 bits**. Die 48 omhelzen je volledige netwerk. Met **de 16 bits die over zijn** kan binnen het bedrijf beslist worden om **subnetwerken** op te zetten.

Een ISP zal zelf zijn adressen te proberen structureren. Een RIR (Europese organisatie) krijgt typisch een /12 of /23 blok. Hierbinnen kan hij gaan kiezen om /48 netwerken te gaan opbouwen.

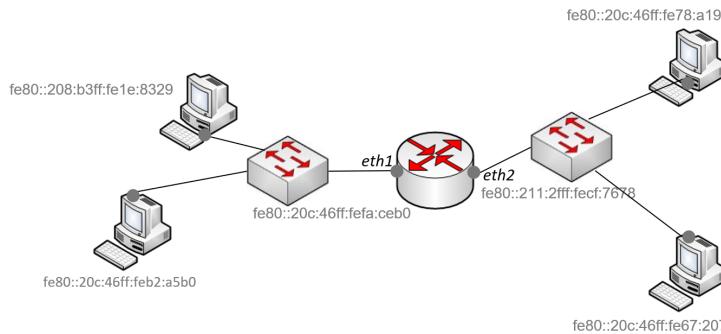
Zo kan het bv. zijn dat binnen België er een heleboel /48 netwerken zijn, maar dat die allemaal vallen binnen een /23 blok die aan Europa is toegekend. Dat wil zeggen dat als we al die adressen in een routertingstabel willen aggregeren dat dat mooi gestructureerd is. Het is dan in principe mogelijk om in een router ergens buiten europa te zeggen, dit /23 netwerk en alle /48 adressen die daaronder vallen zijn te routeren met 1 lijntje.

Een ISP krijgt typisch een /32 prefix. Die kunnen op hun beurt /48 adressen gaan uitdelen.

Deze structuur is iets dat vooraf is beslist om het IPv6 netwerk zo te gaan opbouwen, met als doel om hele **grote blokken van adressen** gaan toekennen op **continentaal niveau**.

IPv6 ULA (Unique Local Address) scope

Stel dat je twee computers hebt verbonden met een switch. Elke computer heeft een link-local adres. Je wilt je netwerken verbinden met een ander netwerk. De link-local adressen zijn niet routeerbaar. Als je niet verbonden bent met een ISP heb je ook **geen globale adressen** toegekend gekregen. Om dit op te lossen kan je een **ULA scope** definiëren. Dit valt te vergelijken met de private adressen die bestaan binnen IPv4. Als we die twee willen verbinden kunnen we een eigen ULA scope gaan **genereren**.



- De scope van het linker netwerk stopt bij eth1 van de router
- De scope van het rechter netwerk stopt bij eth2 van de router

Opbouw adres ULA scope

Er is een stuk adresruimte vrijgehouden om ULA ranges te definiëren, een unieke prefix is voorzien fc00::/7. Door de bit toggle wordt dit **fd00::/7 voor de adressen**.

7	1	40	16	64
1111 110	L	Global ID	Subnet ID	Interface ID

Om zo'n adres te gebruiken moet je een **global ID** genereren, hier bestaat een algoritme voor. De string die hier uitkomt kan je gebruiken als een /48 prefix. Daarbinnen kan je opnieuw een /16 prefix gebruiken om eigen subnetten te definiëren. Die global ID wordt **sudo-random** gegenereerd, zodanig dat de kans heel hoog is dat de **ULA scope globaal uniek** is. Dit heeft zo z'n voordelen. Wanneer lokaal gegenereerd wordt moet de L bit op 1 komen te staan: **prefix fc → fd**. (De L bit op 0 (fc00::/7) is voor in de toekomst)

bv. Global ID: da:c01d:bee2 (40 bits)

→ Prefix: fdda:c01d:bee2:/48

→ Eerste netwerk: ULA fdda:c01d:bee2:**1**::/64

→ Tweede netwerk: ULA fdda:c01d:bee2:**2**::/64

Het wordt benadrukt **geen eenvoudige Global ID** te nemen (bv. 1 of 0), omdat dit **conflicten** gaat opleveren bij het verbinden van twee ULAs.

IPv6 address type

IPv6 adressen bestaan ook uit verschillende types. Een type zegt niks over tot waar een adres geldig is, maar meer waarvoor een adres precies gebruikt kan worden. IPv6 onderscheidt 3 verschillende types.

Een unicast adres voor **een-op-een** communicatie. Een host praat met een andere host.

Er is ook multicast. Een **een-op-veel** relatie.

Daarnaast is een anycast. Dit volgt het **one-to-nearest**, het dichtstbijzijnde adres dat je kan vinden (in aantal hops).

Er is **geen broadcast** meer, dit is vervangen door een multicast.

Multicast adressen

Deze adressen beginnen met **ff::/8**.



Er zijn meer scopes dan gebruikt worden

- *ff01::101 → All NTP servers on the same node as the sender*
- *ff02::101 → All NTP servers on the same link as the sender*
- *ff0e::101 → All NTP servers on the Internet*

Er is **geen broadcast** meer, maar bv. ARP maakt gebruik van broadcast. Indien het toch nodig zou moeten blijken, kan gebruik worden gemaakt van **ff02::1**, **alle nodes binnen het LAN**. Eigenlijk is het niet de bedoeling dat dit gebruikt wordt.

Deze multicast subscriptions zijn terug te vinden in routeringstabellen.

Voorbeeld routeringstabel multicast

```
rout64:~# route -n -6
Kernel IPv6 routing table (Host, reduced)
Destination          Next Hop            Flag Met Ref Use If
fe80::/64            ::                 U    256 0   0 eth1
ff00::/8             ::                 U    256 0   0 eth1
::/0                 fe80::20c:46ff:fefafa:ceb0  UGDAe 1024 0   0 eth1
```

→ link-local netwerk

→ als single node/host ingeschreven op dit multicast netwerk

→ default route naar het internet

Anycast

Naar de **dichtstbijzijnde**.

Het zou bv. gebruikt kunnen worden voor mirror servers, zodat je iets download van de dichtstbijzijnde locatie. Zou kunnen gebruikt worden voor DNS servers, gewoon de dichtstbijzijnde nemen. Ook voor next-hop router, de dichtstbijzijnde router.

Het probleem is dat je de **TCP connectie** die je hebt met een server niet kan veranderen.

Stel voor dat je een mobiele node hebt, dan verhuis je van het ene netwerk naar het andere terwijl je actieve verbindingen hebt. Een andere server komt dan dichter in de buurt, maar de TCP verbindingen kunnen niet worden doorgegeven.

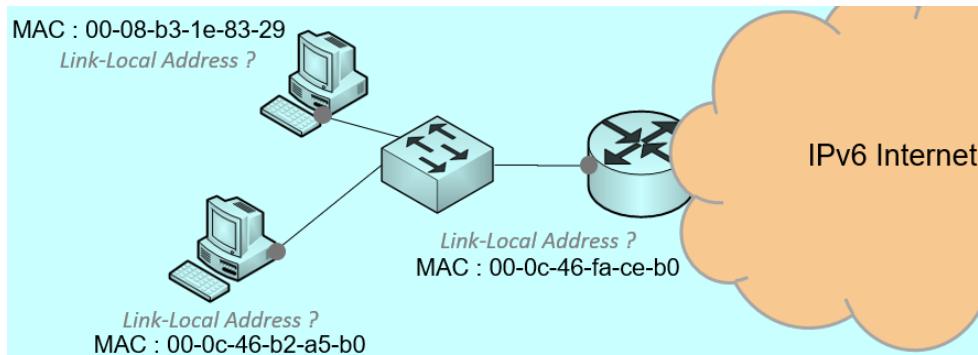
Preserved addresses

Geen broadcast, dus geen nood aan gereserveerde adressen. Dus het all zero's (netwerkadres) of het all one's (broadcast adres) is niet meer voorbehouden.

Response college

Oefening: configureren van klein netwerk. (local, global IPv6)

Hoe gaan de 6 adressen er hier uitzien (local, global)(netwerk: 2001:db8:1::/64)?



Linksboven:

- local: 02:08:b3:fe:ff:1e:83:29 → fe80::208:b3fe:ff1e:8329
- global: 2001:db8:1::208:b3fe:ff1e:8329

Linksonder:

- local: 02:0c:46:fe:ff:b2:a5:b0 → fe80::20c:46fe:ffb2:a5b0
- global: 2001:db8:1::20c:46fe:ffb2:a5b0

Rechts:

- local: 02:0c:46:fe:ff:fa:cd:b0 → fe80::20c:46fe:ffff:cdb0
- global: 2001:db8:1::20c:46fe:ffb2:a5b0

Routerings tabel host

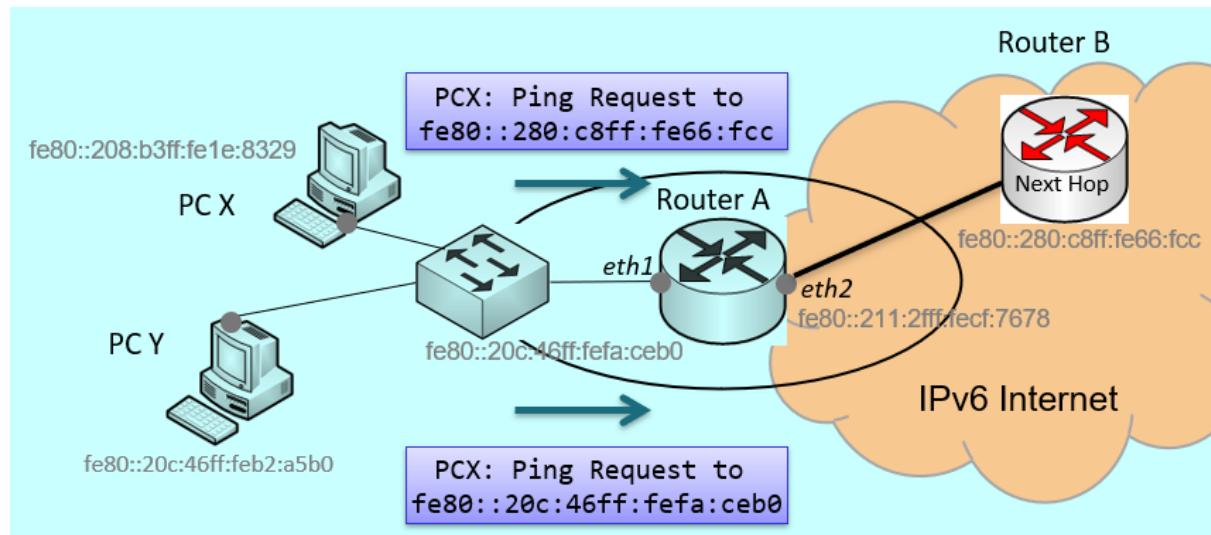
Destination	Next Hop	Flag	Met	Ref	Use	If
2001:db8:1::/64	::	UAe	256	0	6	eth1
fe80::/64	::	U	256	0	0	eth1
::/0	fe80::20c:46ff:fefafa:ceb0	UGDAe	1024	0	0	eth1

→ directly attached network voor de globale adressen (altijd)

→ localhosts

→ default gateway

Oefening 2: routeringstabel router



- a) Kan PC X → router B pingen?
 - b) Kan PC X → router A pingen
- a) Nee, want het is link-local
b) De router wilt antwoorden naar fe80::/64, maar zowel interface eth1 als eth2 begint met fe80::/64. Het onderscheid wordt gemaakt met een **zone-index**. Elke interface ofwel link-local krijgt een uniek zonenummer binnen het OS. Het wordt aangeduid bij de adressen met "*ipv6%zone index*". Het duidt aan welke interface het is van het link-local netwerk.

Oefening 3: geldige IP adressen en scope/type van adressen

Which of the following are valid IPv6 addresses?

2001:6a8::d80:21::49 → ongeldig, maar 1 ":" toegelaten

2001:6a8:1d80:21::49

2001:6g8:1d80:21::49 → ongeldig, "g" is geen hexadecimale waarde

3001:6a8:1d80:21::49

4001:6a8:1d80:21::49 → ongeldig, globale adressen moeten starten met 001, dus kan enkel 2 of 3 zijn

What is the scope/type of these addresses

ff02::2 → Multicast, alle routers in de link-local scope

fe80::1 → Link-local unicast, manueel IID gezet op 1

f8e0::1 → Unassigned, kan niet gebruikt worden

ff02::1:ff0e:8c6c → Solicited node in link-local scope

fd00:1::7 → Unique local (generated locally) (ULA), unicast
Global ID 00:0001:0000, subnet 0000, IID "7"

Valid en/of usable?

2001:6a8:1d80:f21:af78:bb83:310

2a01:578:3::36f7:67f2

fe80::2

fe80:1::1

fe80::1:1

ff02:0:0:0:0:0:9

fc3e:b42d:d178::fcd9:6769:988:7a75

→ Ongeldig, 7 hextetten

→ Unicast adres

→ Link-local unicast, manuele IID op "2"

→ Geldig, bruikbaar binnen fe80::/10, maar niet binnen fe80::/64 (zoals andere nodes)

→ Link-local unicast, manuele IID op "1:1"

→ Geldig, onbruikbaar, L bit staat op 0 (fd00::/8)

Oefeningensessie

Vraag 1 Aggregeer de volgende netwerken tot een zo groot mogelijk geheel

2001:6a8:1d80:7374::/64

2001:6a8:1d80:7375::/64

2001:6a8:1d80:7376::/64

2001:6a8:1d80:7377::/64

→ 2001:6a8:1d80:7374::/62

[4]hex = [0100] bin

[5]hex = [0101] bin

[6]hex = [0110] bin

[7]hex = [0111] bin

2001:6a8:002a::/48

2001:6a8:002b::/48

2001:6a8:002c::/48

2001:6a8:002d::/48

→ 2001:6a8:002a::/47 EN 2001:6a8:002c::/47

[a]hex = [1010] bin

[b]hex = [1011] bin

[c]hex = [1100] bin

[d]hex = [1101] bin

→ 2001:6a8:0028::/45 (niet helemaal juist want dan aggregeer je 8 netwerken in plaats van enkel de 4 gevraagde, extra: 8, 9, e, f)

Vraag 2: Splits de volgende network ranges in 4 subnetwerken van gelijke grootte:

- 2001|6a8|1d80|2080::/58

16b 32b 48b ...

- o 2080 in binary: 0010 0000 10~~00~~ 0000

58b border

- o Subnet 1: 0010 0000 10~~00~~ 0000 => 2001:6a8:1d80:2080::/60
- o Subnet 2: 0010 0000 10~~01~~ 0000 => 2001:6a8:1d80:2090::/60
- o Subnet 3: 0010 0000 10~~10~~ 0000 => 2001:6a8:1d80:20a0::/60
- o Subnet 4: 0010 0000 10~~11~~ 0000 => 2001:6a8:1d80:20b0::/60

- 2001:6a8:1d80:e000::/52

16b 32b 48b ...

- o e000 in binary: 1110~~0000 0000 0000~~

52b border

- o Subnet 1: 1110 0000 0000 0000 => 2001:6a8:1d80:e000::/54
- o Subnet 2: 1110 0100 0000 0000 => 2001:6a8:1d80:e400::/54
- o Subnet 3: 1110 1000 0000 0000 => 2001:6a8:1d80:e800::/54
- o Subnet 4: 1110 1100 0000 0000 => 2001:6a8:1d80:ec00::/54

- 2001:6a8:1d80:7a00::/55

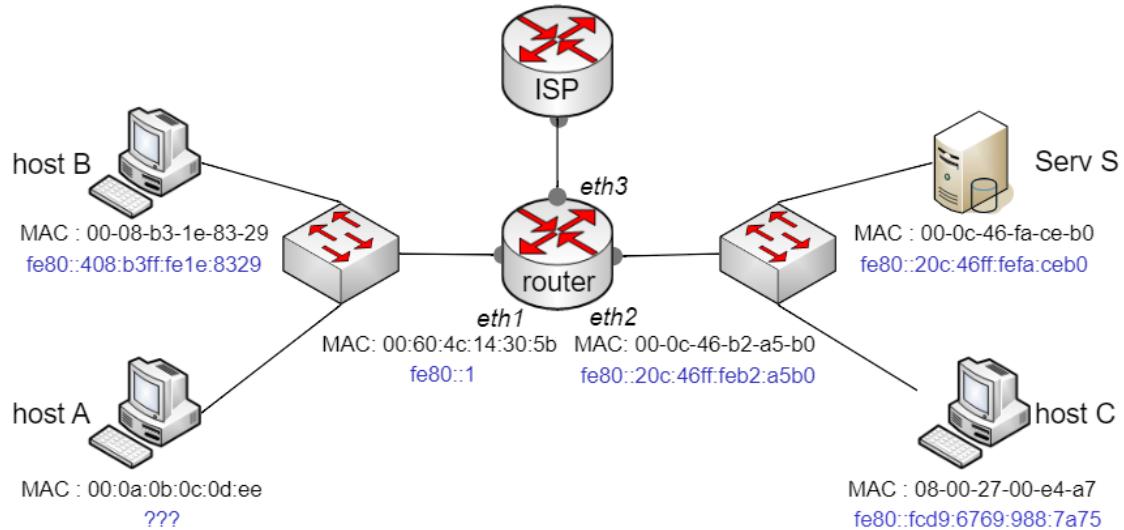
- o 7a00 in binary: 0111 1010 0000 0000

55b border

- o Subnet 1: 0111 1010 0000 0000 => 2001:6a8:1d80:7a00::/57
- o Subnet 1: 0111 1010 0100 0000 => 2001:6a8:1d80:7a80::/57
- o Subnet 1: 0111 1011 0000 0000 => 2001:6a8:1d80:7b00::/57
- o Subnet 1: 0111 1011 1000 0000 => 2001:6a8:1d80:7b80::/57

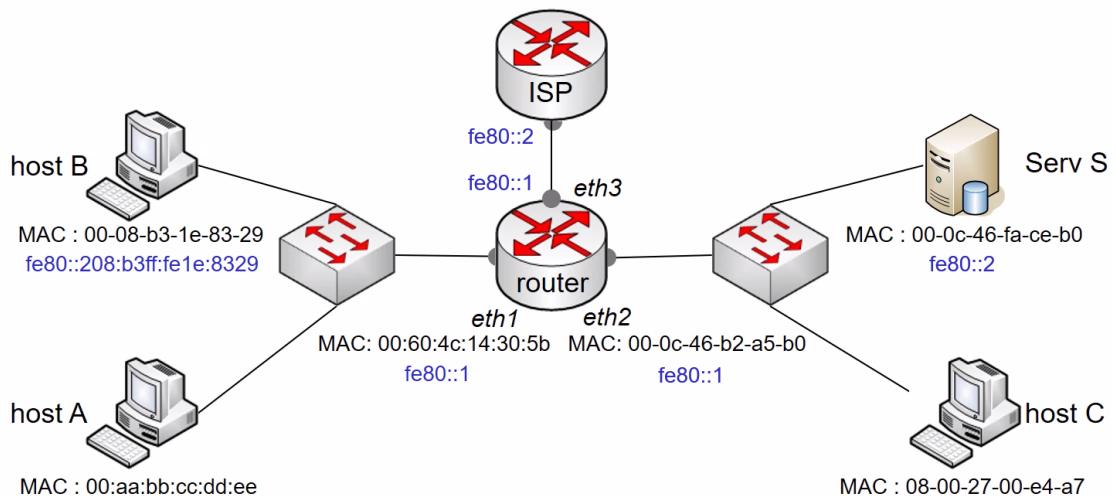
Een hextet bestaat uit 4 nibbles, we gaan aggregeren per die nibbles (zodat er niet over de hextetten heen wordt gegaan).

Vraag 3:



1. Genereer een Link-Local (LL) adres voor host A
fe80::20a:bff:fe0c:dee

2. Geef weer in de tabel hoe de andere hosts hun LL adres werd aangemaakt. Er zit ook vermoedelijk een fout in



	Host B	Host C	Serv S	R / eth1	R / eth2
IID EUI-64 base	x, maar foute bitflip fe80::208:...		x		x
IID privacy extension (md5 based)		x			
Manually assigned IID				x	

3. Een globale prefix 2001:db8:101::/48 werd toegekend aan het bedrijf. Deel op in subnetten van nuttige grootte, en bepaal welke adressen alle interfaces in het netwerk hierdoor zullen krijgen (duidt ze aan op de figuur).

3 Subnets, 2 bits nodig. Laatste hextet kan hiervoor gebruikt worden

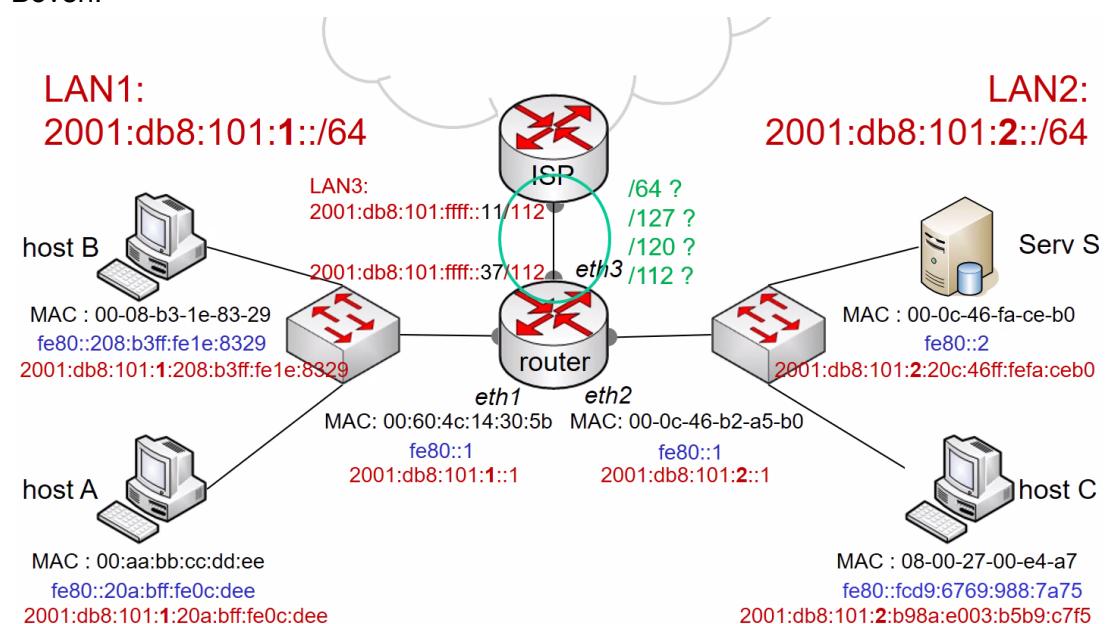
Links: 2001:db8:101:1::/64

- Host A: 2001:db8:101:1:20a:bff:fe0c:dee
- Host B: 2001:db8:101:1:208:b3ff:fe1e:8329
- Rout / eth1: 2001:db8:101:1::1

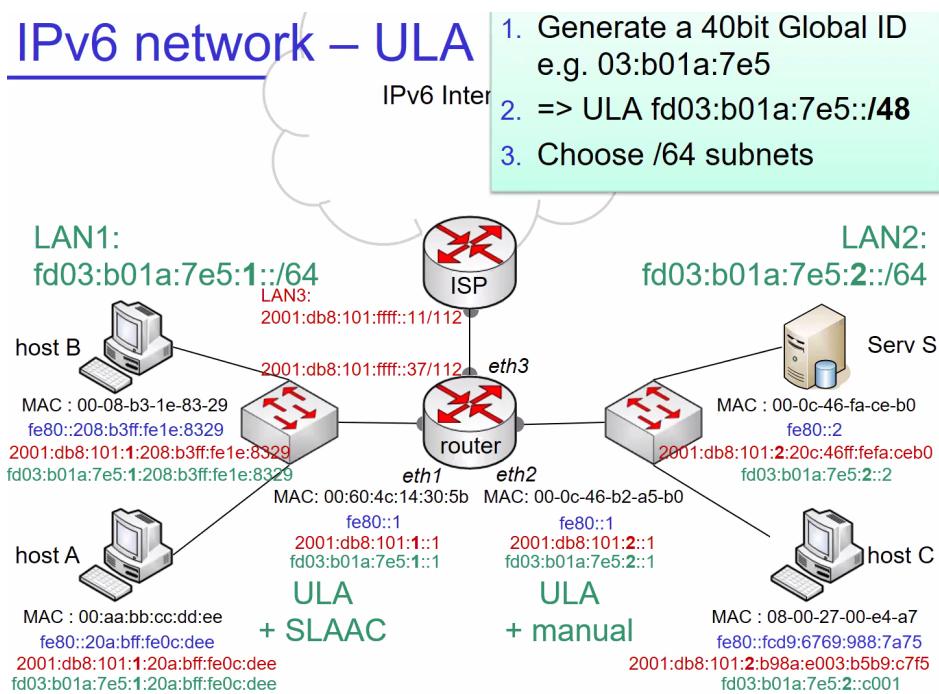
Rechts: 2001:db8:101:2::/64

- Serv S: 2001:db8:101:2::2:20c:46ff:fefab:ceb0
- Host C: 2001:db8:101:2:fcd9:6769:988:7a75
- Rout / eth2: 2001:db8:101:2::1

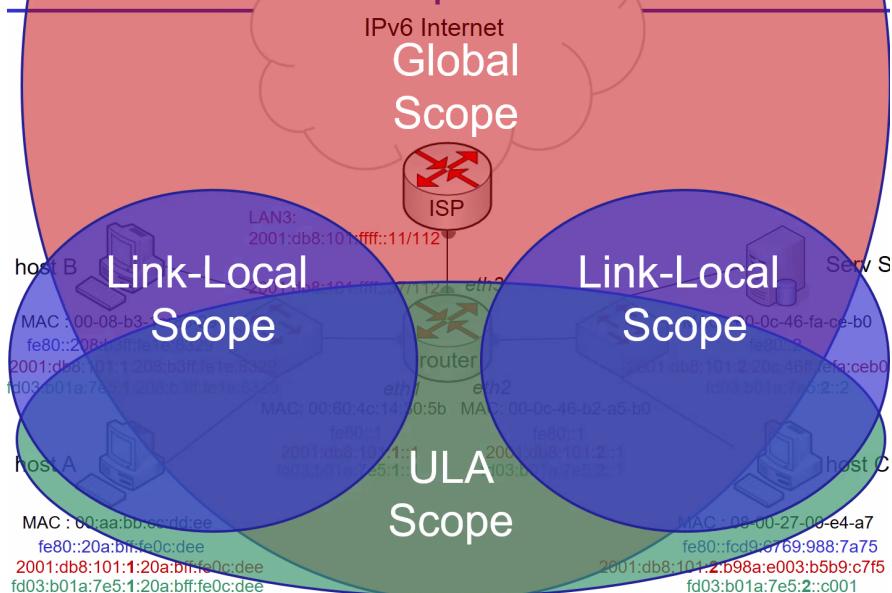
Boven:



IPv6 network – ULA



IPv6 network – scope overview



4. Routeringstabel van de router

Destination	Prefix	Gateway	Interface
::	/0	2001:db8:101:ffff::11	eth3
fe80::	/64	::	eth1
fe80::	/64	::	eth2
fe80::	/64	::	eth3
fd03:b01a:7e5:1::	/64	::	eth1

fd03:b01a:7e5:2::	/64	::	eth2
2001:db8:101:1::	/64	::	eth1
2001:db8:101:2::	/64	::	eth2
2001:db8:101:ffff::	/112	::	eth3

5. Welke routing table entries kunnen worden toegevoegd in die van de ISP?

- a) 2001:db8:101::/48

Correct & fully assigned to our network

- b) 2001:db8:101:1::/63

Foute aggregatie, hier zitten niet alle adressen in. We hebben 1 en 2 gekozen en niet 0 en 1. Dus we aggregeren hoogtens op

- c) fd03:b01a:7e5::/48

ISP is geen deel van de ULA scope

IPv6 address resolution

Solicited-node multicast

Adres resolutie koppelt de **netwerklaag** met de **datalinklaag** en zorgt voor de binding tussen netwerklaag adressen (IP) en datalinklaag adressen (MAC). In IPv4 gebeurde dit in een apart protocol, ARP. Adres resolutie was geen deel van het IPv4 protocol en was iets dat gebeurde op ethernet niveau. Binnen IPv6 is dit **deel van het protocol zelf**. Adres resolutie gebeurt van **ICMPv6 berichten** die ingebakken zitten in het IPv6 protocol.

In IPv6 doen we geen broadcast meer, dus we gebruiken **multicast** (maar geen all-nodes adres ff02::1). We gaan wel gebruikmaken van het **Solicited-node multicast** adres. Het wordt geconstrueerd door de **laatste 24 bits** te nemen van de node zijn **unicast adres** en te appenden aan **ff02::1:ff00:0/104**

bv. IPv6 Unicast: 2001:db8:1::20c:46ff:fe0e:8c6c
→ Solicited-node multicast: ff02::1:ff0e:8c6c

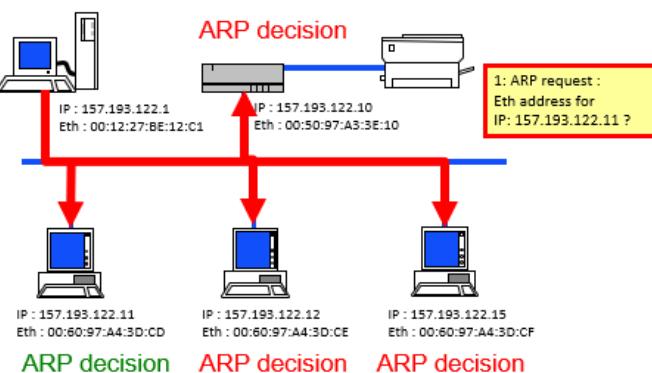
Als je dus een multicast adres configueert op een computer dan schrijft hij zich ook in op de **multicast group** (ff02::1:ff00:0/104) en ook op het **adres** (Solicited-node multicast). Dit heeft het voordeel dat we kunnen multicasten naar deze node, zonder dat andere nodes op deze LAN ingeschreven moeten zijn op deze multicast group. Als we daar een bericht naar sturen weten we dat het enkel zal aankomen bij die ene bestemming → **pseudo-unicast**.

Op **ethernet niveau** (datalink laag) zal er ook een corresponderende multicast group aangemaakt worden.

Multicast is iets dat kan gebeuren zowel binnen de **netwerklaag** als binnen de **datalink laag**. Dit adres wordt opgebouwd met de prefix **33:33:ff** en daaraan worden opnieuw de laatste 24 bits van zijn **unicast adres** toegevoegd.

→ Ethernet (IPv6) multicast: 33:33:FF:0E:8C:6C

ARP broadcast is inefficiënt

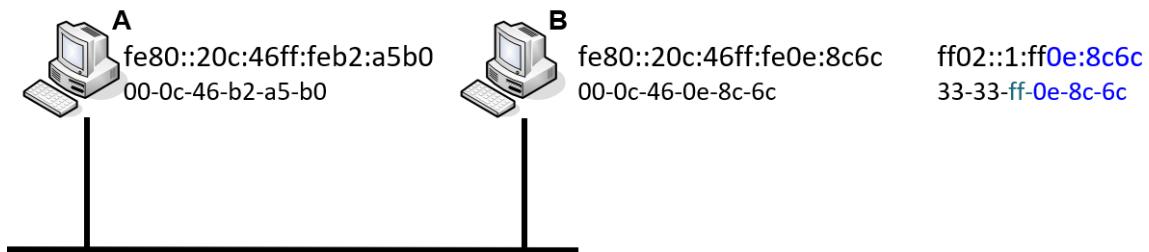


Een **broadcast stoort elke host**. Wanneer de netwerkkaart een broadcast detecteert, neemt deze die binnen en stuurt een **interrupt** naar het OS. Dat is niet schaalbaar, want hoe meer ARP broadcasts binnenkomen, hoe meer de CPU onderbroken wordt.

De multicast group wordt geconfigureerd op de **netwerkkaart** door de CPU. Dit zorgt ervoor dat de CPU niet moet kijken of het voor de host bestemd is, maar de netwerkkaart dit kan doen.

Address Resolution IPv6 (NS & NA)

Het lijkt in een zekere zin op ARP, maar maakt nu gebruik van multicast en unicast.



→ Als computer B een link-local adres genereert, dan schrijft hij zichzelf meteen in op dat multicast adres. Ook op zijn **netwerkkaart** (NIC) zal hij een **multicast adres** bij programmeren dat correspondeert met dezelfde 24 bits. Zo schrijft hij zich in op zowel de datalinklaag als de netwerklaag in op een multicast adres.

Ethernet Header dst: 33-33-ff-0e-8c-6c src: 00-0c-46-b2-a5-b0	IPv6 Header dst: ff02::1:ff0e:8c6c src: fe80::20c:46ff:feb2:a5b0	ICMPv6 Header	Neighbor Solicitation Header target: fe80::20c:46ff:fe0e:8c6c
--	---	---------------	---

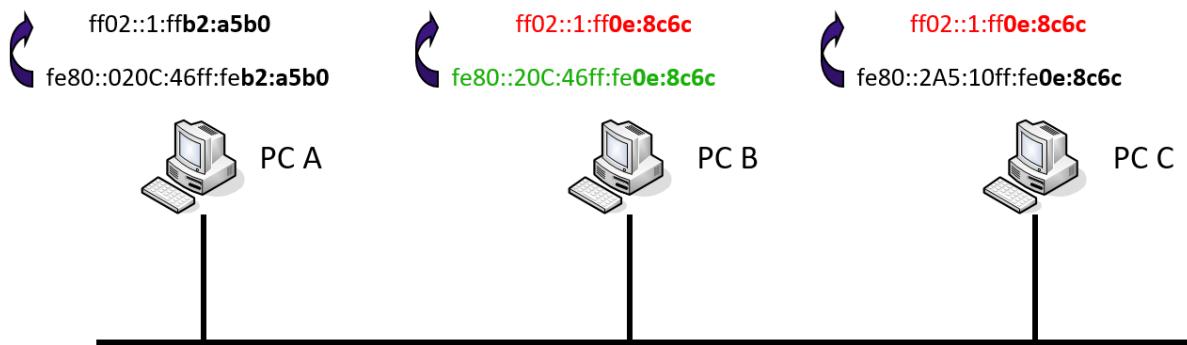
→ Als A op zoek gaat naar dit adres, genereert hij een **Neighbor Sollicitation (NS)**. Hierin staat de source van computer A, hij kent zijn eigen IPv6 adres, zijn eigen MAC adres. Hij stuurt een sollicitation met de vraag "wie is de eigenaar van dit IPv6 adres, mag ik weten wat je MAC adres is". Uit dit adres kunnen we het Solicited-node multicast adres afleiden en deze steken we in de IPv6 header. Nadien construeren we het adres voor de ethernet header.

De datalinklaag gaat dit ook multICASTEN en het komt aan bij computer B. Computer B is ingeschreven op dit datalinklaag multicast ethernet adres. Neemt dat binnen en geeft het door aan de multicast subscriptie die bestaat op host B. Die krijgt het pakket binnen en zal deze (applicatielaag) vraag interpreteren binnen het OS.

Ethernet Header dst: 00-0c-46-b2-a5-b0 src: 00-0c-46-0e-8c-6c	IPv6 Header dst: fe80::20c:46ff:feb2:a5b0 src: fe80::20c:46ff:fe0e:8c6c	ICMPv6 Header	Neighbor Solicitation Header target: fe80::20c:46ff:fe0e:8c6c mac: 00-0c-46-0e-8c-6c
---	---	---------------	--

→ Computer B genereert het antwoord, een **Neighbor Advertisement (NA)**. Hierin gaat computer B antwoorden met "Hier is mijn IPv6 adres, ik ben de eigenaar, en dit is mijn MAC adres". Dit wordt teruggestuurd naar de afzender. Dit is dus een **unicast** antwoord op de multicast Neighbor Sollicitation.

Pseudo-unicast



Stel dat er toch twee hosts zijn op het netwerk met dezelfde 24 bits achteraan in hun IPv6 adres. Als PC A een bericht uitstuurt, een Neigbor Sollicitation, gaan zowel PC B als C deze vraag ontvangen. PC B heeft inderdaad het gevraagde IPv6 unicast adres en zijn OS zal de vraag in de application layer interpreteren. Hier zal die detecteren dat PC B het adres heeft en zal antwoorden met een unicast antwoord. PC C zal ook het bericht ontvangen en het OS zal dit ook interpreteren. Het OS zal zien dat deze vraag niet voor hem bedoeld is, het is een ander unicast IPv6 adres.

We gaan er steeds van uit dat op de datalinklaag ook **ethernet multicast** wordt ondersteund. Is dit niet het geval zal **ethernet** een **fallback** mechanisme doen naar **broadcast**. Stel dat dit bericht wordt gebroadcast naar alle computers op het netwerk, maar die zijn op ethernet niveau niet ingeschreven op het multicast adres. De netwerkkaart zal het niet oppakken, dus komt het zelfs niet binnen in het OS.

IPv6 Duplicate Address Detection (DAD)

Een IPv6 node kan een eigen IPv6 adres genereren, maar moet dan een stap ondernemen om te detecteren of dat adres wel degelijk uniek is. Deze stap heet **Duplicate Address Detection (DAD)**. Het maakt gebruik van ICMPv6 boodschappen, de **Neighbor Solicitation (NS)** en de **Neighbor Advertisement (NA)**.

Er zijn 3 manieren om aan een host identifier (IPv6 adres) te geraken. De eerste twee bestaan ook in IPv4 en zijn een **statisch adres** of eentje via een **DHCP** service. De derde mogelijkheid is **zelf een genereren**. Als je die host identifier zelf genereert, dan moet je ervoor zorgen dat dat **uniek** is binnen de scope waarin je opereert. Je kan dit nagaan door gebruik te maken van een **Neighbor Sollicitation**.

Het proces gaat in zijn werk door een adres te genereren en dan het adres **tentative** aan te nemen. Je zou het willen gebruiken, maar je mag het nog niet gebruiken tot je zeker bent dat het uniek is. Je stuurt een **Neighbor Sollicitation** in het netwerk en als er iemand antwoordt, zal dit zijn met een **Neighbor Advertisement**. Aangezien degene die de sollicitation heeft gedaan niet beschikt over een IPv6 adres (::), zal er geantwoord worden met een advertisement naar all-nodes on link-local adres (ff02::1). Indien dit gebeurt markeer je het adres als **duplicate** en ga je dat niet gebruiken. Als niemand antwoordt na een bepaalde timeout, dan ga je de status van het adres van **tentative** naar **preferred**.

IPv6 StateLess Address AutoConfiguration (SLAAC)

IPv6 laat toe dat een node zijn adres helemaal zelf configureert, zonder dat het manueel moet worden ingegeven door een administrator of dat er nood is aan een DHCP service in het netwerk. Dit proces heet **StateLess Address AutoConfiguration (SLAAC)**.

Binnen ICMPv6 zijn er meerdere soorten pakketten gedefinieerd. Neighbor Solicitation (NS) en Neighbor Advertisement (NA). Die laten het toe om een **link-local adres** (fe80::/64) zelf te kiezen, de **interface identifier** te genereren en na te gaan met **Duplicate Address Detection (DAD)** na te gaan of dat adres uniek is binnen het netwerk.

Wil je een **globaal adres** (andere scope) heb je daarvoor **Router Sollicitation (RS)** en **Router Advertisement (RA)**. Die communiceren het **netwerkgedeelte** binnen het netwerk en zorgen voor **Router Discovery**. Dat laat toe om een **globaal adres** in te stellen zonder bv. DHCP.

Autoconfiguration biedt dus de mogelijkheid zelf een adres te genereren zonder servers. De idee is dat je de router minimaal moet configureren. Simpelweg dat het **globale adressen** heeft binnen een /64 prefix en dat hij dat kan verdelen aan nodes. De nodes kunnen aan de hand van die prefix zelf een adres kiezen.

Voorbeeld: SLAAC

(1) Local Address Detection

Je eigen link-local adres genereren. Je kiest je eigen Interface Identifier (IID) en duidt het aan als tentative. Je gaat de nodige multicast groepen joinen: all-nodes (ff02::1) en solicited-node. Nadien doe je aan Duplicate Address Detection (DAD) zodat adres van tentative → **preferred**. Dan heb je een **link-local adres** dat je kan gebruiken om full-stack operation te gaan doen binnen het LAN waarmee je nu verbonden bent bv. op zoek gaan naar een router binnen dat netwerk.

(2) Global Address Detection

Je kan dan een Router Sollicitation doen, hiervoor is een specifiek multicast adres: ff02::2. Elke router moet ingeschreven zijn op dit multicast adres. De router zal antwoorden met een Router Advertisement die mededeelt wat er van informatie beschikbaar is over het netwerk bv. de prefix, MTU, ... behalve DNS. Dit wordt verstuurd naar het all-nodes adres (ff02::1). Eenmaal dat je die informatie hebt kan je de prefix binnen nemen en genereer je je eigen identifier (IID). Nu moet je opnieuw DAD doen om te bevestigen dat dit adres uniek is binnen jouw netwerk.

IPV6 - DHCPv6

Een DHCP server zit niet noodzakelijk in een IPv6 netwerk. Je kan het nog steeds gebruiken voor gewone operatie of om een DNS server te ontvangen. De DNS informatie zitten niet standaard in de Router Advertisement via SLAAC. Dus DHCP is een optie die gaat plaatsvinden nadat er eerst adressen zijn gegenereerd met behulp van SLAAC.

Als SLAAC heeft plaatsgevonden kan er in de Router Advertisement een **flag** worden aangezet om aan te duiden dat het netwerk ook nog een DHCP server heeft. De DHCP server kan dan gecontacteerd worden op een vooraf gedefinieerd multicast adres (ff02::1:2).

Nadat de client een router sollicitation en een router advertisement heeft gedaan, gaat hij contact opnemen met een DHCPv6 server en die zal extra informatie aanbieden waaronder bv. de DNS server.

DHCPv6 kan gebruikt worden op twee manieren. Het kan gebruikt worden om alleen maar **extra informatie** te verkrijgen, **Stateless DHCPv6**. De tweede mogelijkheid is dat de DHCP server gaat managen in het netwerk en dat hij werkt als een klassieke DHCP server en niet alleen die extra informatie gaat verdelen, maar **ook adressen uitdelen** en bijhoudt wie welk adres heeft, **Stateful DHCPv6**.

DHCPv6 werkt zoals DHCPv4 ook met 4 berichten, maar ipv "discover, offer, request, acknowledge", "solicit, advertise, request, reply" heten. DAD wordt uitgevoerd zelfs na het ontvangen van een adres van de DHCP server.

H5: Netwerklaag

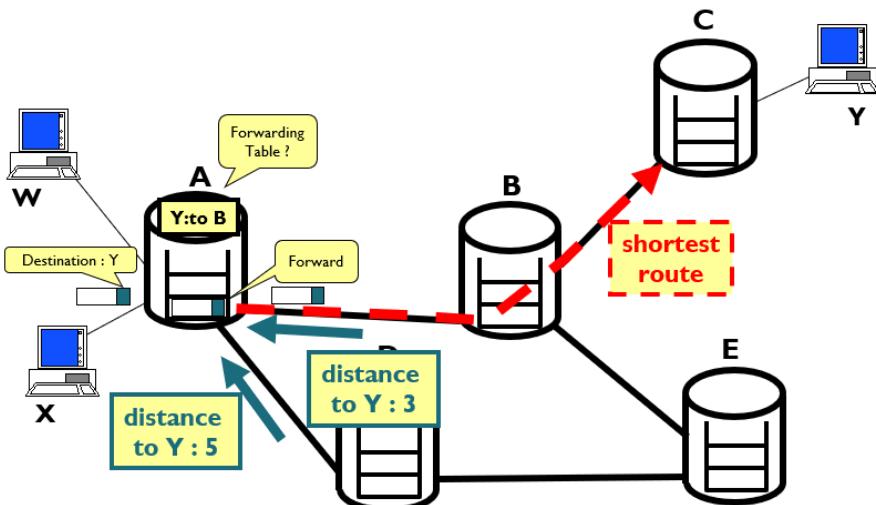
Sectie 5.1 Introductie

Routing <> Forwarding

Routing is het proces dat wordt gebruikt door routers om te bepalen welk **pad** moet genomen worden naar gegeven bestemmingen. Hiervoor kunnen ze gebruik maken van **gecentraliseerde** of **gedistribueerde** algoritmen om het kortste pad/kortste afstand te bepalen tussen bron netwerk en bestemmings netwerk.

Forwarding is het proces dat zich bezighoudt met het **ontvangen** en **doorsturen** van datapakketten gebruikmakend van de forwarding tabel.

Voorbeeld:



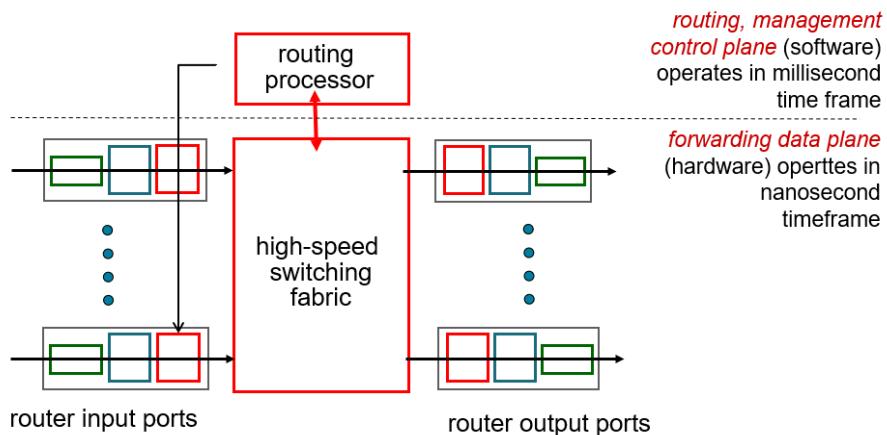
→ Opdat router A zijn forwarding tabel zou kunnen invullen voor de bestemming y moet hij een kortste pad kunnen berekenen naar die bestemming. Dit doet hij gebaseerd op afstandsinfooratie die hij ontvangt van andere routers. Eenmaal die informatie is ontvangen kan hij de forwarding tabel invullen.

Router architecture overview

Een router heeft een bepaalde hardware architectuur.

Die bestaat enerzijds uit **input poorten**, waarop functionaliteit zich bevindt om frames te ontvangen en te versturen, gebruikmakend van gegeven Link Layer functionaliteit.

Anderzijds bestaat het uit hardware om lookups te kunnen uitvoeren in de **forwarding tabellen**. Op basis daarvan moet geschakeld worden naar een gegeven uitgaande poort en dit gebeurt in heel snelle schakellogica die centraal zit.

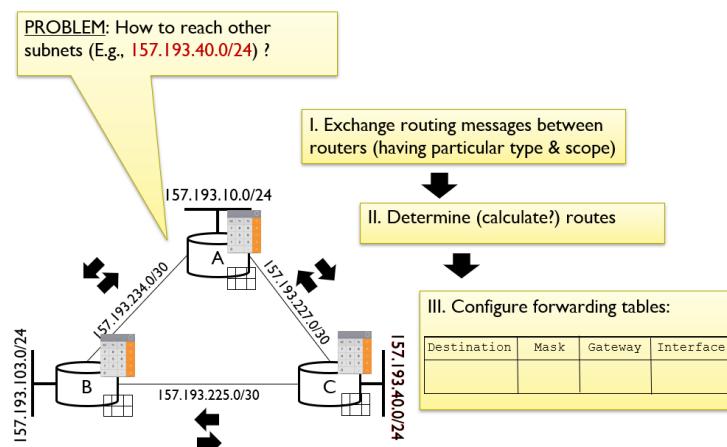


→ Bovenaan zit de **routing processor**. Dit is de processor die gebruikt wordt om **routing algoritmen** op uit te voeren. Deze maken gebruik van **software** en maken deel uit van de '**control plane**'. Routes berekenen gebeurt meestal in de orde van milliseconden of iets langer.

→ De '**data plane**' wordt geïmplementeerd in de router input poorten en de snelle schakel logica. Het moet heel snel kunnen uitgevoerd worden, vaak in de orde van nanoseconden.

→ De '**control plane**' beïnvloedt de '**data plane**', aangezien de uitkomst van de routeringsprocessen ingevoerd wordt in de kopieën van de forwarding tabellen op de router poorten.

Globale routeringsproces



Het probleem van routering is dat een gegeven router moet kunnen bepalen hoe een gegeven bestemming bereikt kan worden. In het voorbeeld wilt router A een route bepalen naar het subnet rechtsonder verbonden met router C. We kunnen dit manueel bepalen, maar in een groter netwerk hebben we snel nood aan meer geautomatiseerde routeringsprocessen.

In een eerste stap gaan routers **informatie met elkaar uitwisselen**. Dit wordt bepaald door twee zaken, de **scope** en het **type** van het gebruikte protocol.

Eenmaal die informatie is uitgewisseld, kunnen routers bepalen **welke route** ze gaan gebruiken.

Tot slot worden de **forwarding tabellen** ingevuld op basis van de gekozen routes.

Ieder routing protocol volgt ongeveer zo'n proces. Er kunnen meerdere iteraties nodig zijn om het gewenste resultaat te bereiken.

Routing Protocol Scope

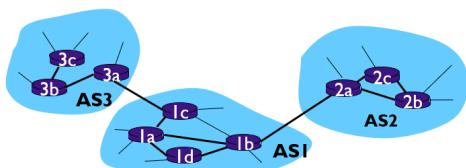
Moeten alle routers van het internet met elkaar en routingsinformatie met elkaar uitwisselen?

Neen, het internet is hier veel te groot voor, de routeringstabellen zouden veel te groot worden en zou heel veel bandbreedte innemen (veel verkeer).

Daarom wordt het internet opgesplitst in **Autonomous Systems (AS)**. Dit zijn netwerken van routers die beheert worden door **dezelfde instantie** (ISP's, BelNet, ...).

Op die manier ontstaat er een **hiërarchisch systeem** van routeren waarin we **intra-AS** routing protocollen gebruik voor routers binne **hetzelfde autonome systeem**. En **inter-AS** routing protocollen tussen routers van **verschillende systemen**.

Voorbeeld:



→ Gegeven een netwerk bestaande uit drie autonome systemen. Elk met hun eigen routers.

Om routes te bepalen binnen zo'n autonoom systeem gaan enkel die routers met elkaar informatie uitwisselen aan de hand van **intra-AS routing protocollen**. Alle routers binnen het AS moeten **hetzelfde routing protocol** gebruiken. Op die manier leren routers hoe ze bestemmingen binnen hun eigen AS kunnen bereiken.

Om routes te leren naar andere systemen worden **inter-AS** protocollen gebruikt. De enige routers die participeren in deze context zijn **gateway routers**. Dit zijn routers die verbonden zijn met andere AS'en. Deze zijn verantwoordelijk voor communicatie tussen hun eigen AS en andere AS'en.

Routing Protocol Type

Los van de scope kunnen routers verschillende soorten van informatie met elkaar gaan uitwisselen. We onderscheiden drie soorten protocollen.

De eerste zijn protocollen waarmee routers **link-states** (toestandsinformatie van links) met elkaar uitwisselen. Deze toestands informatie geeft aan of een link up/down is, capaciteit, vertraging van die link, ... → **link-state protocollen**

Een tweede categorie van routing protocollen wisselt **afstands informatie** naar bestemmingen uit. → **distance vector protocollen**

Een derde categorie wisselt de nodige **hops** naar een gegeven bestemming uit.
→ **path vector protocollen**

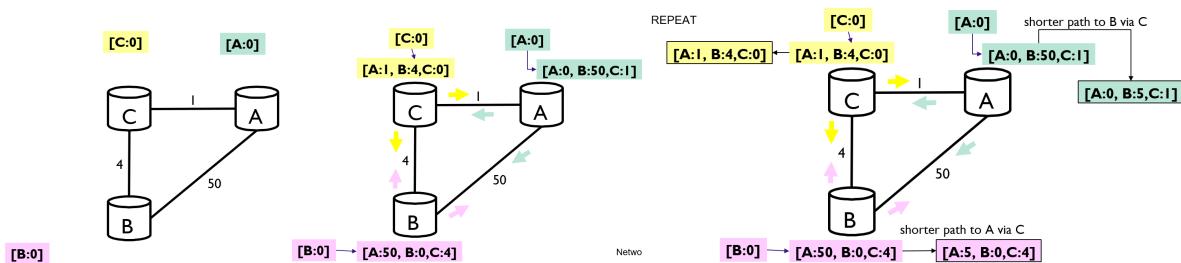
Routing Protocol Overview

scope	type		
	link-state	distance vector	path vector
intra-AS /Interior Gateway Protocol (IGP)	OSPF IS-IS	RIP IGRP	
inter-AS / Exterior Gateway Protocol (EGP)			BGP

Sectie 5.2 Routeringsalgoritmen: distance vector routing

Een van de oudste routing mechanismen. Het steunt op een volledig gedistribueerd mechanisme dat bestaat uit een aantal stappen om de kortste paden naar gegeven bestemmingen in een netwerk te bepalen. Op basis hiervan kunnen de forwarding tabellen ingevuld worden.

Werking:



Elke link tussen de routers heeft een bepaalde **kost**. Hoe hoger de kost, hoe minder wenselijk de link gebruikt wordt.

Bij distance vector routing houdt iedere router een **vector** bij met de **afstanden** naar alle routers in het netwerk.

In de eerste instantie wordt die vector in iedere router geïnitialiseerd op 0, voor de afstand naar zichzelf.

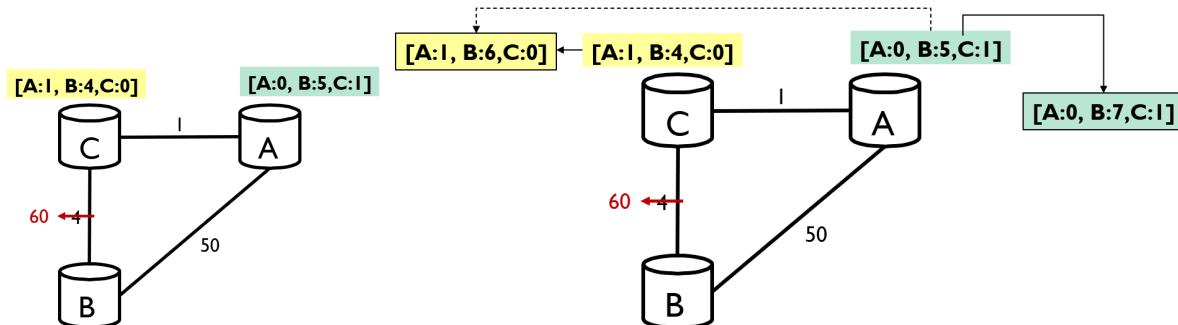
In de tweede stap gaat de router die distance vector **adverteren** naar zijn buren. Gegeven de ontvangen distance vectoren gaat iedere router nu zijn eigen distance vector updaten. Wanneer het een nieuwe bestemming betreft, worden die **toegevoegd** aan de bestaande vector. Wanneer het gaat over bestemmingen die reeds in de eigen distance vector zaten, dan wordt een eenvoudige **vergelijking** gemaakt (Bellman-Ford). Als het pad naar een gegeven bestemming koper is via een buur, rekening houdend met de afstand van jezelf naar die buur. Dan wordt dit nieuwe koperste pad opgeslagen, alsook de buurt die hier toegang tot geeft.

Dit proces wordt **voortdurend herhaald** zodat nodes ook paden kunnen leren via buren van hun buren enz... Op die manier **convergeren** de distance vectoren van alle nodes naar de koperste afstanden.

Bijzondere karakteristieken:

Goed nieuws verspreidt zich **snel** in het netwerk, terwijl **slecht nieuws traag** gaat. Hiermee bedoelen we dat wanneer er een sneller pad opduikt tussen de distance vectoren, dat routers dit snel oppikken en snel convergeren naar dit nieuwste snelste pad. Wanneer er zich echter een fout voordoet in het netwerk, reageren de distance vectoren erg traag.

Count to infinity (traag slecht nieuws)



→ De link tussen router C en B krijgt een grotere kost toegewezen. Dit kan te wijten zijn aan netwerk congestie, netwerk fout, hogere kost van de verbinding, ... Ten gevolge van de hogere link kost moet router C zijn distance vector richting B gaan updaten. Die update gebeurt zoals normaal aan de hand van de ontvangen distance vectoren van zijn buren. Aangezien router A een distance vector heeft doorgestuurd naar router C waarin hij aangeeft dat B kan bereikt worden met kost 5, beslist router C dat de nieuwe kortste afstand naar router B via A verloopt. Omdat de kost van A naar B gelijk is aan 5 en de kost van C naar A gelijk is aan 1. De nieuwe distance vector in C naar B wordt bijgevolg 6.
Uiteraard is dit een verkeerde beslissing vermits de gecommuniceerde distance vector van A naar C eigenlijk gebruik maakte van de distance vector die door C reeds aan A werd gecommuniceerd.

In de volgende iteratie gaat ook C zijn distance vector naar zijn buren doorsturen, waardoor ook A zijn distance vector naar B moet gaan updaten. Ook voor A is het pad nog steeds het kortst via C, aangezien de kost via C nog steeds gelijk is aan 1 en de afstand van C naar B werd gecommuniceerd als 6 in deze iteratie. Bijgevolg wordt de nieuwe distance vector van A naar B 7. We zijn hier in een traag aftellend proces terechtgekomen. Tot A zich realiseert dat het kortste pad zich realiseert via zijn rechtstreekse link.

Het proces gaat blijven duren tot de distance vector die C naar A stuurt, 50 wordt. Op dat moment gaat A zich realiseren dat het kortste pad van A naar B via de rechtstreekse link naar B loopt.

Dit trage update-proces lijkt oneindig lang te duren en wordt daarom ook '**count to infinity**' genoemd.

Oplossingen:

Split horizon geeft nooit de distance terug aan een buur voor een bestemming waarvoor het zelf gebruik maakte van die buur.

Split horizon poisoned reverse geeft wel de distance terug voor een bestemming waarvoor het gebruik maakte van die buur, maar zet de distance op **oneindig**. De buur die deze oneindige afstand ontving voor een gegeven bestemming is dan verplicht om terug te vallen op andere distance vectoren voor die bestemming.

RIP (Routing Information Protocol) (intra-AS)

Dit is een van de oudste protocollen voor **intra-AS** gebruik. Het is een eenvoudig protocol dat best kan gebruikt worden in kleine netwerken. Iedere router wisselt iedere 30 seconden distance vectoren uit voor alle routers die buren zijn. De afstands metriek die gebruikt wordt is een eenvoudige **hop count**, eerder dan de link kost. Iedere advertisement die naar een buur verstuurd wordt kan maximaal de afstand naar 25 bestemmingen/subnetten bevatten.

Als er een link fout optreedt door bijvoorbeeld een defecte kabel, komen er geen advertisements van de betrokken buur. Wanneer dit 180 seconden zo blijft gaan de aangrenzende routers de **bijhorende buur** als onbereikbaar aanduiden en **alle routers via die buur** als ongeldig maken. De bijhorende distance vectoren worden dan geüpdatet naar de bijhorende buren.

RIP routing daemon

Routing algoritmen worden uitgevoerd door een **routing processor in software**. Vaak is dit een **daemon proces** dat luistert op een gegeven poort, net zoals een applicatie server dat doet bv. een webserver. De **advertisements** die worden uitgestuurd naar de buren worden geëncapsuleerd in een **IP packet, zonder** gebruik van een **transport layer protocol**.

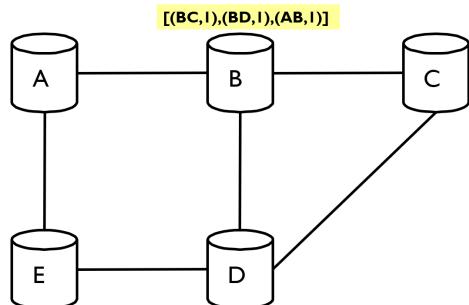
Sectie 5.3 Intra-AS routering: Link-state routing & OSPF

De **distance vector protocollen** zoals eerder behandeld zijn in staat om kortste paden te gebruiken naar gegeven bestemmingen, zonder dat individuele nodes beschikken over de volledige route naar die bestemming. Distance vector protocollen houden enkel vectoren bij met **afstanden** naar hun bestemmingen en hebben **geen inzicht in de topologie** van het netwerk.

Link-State routing protocollen hebben **kennis van de hele netwerktopologie** en bekomen dit door de **uitwisseling van Link-States**.

Link-State Packets

Fase 1:



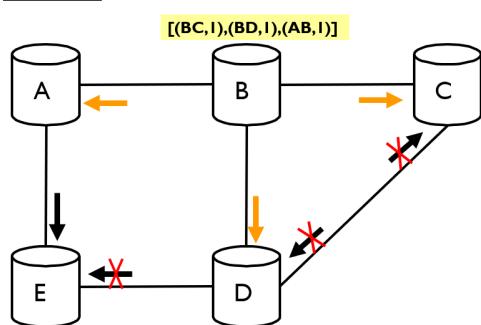
Iedere router kan op zich een lijst maken van de links waarmee hij verbonden is. Router B weet bijvoorbeeld dat hij een link heeft met router C en dat de kost om die link te gebruiken = 1. Deze informatie vormt de link-state voor de verbinding B-C.

In dit voorbeeld wordt **hop count** als kost meting gebruikt. In meer complexe gevallen kunnen **reële getallen** gebruikt worden als metriek. Die kunnen door de netwerkbeheerder bepaald door de netwerkcapaciteit of andere karakteristieken.

Analoog kan B link-states opstellen voor zijn andere verbindingen D-B en A-B.

Iedere router voert hetzelfde proces uit.

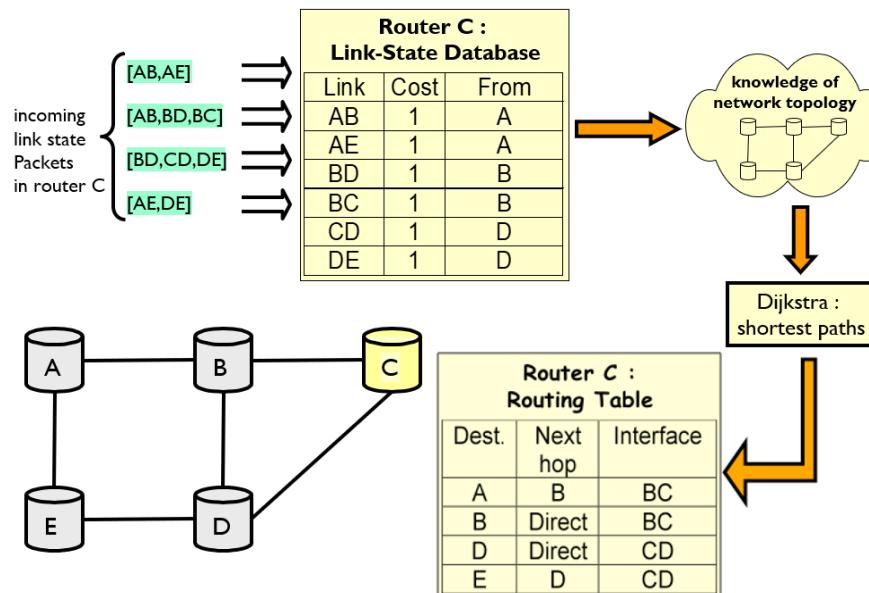
Fase 2:



In de tweede fases gaan de routers die link-states doorsturen naar zijn buren. Die buren kunnen op hun beurt die link-states verder doorsturen ofwel '**flooden**' over het netwerk. Om te voorkomen dat die link-states eindeloos rondgestuurd worden in het netwerk maakt men gebruik van **sequence numbers**. Deze zorgen er voor dat de link-state berichten worden genegeerd als ze afkomstig zijn van **dezelfde bron** en hetzelfde of ouder **sequence number** hebben als een eerder ontvangen link-state bericht. Dergelijke link-state pakketten kunnen op regelmatige basis rondgestuurd worden in het netwerk of wanneer er iets

verandert in de topologie. Op die manier kan iedere router na enige tijd een database vormen op basis van de ontvangen link-states.

Link-State Routing Protocol Overview



Router C ontvangt bijvoorbeeld informatie van de volgende links, afkomstig van andere routers.

Vervolgens gaat router C die opslaan in zijn **Link-State database** en hieruit kan hij de **volledige topologie van het netwerk** uit opbouwen.

Met de kennis van die netwerktopologie kan iedere node, alsook C, het **Dijkstra algoritme** uitvoeren om kortste paden te bepalen voor iedere bestemming in het netwerk.

Tot slot, gegeven die kortste paden kan router C, net als iedere andere router, zijn **forwarding tabel** gaan opstellen. Iedere router berekent dus de kortste paden voor zichzelf en gaat op basis daarvan de forwarding tabellen configureren. Een belangrijke aanname hiervoor is dat iedere router beschikt over **dezelfde topologie informatie** en gebruik maakt van precies **hetzelfde kortste pad algoritme**. Wanneer dit niet het geval zou zijn, kunnen er inconsistenties ontstaan in het routeren.

Wanneer er een fout optreedt in het netwerk is het tijdelijk mogelijk dat 1 router steeds beschikt over meer geupdate informatie dan andere. Over heel korte tijdspannes is het mogelijk dat netwerk lussen optreden.

Link-State Routing Protocol – OSPF (Open Shortest Path First)

Eerste keuze van netwerkprotocollen voor 1 groot systeem. Ze hebben **geen 'count to infinity'** probleem en **convergeren** in het algemeen **zeer snel**. Het nadeel is dat ze iets **meer computacioneel vermogen** van de individuele routers vergen.

OSPF (Open Shortest Path First) is een van de voornaamste Link-State Routing protocollen. Het biedt ondersteuning voor zowel IPv4 als IPv6 netwerken. OSPF maakt gebruik van **advertisements** om link-states te verspreiden. Die worden gebundeld per bron router (1 entry per buur router) en **geflood** over het hele netwerk. Links zijn niet enkel links **tussen routers**, maar ook links van routers naar **andere subnetwerken** waartoe ze toegang geven.

OSPF advertisements worden direct in een **IP packet** geëncapsuleerd zonder gebruik te maken van een Transport Layer protocol.

OSPF routing daemon

Iedere router maakt gebruik van een OSPF daemon die hij op zijn routing processor uitvoert, luisterend op IP packets (met protocol nummer 89, wat overeenstemt met OSPF). Op basis van de gevonden routes gaat OSPF de forwarding tabel van de router voorzien van de nodige configuratie.

OSPF messages

Hello

Wordt gebruikt om **aangrenzende** (buur) **routers** te ontdekken, gebruikt een **refresh** mechanisme.

Database Description

Bevat **Link-State Advertisements (LSA) headers**.

Link-State Request & Update

Mechanisme om te vragen naar een **specifiek LSA** (Link-State Advertisement) in **LSDB** (Link-State database).

Link-State Acknowledgement

Mechanisme om te erkennen (**acknowledge**).

Link-state advertisement (LSA)

Deze bevatten de **essentiële informatie over de netwerktopologie**. Er zijn twee types.

De eerste zijn **router LSA's** en worden gebruikt door de router om zijn interfaces met bijhorende kost en type te delen.

De tweede zijn **Network LSA's** en worden gebruikt wanneer meerdere routers op LAN zijn aangesloten om het designated router IP-adres, netwerkprefix en router-ID's van de routers aan te kondigen.

OSPF “advanced” features

OSPF heeft een aantal extra functionaliteiten die het aantrekkelijk maken voor grote productienetwerken.

Advertisements kunnen gebruik maken van **authenticatie mechanismen** om te voorkomen dat eerder wie een Link-State message kan introduceren in het netwerk.

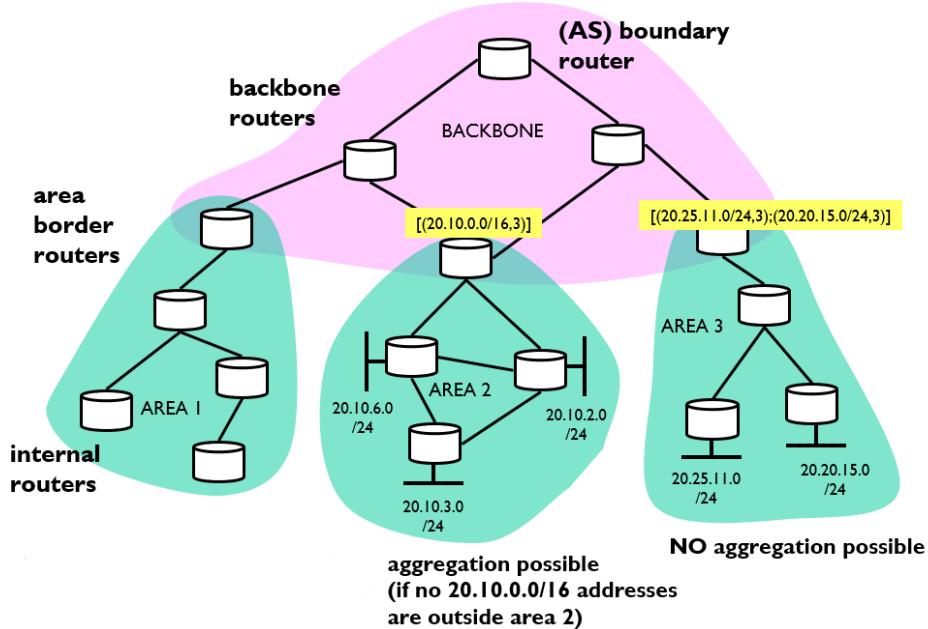
Wanneer het routing algoritme van OSPF ontdekt dat er **meerdere kortste paden** zijn, dan kunnen die geconfigureerd worden in de forwarding tabel. Tenminste als in de 'forwarding plane' of 'data plane' ondersteuning voor voorzien is.

De standaard forwarding tabellen kunnen slechts 1 entry voorzien voor een gegeven bestemming. Als een router **ECMP** (Equal-Cost Multipath-Routing) gebruikt kan de forwarding een of **meerdere entries voor dezelfde bestemming** voorzien. Een verspreidings algoritme in die router kiest dan welke entry gekozen moet worden. Dit kan dan round-robin zijn of een hash op basis van de header van het binnenkomende pakket. OSPF voorziet naast unicast, welke focust op IP verkeer van 1 bron naar 1 bestemming, ook **multicast**.

Tot slot kan OSPF gebruik maken van area's in een **hiërarchie** om het protocol schaalbaarder te maken naar autonome systemen.

Hierarchical OSPF (2-levels)

Het probleem bij grote autonome systemen, soms wel ook wel **domeinen** genoemd. Is dat het **verspreiden van Link-States** veel **overhead** bezorgt aan de routers. In een groot genoeg netwerk is er steeds wel een update van een Link-State die er voor zorgt dat de routers opnieuw paden moeten berekenen en forwarding tabellen aanpassen.



We kunnen het netwerk opdelen in **hiërarchische areas** om het protocol beter te laten schalen. Iedere area gaat enkel link-state informatie verspreiden binnen een gegeven area. Op die manier gaan Link-State advertisements niet het gehele netwerk beladen. Routers binnen een gegeven area gaan enkel met elkaar communiceren en kortste paden berekenen met bestemmingen binnen de area volgens de eerder geziene processen.

In een hiërarchische OSPF topologie van routers beschikken we steeds over een **backbone area**. De backbone van het netwerk is een partitie die de andere delen van het netwerk met elkaar verbindt. De backbone area vormt zo de top van een boomstructuur en is steeds nodig om communicatie toe te laten tussen de verschillende area's. Edge routers van die backbone area kunnen toegang geven tot andere area's. Routers (uit die area's) die geen verbinding maken met de backbone worden **internal routers** genoemd. De edge routers die wel verbinding maken met de backbone worden **area gateway routers** genoemd. Deze routers zijn verantwoordelijk voor hun area's en acteren in een **Link-State Protocol** in de backbone area alsof ze een rechtstreekse link hebben met netwerken uit de onderliggende area. De routers in de backbone area noemen we **backbone routers**. Bovenaan de backbone bevindt zich een router die mogelijk verbinding kan maken met ander autonome systemen.

Iedere area bevat routers die toegang geven tot bepaalde subnetten waar mogelijk hosts aan hangen. **Area border routers** moeten de netwerken waartoe ze toegang geven zoveel mogelijk **aggregeren** om de belasting van het netwerk en de routers te minimaliseren.

Verkeer tussen de area's moet steeds verlopen via de **backbone area**, ook wel **area 0** genoemd. Een border router kan meerdere area's bedienen en 1 area kan verschillende border routers gebruiken.

Hierarchical OSPF takeaway

- Er wordt een **twee-niveau hiërarchie** gebruikt met een backbone op het hoogste niveau.
- **Link-States** worden enkel geflood binnen een area, zodat iedere router een volledig topologie overzicht geeft van zijn eigen area, maar niet naar bestemmingen in andere area's.
- Dat **area border routers** reachability samenvatten voor hun eigen AS (autonome systemen) en adverteren naar andere area border routers via de backbone.
- **Backbone routers** draaien het Link-State Protocol enkel onder de routers van de backbone area.
- **Boundary routers** maken verbindingen met andere autonome systemen.

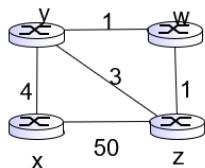
Link-State vs. Distance Vector Routing

Link State Routing	Distance Vector Routing
• E.g., OSPF, IS-IS	• E.g., RIP
• Every router knows full topology • $O(\#edges)$	• Every router knows <u>neighbors'</u> distance to <u>dest's</u> • $O(\#\text{neighbors} * \#\text{nodes})$
• Shortest Path Routing (Dijkstra)	• Shortest Path Routing (Bellman-Ford)
• Updates on link changes	• Periodic updates • Lower memory and <u>cpu overhead</u>
• Fast reaction to changes	• Slow convergence
• Usable in networks of up to thousands of nodes	• Usable in networks of up to tens of nodes (hop limit 15)

→ Het belangrijkste verschil is dat routers bij Link-State Routing overzicht hebben van de **netwerktopologie**, terwijl bij distance routing vector protocollen enkel kennis beschikbaar is over de **lengte van het kortste pad**. Bijgevolg is de **geheugenruimte** die nodig is voor Link-State Protocollen groter dan die voor Distance Vector Protocollen. Link-State **convergeert snel**, zelfs in relatief grote netwerken, maar vraagt relatief **veel resources** van de router. Distance vector routing vergt **minder resources**, maar **convergeert minder snel** en is slechts bruikbaar voor **kleinere netwerken**.

Uitleg zelftest vanaf slide 27

Vraag 1:



De kortste afstand naar x die router w doorgaat aan router z is ONEINDIG.

0 (0 %)

De kortste afstand naar x die router z doorgaat aan router w is 6.

0 (0 %)

Initieel Update proces bij update van 4->60

time	t0	t1	t2	t3	t4
Z	$\rightarrow w, D_z(x)=\infty$		No change	$\rightarrow w, D_z(x)=\infty$	
	$\rightarrow y, D_z(x)=6$			$\rightarrow y, D_z(x)=11$	
W	$\rightarrow y, D_w(x)=\infty$		$\rightarrow y, D_w(x)=\infty$		No change
	$\rightarrow z, D_w(x)=5$		$\rightarrow z, D_w(x)=10$		
Y	$\rightarrow w, D_y(x)=4$	$\rightarrow w, D_y(x)=9$		No change	$\rightarrow w, D_y(x)=14$
	$\rightarrow z, D_y(x)=4$	$\rightarrow z, D_y(x)=\infty$			$\rightarrow z, D_y(x)=\infty$

De kortste afstand naar x die router z doorgaat aan router w is ONEINDIG.

3 (100 %)

Wanneer de kost van x naar y oploopt van 4 naar 60, dan gaat zich een count to infinity probleem voordoen.

1 (33,33 %)

Wanneer de kost van x naar y oploopt van 4 naar 60, dan is de eerstvolgende kortste afstand die y gaat doorgeven aan w de afstand 9.

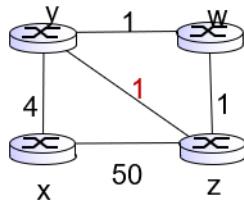
0 (0 %)

via $Z = 6 + \text{afstand naar } Z=3 \rightarrow 9$

Wanneer de kost van x naar y oploopt van 4 naar 60, dan kan zich geen count to infinity voordoen mocht de kost tussen y en z gelijk zijn aan 1.

1 (33,33 %)

→ 'Z | → w, $D_z(x) = \dots$ ' = Vanuit 'Z' naar x, wat geeft Z door aan w van kortste afstand



Update proces bij update van 4->60

time	t0	t1	t2	t3	t4
Z	$\rightarrow w, D_z(x)=5$		$\rightarrow w, D_z(x)=\infty$		$\rightarrow w, D_z(x)=50$
	$\rightarrow y, D_z(x)=\infty$		$\rightarrow y, D_z(x)=6$		$\rightarrow y, D_z(x)=50$
W	$\rightarrow y, D_w(x)=\infty$		$\rightarrow y, D_w(x)=6$		$\rightarrow y, D_w(x)=\infty$
	$\rightarrow z, D_w(x)=5$		$\rightarrow z, D_w(x)=\infty$		$\rightarrow z, D_w(x)=8$
Y	$\rightarrow w, D_y(x)=4$	$\rightarrow w, D_y(x)=60$		$\rightarrow w, D_y(x)=7$	
	$\rightarrow z, D_y(x)=4$	$\rightarrow z, D_y(x)=60$		$\rightarrow z, D_y(x)=\infty$	

→ Laatste oplossing is fout, count to infinity

Vraag 2:

Fout, want OSPF router doet een flood.

Sectie 5.4 Routing among the ISPs: structuur van het internet

In de jaren 60 ontsprong het idee om de mini netwerken tussen universiteiten hun mainframes te verbinden met elkaar. Dit vormde een soort mini internet of het toenmalige ARPANET. Door alle mainframes met elkaar te verbinden in een best-effort packet switched netwerk hoopte men destijds erg bestand te zijn tegen het al dan niet uitvallen van mainframes of de verbindingen daartussen.

Startend met het ARPANET, bestaande uit 4 sites, komen er al snel 5 nieuwe universiteiten bij in het daaropvolgende jaar. Ook in de daarop volgende 5 jaar breidt het netwerk steeds uit. Het breidt zich steeds uit naar meerdere wetenschappelijke instellingen, daarbij zelfs gebruik makende van satellietverbindingen om sites van over de oceaan van connectiviteit te voorzien. Tot op dat moment zijn alle verbindingen punt-tot-punt verbindingen die het resultaat zijn van grotendeels individuele investeringen van instellingen om toegang te krijgen tot het uitgebreide ARPANET.

In 1977 groeide het idee om vanuit een nationale organisatie vanuit de VS een soort 'backbone' netwerk te voorzien met een aantal **points-of-presence in verschillende locaties**, waar nieuwe partijen makkelijk op kunnen aansluiten, eerder dan zelf een volledig nieuwe verbinding te moeten aanleggen. Op die manier ontstond de backbone van de national science foundation of NSF in de VS.

Waar initieel slechts een handvol instellingen toegang had tot dit netwerk, groeide deze backbone snel uit tot een netwerk waartoe vele instellingen toegang kregen tot het netwerk. Waar deze NSF backbone voormalig een groot mazig netwerk vormde, verspreid over de hele VS, bestond al snel een nood om ook meer op regionaal niveau kleinere instellingen toegang aan te bieden tot het netwerk. Op die manier ontstonden kleinere regionale netwerken die verbonden waren met de NSF backbone via een fijnmaziger netwerk dat toegang bood aan kleinere instellingen. Nog later breidde die backbone zich niet alleen uit over Amerikaans grondgebied, maar ontstonden meerdere backbone netwerken over de hele wereld. Elk van die backbone netwerken gaf opnieuw toegang aan regionale netwerken die op hun beurt weer toegang gaven aan andere partijen. Op die manier groeide het internet tot een netwerk van netwerken.

Al snel hadden partijen nood aan kortere verbindingen die niet noodzakelijk de gehele hiërarchie van backbone netwerken moest doorlopen. Om deze shortcuts in de topologie mogelijk te maken, ontstonden **internet exchange points**. Dit zijn locaties waar verschillende partijen met elkaar kunnen **peeren**. Dit betekent dat ze rechtstreeks met elkaar verbinding kunnen maken op gegeven points-of-presence.

Zo'n point-of-presence ziet er uit als een moderne kruising tussen een telefooncentrale en een datacenter. Hier kunnen ISP's rechtstreeks verbindingen met elkaar aanmaken alsook grote cloud spelers zoals Google, Facebook, ... Op die manier ontstonden de autonome systemen zoals we die vandaag kennen.

Tiers

We spreken over tier-1 netwerken in de core van ons internet. Daarnaast spreken we over een 250-tal **internet exchange points** en vaak hebben die points of presence in hoofdsteden van vele landen.

Die Tier-1 netwerken geven toegang aan een aantal **regionale tier-2 netwerken**. Dit zijn vaak meer regionale internet providers (Telenet, Proximus, ...).

Tot slot hebben we de meest vermaasde toegangsnetwerken van **tier-3** providers die rechtstreeks toegang voorzien tot de eindgebruikers. Service providers kunnen zowel tier-2 als tier-3 providers zijn (Telenet).

Ieder van die netwerken wordt beheerd door een afzonderlijke partij en zijn daarom **autonome systemen**. De verschillende netwerken die het internet opbouwen vormen dus verschillende autonome systemen. In totaal zijn er momenteel ~65000 AS'en die meer dan 3.5 miljard eind toestellen met elkaar verbinden.

Content provider networks

Grote cloud spelers proberen een parallel wereldwijd netwerk aan te leggen naast het publieke internet. Grote spelers (Google, Akamai, ...) kunnen de connectiviteit tussen hun wereldwijd verspreidde datacenters voor een groot stuk zelf voorzien. Een content provider zoals Akamai heeft op vele plaatsen in de wereld kleine datacenters dicht bij de gebruiker, waardoor ze content (bv. Netflix) lokaal kunnen cachen en snel bij de eindgebruikers brengen. Dit zorgt er voor dat ze betere service kunnen leveren aan hun klanten en dus betere inkomsten kunnen genereren. Op die manier is het internet een netwerk van autonome systemen. Ieder **autonomus systeem (AS)** maakt gebruik van z'n eigen routing protocol zoals OSPF of RIP, maar kan ook manueel geconfigureerd worden. Ieder AS krijgt een **nummer** dat maximaal 32 bits lang kan zijn.

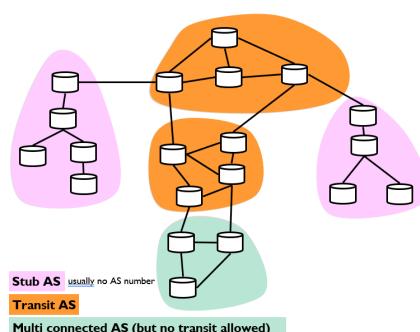
Via 'whois' kan je opzoeken welke partij hoort bij een gegeven AS nummer.

Er is een verschillende term voor verschillende soorten autonome systemen.

Een AS die maar via 1 connectie verbinding maken met de rest van het internet, een **stub AS**. Deze AS'en hebben meestal geen nummer toegewezen gekregen.

Een AS dat twee of meerdere AS'en met elkaar verbindt wordt een **transit AS** genoemd.

Tot slot spreken we over een **multi connected AS** wanneer het AS via meerdere verbindingen toegang geeft tot andere AS'en. Ondanks die meerdere verbindingen gaat dergelijk AS nooit doorgaand verkeer behandelen, maar dienen de verschillende verbindingen enkel als **backup** mocht 1 verbinding falen. Dergelijke setup wordt vaak gebruikt door partijen zoals banken die zeker willen zijn van hun bereikbaarheid.

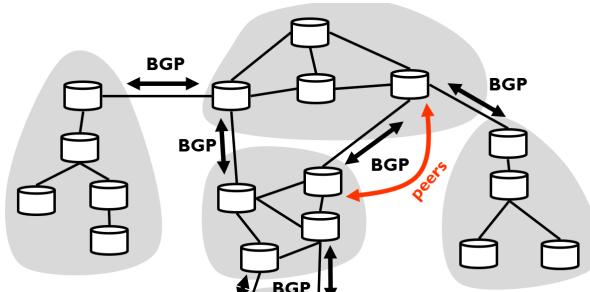


In dit internet landschap van verschillende soorten AS'en moet een inter-AS routing protocol zoals **BGP** nu **routes** ontdekken naar verschillende routers in het internet.

Sectie 5.4 Routing among the ISPs: BGP

Eerder zagen we dat een inter-AS routing protocol zoals OSPF gebruikt wordt om routes te ontdekken en verspreiden binnen een autonoom systemen. Om routes te ontdekken tussen andere autonome systemen wordt **BGP (Border Gateway Protocol)** gebruikt.

BGP for inter-AS routing



Enkel die routers die verbinding maken met **andere AS'en** maken gebruik van het BGP protocol, die routers zijn **border gateways**. Als twee border gateways tegen elkaar spreken, dan wordt dit '**peers**' genoemd.

Voor Stub AS'en weten we dat die 1 verbinding naar het internet hebben. In dat geval kunnen we een **default gateway** voorzien dat alle internetverkeer via die gateway verstuurt. Op die manier moeten we **geen BGP** gebruik voor **Stub AS'en**.

BGP basics

BGP maakt gebruik van TCP in de transportlaag (poort 179) om routers met elkaar BGP te laten spreken. BGP hoeft niet noodzakelijk plaats te vinden tussen **rechtstreeks verbonden routers**.

BGP is een **path vector protocol**. Het erft enkele zaken van distance vector protocollen, maar vermeidt de grootste problemen ervan. Een path vector protocol zoals BGP wisselt **geen vectoren van afstanden** naar bestemmingen uit, maar wisselt **paden naar bestemmingen** uit. Een **bestemming** in deze context is een **subnet** dat voorgesteld wordt door zijn netwerkprefix.

Een groot **voordeel** van BGP is dat we **andere paden** toelaten dan kortste paden. Dit is voornamelijk nuttig wanneer je gebruik maakt van **routing policies**.

Er zijn 4 soorten BGP berichten: (1) OPEN, opzetten van verbinding en afspraken maken voor de mogelijkheden van beide partijen. (2) UPDATE, network reachability informatie adverteren, bijwerken, terugtrekken, ... (3) NOTIFICATION, bv. bepaalde verbinding/categorie van reachability informatie niet meer beschikbaar. (4) KEEPALIVE

BGP principles (3 principles)

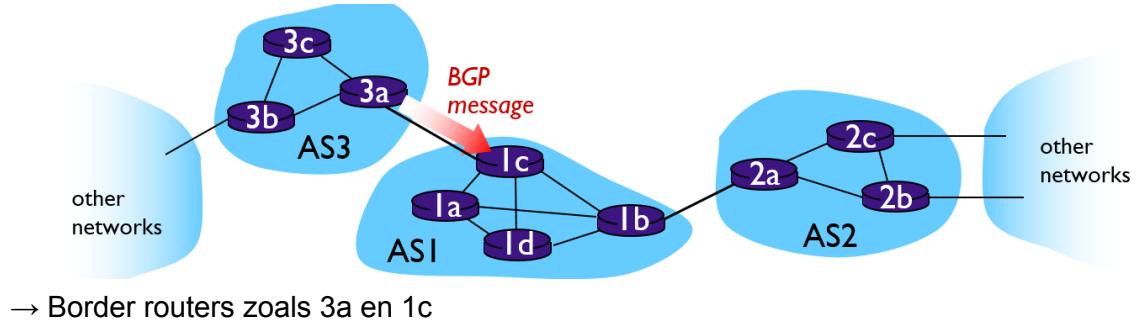
Een border gateway router moet dus leren hoe bestemmingen in andere netwerken of andere AS'en bereikt worden. De basisprincipes van BGP zijn eenvoudig. BGP maakt gebruik van 3 principes.

- (1) Routers gaan **adverteren** tot welke **subnetten of prefixen** ze toegang bieden, ze gaan m.a.w. hun **reachability informatie** doorgeven aan andere routers.

- (2) Routers gaan ontvangen reachability informatie **propageren**/doorsturen naar andere BGP routers.
- (3) BGP routers kunnen ontvangen **paden/routes** gaan **selecteren** aan de hand van een van hun eigen voorkeuren/**policies**.

Principle 1: Advertisement of reachability

Omdat BGP routers paden moeten leren, moeten andere routers die informatie eerst verspreiden. Welke router gaat hier aan participeren?



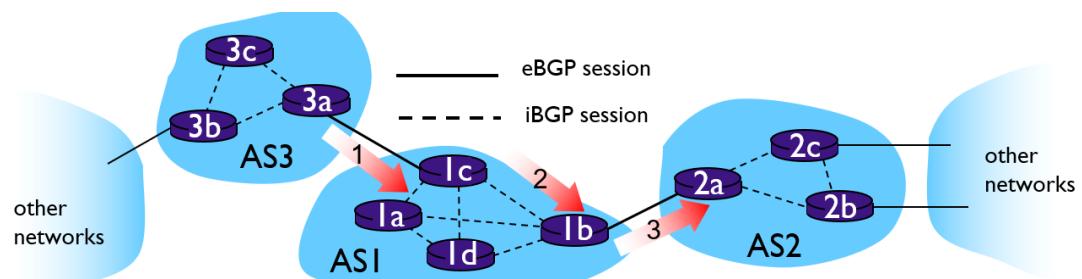
Iedere BGP router gaat aangeven welke **netwerk prefixen** het kan bereiken.

Wanneer bv. router 3a zijn reachability informatie doorgeeft aan 1c is dat impliciet een **beloofte**. Het is een belofte dat wanneer router 3a pakketten ontvangt van 1c, hij ze zal afleveren op hun bestemming.

Om overhead te beperken gaan BGP routers zoveel mogelijk netwerkadressen **aggregeren**.

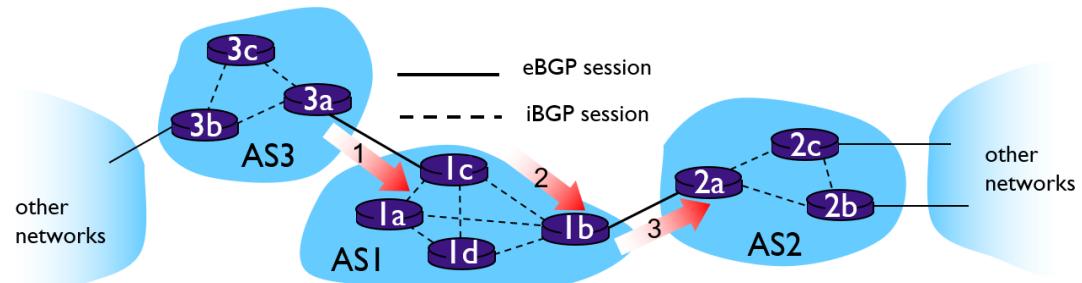
Zo'n advertisement bevat een path vector (**AS-PATH**) naar de netwerkprefix. De path vector definieert welke tussenliggende AS'en moeten genomen worden om tot bij een gegeven netwerk te geraken. De **hops** in deze context zijn geen routers, maar **autonome systemen**. Ook wordt de **volgende hop (NEXT-HOP)** die moet genomen worden om dit netwerk te bereiken meegegeven.

Principle 2: Propagation



Hoe border gateway routers (zoals 1b) die niet rechtstreeks verbonden zijn met het andere AS hiervan op de hoogte geraken. Hiervoor maakt BGP gebruik van twee soorten sessies. We spreken over **eBGP (exterior BGP)** voor sessies van routers tussen **verschillende AS'en**. **iBGP (interior BGP)** wordt gebruikt om **binnen hetzelfde AS een full-mesh** op te zetten tussen alle gateway border routers. Dit betekent dat in een AS alle mogelijke koppel routers een iBGP sessie met elkaar hebben opgezet.

eBGP & iBGP interaction

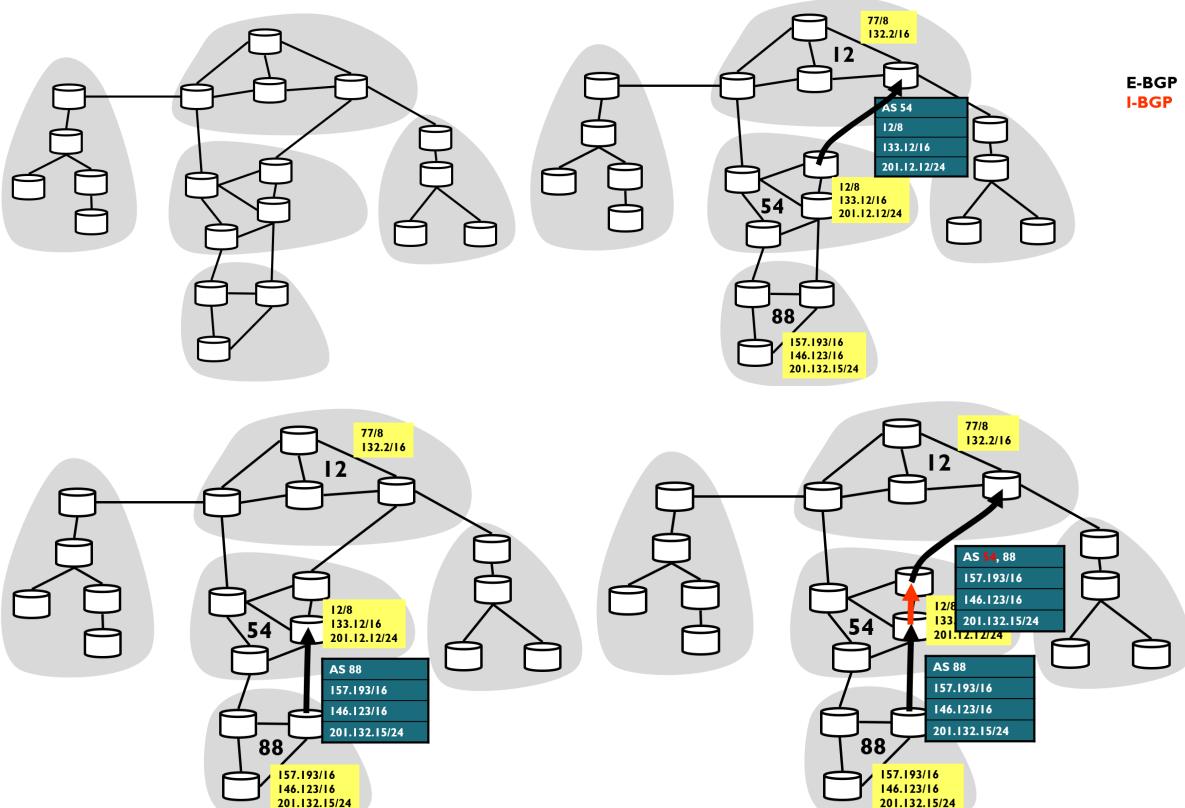


iBGP zorgt ervoor dat alle routes die via eBGP van een ander AS geleerd worden, doorgesluisd worden naar de andere border gateway routers van hetzelfde AS. Op die manier ontstaat interactie tussen eBGP en iBGP in het **propagatie proces**.

(1) In eerste instantie gaat een router (bv. 3a) reachability **informatie doorsturen** naar bv. 1c van zijn aangrenzend autonoom systeem. (2) Nadien wordt die **informatie gespiegeld** via iBGP naar alle andere **border gateway routers** van het AS (bv. 1b). (3) Tot slot gaan de andere border gateway routers (bv. 1b) opnieuw hun **informatie verspreiden** naar andere AS'en via eBGP.

Tekens wanneer een BGP router een nieuwe prefix leert, kan die een entry toevoegen aan zijn forwarding tabel. Het is duidelijk dat die tabellen in de context van het internet snel heel groot worden.

Propagation example

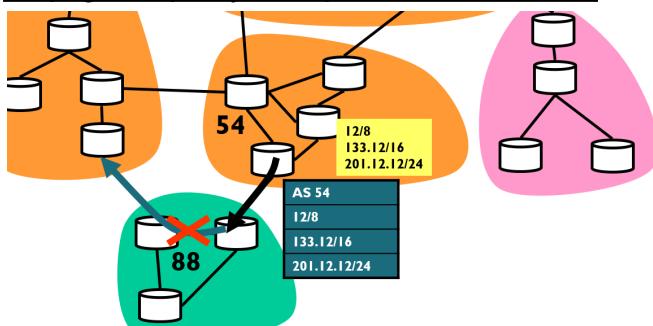


Propagation policies

Ieder AS heeft de keuze om **ontvangen reachability informatie al dan niet door te sturen** naar andere AS'en. Hiervoor kan gebruik gemaakt worden van **propagation policies**. Deze kunnen gedreven zijn door commerciële doeleinden. Een ISP horende bij een gegeven AS kan bv. enkel reachability informatie doorspeLEN aan bepaalde klanten en niet naar andere anderen, en niet naar concurrenten.

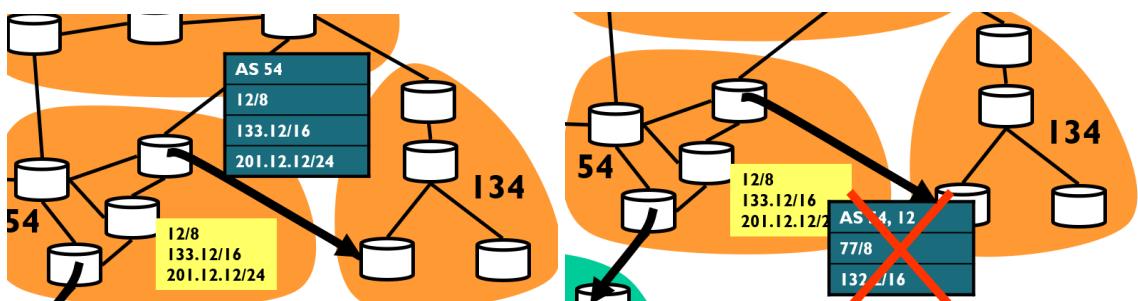
Wanneer je geen reachability informatie doorspeLT naar andere AS'en, geraken die niet op de hoogte dat er mogelijks een **pad** is **naar** die netwerken. Bijgevolg gaan die het netwerk niet gebruiken om bestemmingen te bereiken. Dergelijke policies geeft je als AS de controle over hoe het netwerk zal gebruikt worden.

Propagation policy example – dual-homed AS



→ AS 88 is hier een dual-homed AS, maar doet niet aan propagation, dus zal de reachability informatie niet doorgeven.

Een **dual-homed AS** = een met twee links.



→ AS 54 is hier een transit AS. Het kan wel kiezen de reachability informatie van 12 (bovenaan) niet door te geven aan bepaalde AS'en zoals hier bv. 134.

III. Route selection

Border gateway routers kunnen verschillende routes ontvangen van een gegeven netwerk. 1 die bv. gebruik maakt van AS pad '1, 2, 3' en een andere dat gebruik maakt van een pad AS pad '4, 5, 6'. Dit biedt mogelijkheden om enkel die routes te selecteren waarvoor dat gewenst is. BGP maakt hiervoor gebruik van een aantal **selectieregels, in volgorde van prioriteit**.

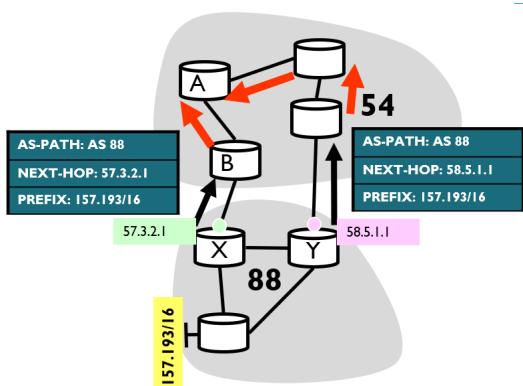
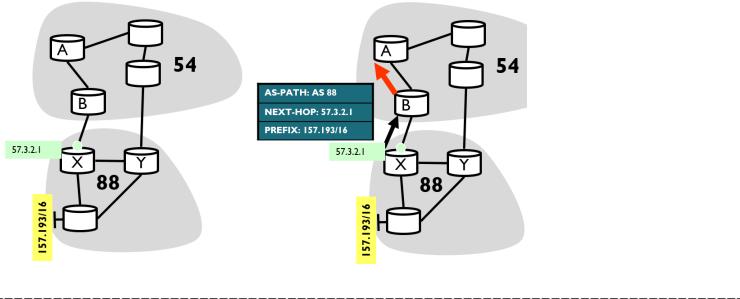
Als eerste attribuut wordt gekeken naar een **local preference** value die hoort bij gegeven AS'en. Men kan dus bijvoorbeeld als netwerk beheerder aangeven dat ontvangen routes van AS 1 gekozen worden boven die van AS 2.

Wanneer er geen verschil is in de local preference wordt gekeken naar de **lengte** van de **ontvangen paden**. Hier wordt bij voorkeur het **kortste pad** gekozen in termen van het aantal hops of tussenliggende AS'en (**shortest AS-path**).

Wanneer ook de padlengte van verschillende routes gelijk is wordt gekeken naar de **dichtstbijzijnde volgende hop** die gekozen moet worden. hoe dichter, hoe beter. Dit wordt ook wel 'hot potato routing' genoemd.

Mogelijks kan je ook nog andere criteria specifiëren.

Route selection example

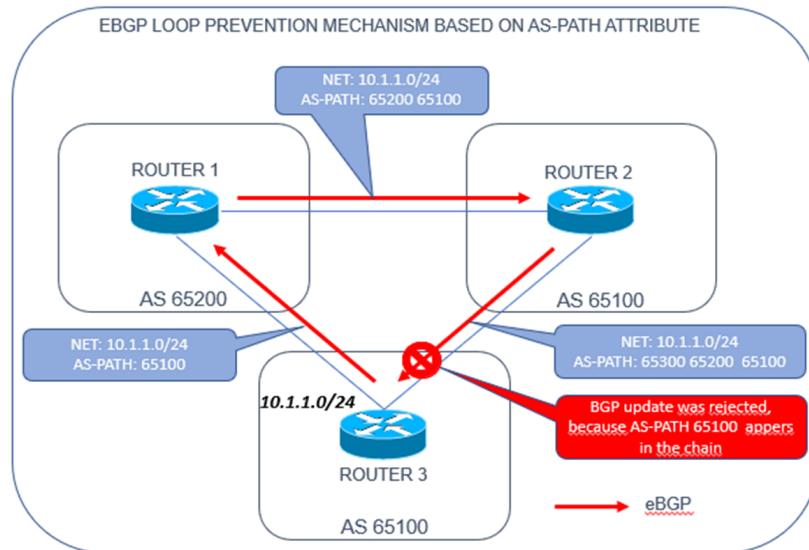


→ Stel dat de onderste router uit AS 88 toegang biedt tot het gegeven /16 netwerk. Vanuit AS 54 kan dit bereikt worden via border gateway router X als next-hop. Deze reachability informatie is te zien op de tweede afbeelding. Als alternatief kan de andere border gateway router Y alternatieve reachability informatie doorspelen. Hierin geeft hij aan dat de bijbehorende interface van router Y als next-hop kan gebruikt worden. Op die manier ontvangt via iBGP router A, twee routes voor dezelfde bestemming. Vermist de info werd ontvangen via hetzelfde AS 88, speelt de local preference geen rol in de selectie van de route. → De criteria die gebruikt zal worden om te beslissen welke route zal gekozen worden is op basis van de dichtstbijzijnde next-hop. Via het gebruikte intra AS routing protocol

(iBGP) leert router A dat de groene interface een korter pad oplevert als next-hop en daarom zal het pad langs X gekozen worden.

Response college

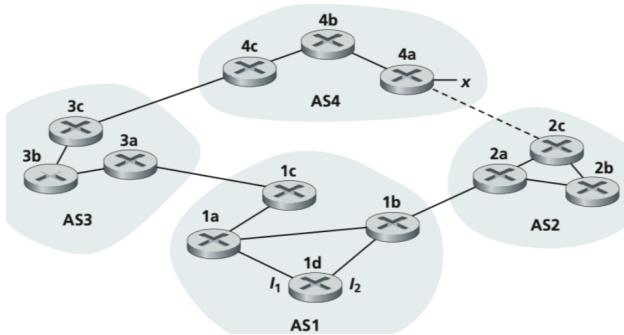
- 1) Geef aan welk van de volgende uitspraken correct is m.b.t. BGP, subnetten en prefixen.
 - a) Subnet is deel van een groter netwerk
 - b) Alle interfaces van een router behoren tot een gegeven subnet.
→ Routers verbinden verschillende subnetten → iedere interface behoort tot verschillend subnet
 - c) Een gegeven prefix kan 1 of meerdere prefixen omvatten.
→ d.m.v. aggregatie
 - d) Wanneer een router een prefix adverteert in een BGP sessie, dan kan dit nooit een geaggregeerd prefix zijn.
→ Gaat dit juist wel doen om zo min mogelijk berichten te moeten uitwisselen met een ander systeem
- 2) Waartoe kan het AS-PATH attribuut bij BGP advertisements gebruikt worden?
 - a) AS-PATH geeft de opeenvolging van te nemen routers naar een gegeven subnet (prefix) terug vanaf de adverteerende BGP router
→ routers → AS'en
 - b) Het AS-PATH attribuut kan gebruikt worden door routers om te detecteren of er lussen in het aangekondigde pad naar een subnet te vinden zijn.



- c) Het ontvangen van meerdere AS-PATHs voor een gegeven bestemming laat BGP routers toe om te kiezen welke route ze prefereren en configureren in hun forwarding plane.
- d) Aan de hand van het AS-PATH attribuut, weten routers welke volgende hop of router ze moeten nemen naar een gegeven subnet/prefix/
→ NEXT-HOP attribuut doet dit

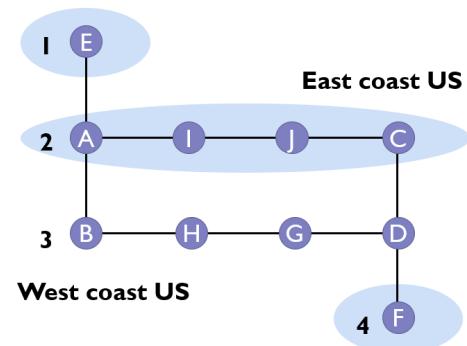
- 3) Welk van de volgende uitspraken met betrekking tot BGP policies zijn correct?
- Wanneer een BGP-router een pad ontvangt, dat werd bekendgemaakt door zijn buur, moet het de identiteit van zijn eigen AS aan het ontvangen pad toevoegen en dit nieuwe pad doorsturen naar al zijn buren.
 - Als een AS - of BGP router van een AS - een pad naar een gegeven subnet/prefix adverteert naar zijn buren, dan belooft hij in principe om ontvangen verkeer naar die bestemmingen te routeren.
 - Een tier 1 ISP heeft de keuze om doorgaand verkeer tussen andere tier-1 ISPs al dan niet toe te laten. Om dit te implementeren kan hij ervoor kiezen om routes naar gegeven bestemmingen niet te adverteren, waardoor verkeer voor die bestemmingen niet door het beschouwde AS kan verlopen.
→ Principle 2: Propagation
 - BGP policies kunnen ervoor zorgen dat de route naar een gegeven bestemming (subnet of prefix) vele malen langer is dan het kortste pad naar die bestemming.

4) Netwerkprotocol

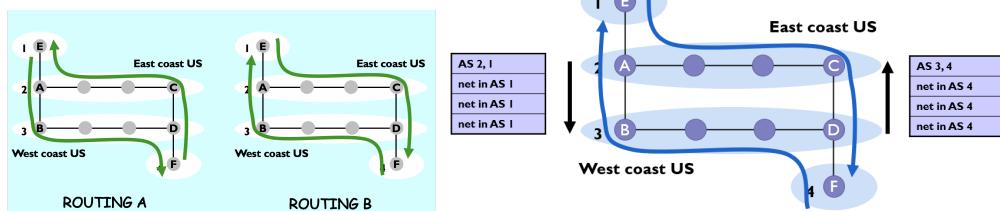


-
-
-
- Van welk routeringsprotocol ontvangt router 1d informatie over prefix x?
→ iBGP/IGP
-
- Wanneer router 1d informatie over x ontvangt, zal hij een record (bestemming=x, uitgaande link=I) in zijn forwarding tabel opnemen. Zal I in dit geval L1 of L2 zijn?
→ L1, kortste pad (stippellijn betekent systemen tussen)
- Stel dat er tussen AS2 en AS4 wel een fysieke link aanwezig is (stippellijn) en dat router 1d zowel informatie ontvangt en ontdekt dat x zowel bereikbaar is via AS2 als via AS3. Zal L dan L1 of L2 worden?
→ L2, dezelfde AS-PATH lengte, maar 2a ligt dichter
- Stel dat er een ander AS, AS5, is op het pad tussen AS2 en AS4 (niet op de figuur weergegeven). Stel dat router 1d informatie ontvangt en ontdekt dat x zowel via het pad AS2, AS5, als via het pad AS3 bereikbaar is. Zal L dan L1 of L2 worden?
→ L1, kortste AS-PATH

- 5) Gegeven het volgende netwerk voor gebaseerd op de VS. ISP horend by AS2 levert een nationale backbone dienst aan ISP met AS1. AS1 is verbonden met subnet x, AS4 is verbonden met subnet y. Routers E en A zijn peers en maken gebruik van BGP. Stel dat er dataverkeer is van AS1 naar AS4. De ISP van AS2 zou willen dat het verkeer verloopt via de AS3 via westkust (zodat de ISP van AS3 de kosten van het transport naar de andere kant van het land voor zijn rekening neemt), terwijl de ISP van AS3 liever heeft dat het dataverkeer via het knooppunt met AS2 aan de oostkust loopt (zodat ISP van AS2 het dataverkeer naar de andere kant van het land voor zijn rekening neemt).



- a) Gegeven de volgende 2 routeringsscenario's: ROUTING A en ROUTING B). In het scenario A worden verkeer van AS1 naar AS4 via de westkust getransporteerd en verkeer van AS4 naar AS1 via de Oostkust. Welke van de twee opties is mogelijk te bereiken aan de hand van BGP ?



- → Je kan enkel informatie verschaffen van waar verkeer mag binnenkomen vanaf andere transit AS. Als ook informatie van B naar A zou worden gegeven, zou alle verkeer via AS 3 gaan.
- b) Welk subnet (prefix) adverteert BGP router A naar BGP router B ("X" of "Y")?
→ X
- c) Welk pad adverteert BGP router A naar BGP router B voor het geadverteerde subnet? Antwoord zonder spaties in de vorm van "[AS3,AS1]" (hierbij is AS1 de eerste hop, en AS3 de tweede)
→ [AS1,AS2]
- d) Welk netwerkprefix adverteert BGP router D naar BGP router C ("X" of "Y")?
→ Y
- e) Welk pad adverteert BGP router D naar BGP router C voor het geadverteerde subnet? Antwoord zonder spaties in de vorm van "[AS3,AS1]" (hierbij is AS1 de eerste hop, en AS3 de tweede)
→ [AS4,AS3]
- f) Wanneer traceroute uitgevoerd wordt van een host in subnet y naar een host in subnet x, welke tussenliggende hops worden dan gedetecteerd? Antwoord met hoofdletters in de vorm van "[A,B,C,D]".
→ [D,G,H,B,A,E]

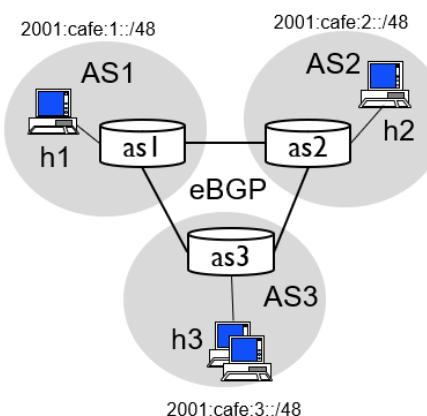
BGP in practice

Voor met video te combineren!

1. Start Mininet topology
2. Inspect configuration
 - as1r
3. Check routes:
 - from h1 to h3 and h3 to h1 and
4. Inspect BGP state in as2r2
 - Neighbors
 - RIB
5. Wireshark on link
 - AS1-AS2

Peering relationship:

- AS1 is paying client of AS2
- AS2 is paying client of AS3
- AS1 is equal peering of AS3



Sectie 5.5 Programming the network

Programmability

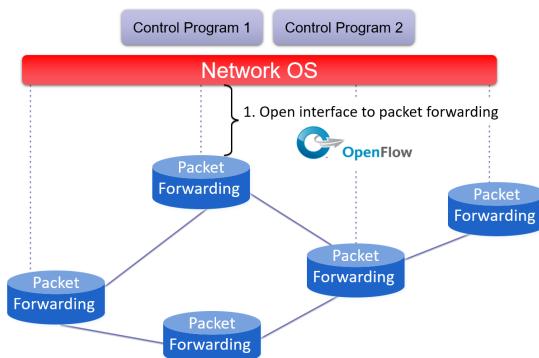
Splitsing van **control & data plane** door gebruik te maken van open APIs.

Geen nood aan een groot proces zoals in een traditioneel computernetwerk. Dit is het grote voordeel van die ontkoppeling door SDN. Er kan een design worden opgesteld, een implementatie in controle software worden gemaakt, getest in een ge-emuleerd netwerk, en in productie genomen.

Software-Defined Networks

De volledige controle van het netwerk in **software** schrijven en draaien op een **gecentraliseerde controller**. Die is verantwoordelijk om alle routers en switches te configureren.

Hiermee stappen we af van het **traditionele gedistribueerde controlemodel**: eigen forwarding hardware, OS en functionaliteiten, specifiek van de fabrikant. Aan de hand van bv. OSPF kunnen ze hun eigen forwarding configureren.



We gaan het stuk controlesoftware loskoppelen van de routers en gaan centraliseren in 1 controller systeem. Die controller staat in verbinding met elke switch of router.

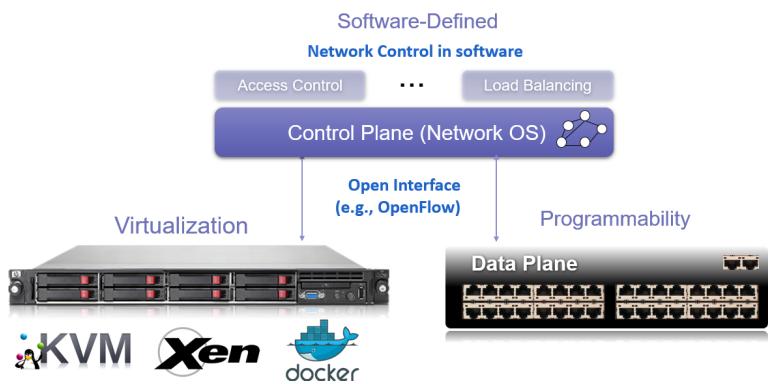
De **open interface** naar de **data plane** speelt hier een cruciale rol in. Een bekend voorbeeld hiervan is OpenFlow.

Bovenaan de controller is er een soort **netwerk besturingssysteem** dat een aantal **basisfuncties** voor de gebruiker implementeert.

Bv. de netwerktopologie, ieder van de **controle applicaties** kan dan gebruik maken van de topologie informatie.

Network Virtualization

Data plane in software



Sectie 5.5 Software Defined Networking (SDN): Programmability + Software Defined Networks

Het probleem

Traditionele computernetwerken zijn **duur en moeilijk te beheren**, bevatten vaak **configuratiefouten** en zijn moeilijk te **upgraden**.

Dit is gekomen doordat het internet is gegroeid uit een klein netwerk, het ARPANET. De principes van toen zijn vrijwel hetzelfde gebleven. Netwerk nodes/routers bestaan uit een **data plane** dat zich enkel moet bezighouden van het doorsturen van data pakketten op basis van een forwarding tabel. Iedere node gaat daarin pakketten ontvangen, filteren, bufferen en eventueel doorsturen. Het **control plane** van het netwerk moet ervoor zorgen dat routers informatie met elkaar uitwisselen zodat ze routes kunnen bepalen om de forwarding tabel van nodes in te vullen. Het **management plane** van een netwerk voorziet functionaliteit dat beheerders kunnen invullen op hun verschillende netwerk nodes om die te configureren, metingen te vergaren of problemen op te lossen.

De evolutie

Helaas evolueerde de service die het internet moest gaan bieden doorheen de jaren naar een veel complexere combinatie van zaken dan enkel pakketten doorsturen. Netwerken werden steeds voor specifieke doeleinden gebruikt en brachten nieuwe eisen met zich mee (nieuwe **control requirements** → grotere complexiteit).

In de plaats van 1 globaal netwerk, hebben we nu ook verschillende logische netwerken aan de hand van VLAN's en andere, die elk apart ook nog eens gemanaged moeten worden. Daarboven proberen we ook allerhande access control lists toe te voegen die bepalen wie het netwerk mag gebruiken en wie niet.

Traffic engineering maakt onze routering complexer en moeten toe gaan laten dat beheerders heel specifiek kunnen kiezen welke paden voor welke bestemmingen gebruikt moeten kunnen worden. (*Voorbeelden hiervan zijn MPLS en EXCP. Dit zijn technologieën die toelaten om enerzijds paden te reserveren aan de hand van labels of anderzijds toe te laten dat meerdere paden met dezelfde kost kunnen gebruikt worden.*)

NAT devices, firewalls en anderen hebben het leven van een netwerkbeheerder er niet makkelijker op gemaakt.

Veel van die technologieën waren dan ook los van elkaar ontwikkeld en creëren een heel complex control plane. Dit staat in schril contrast met het originele modulaire 'data plane'.

Klassieke router architectuur



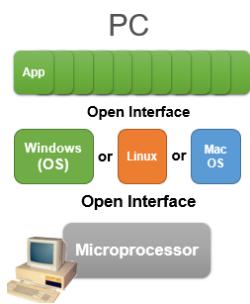
→ Verschillende **routerings en management protocollen** die efficiënte en configurerbare routering moeten toelaten. Over verschillende (logische) netwerken heen.

→ Iedere router draait zijn **eigen OS** (verschillend per merk)

→ **Gespecialiseerde forwarding hardware** die snel tientallen megabytes werkgeheugen en vele poorten bevat om zijn werk te kunnen doen, pakketten doorsturen.

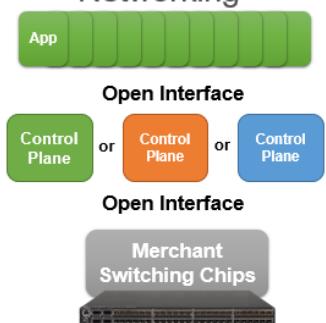
Het gebruik van standaarden voor al die protocollen zorgt er voor dat toestellen van verschillende fabrikanten zonder problemen met elkaar kunnen **samenwerken**. Aan de andere kant maken de vele standaarden het ook moeilijk om snel **nieuwe functionaliteiten** in het netwerk te krijgen. Dit laatste is een probleem voor de grote **cloud** spelers (Google, Amazon, ...). Hun business model is net om snel vernieuwende services op de markt te brengen en die zijn dan vaak afhankelijk van het onderliggende netwerk.

Dit herhaalt de geschiedenis van de mainframes. Destijds werden die mainframes slechts door enkele fabrikanten beheerd of gefabriceerd. Ze voorzagen specifieke hardware met hun eigen besturingssysteem en hun eigen applicaties. Als klant hing je vast aan die spelers. Dit veranderde snel met de komst van de PC met open-interfaces.



Nieuwe architectuur

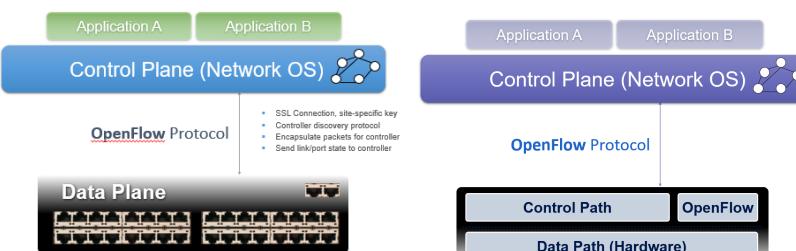
Software-Defined Networking



Diezelfde situatie konden we een aantal jaar geleden herkennen in het landschap van de netwerkhardware. Er is een gesloten systeem met fabrikant specifieke hardware, besturingssysteem en applicaties. Het kernidee van **SDN (Software Defined Networking)** bestaat erin om net als bij pc's die netwerk interfaces **open** te maken, zodat iedereen snel zijn eigen OS en applicaties kan schrijven.

Onderaan dit model hebben we **goedkope switches** met daarop een **control plane** besturingssysteem en daarop **applicaties** die we zelf kunnen programmeren.

Openflow (protocol)



Dit is het protocol dat gesproken wordt tussen de **control plane** en het **data plane**, ofwel het protocol dat je kan spreken met pure switching hardware om het te configureren.

De **control plane** bestaat in een SDN architectuur uit 1 of meer **gecentraliseerde controller(s)**. Je kan dit zien als een krachtige server in een datacenter die interageert met vele switches. Hij kan dit doen aan de hand van een aparte fysische verbinding, maar kan ook gebruik maken van het netwerk tussen de switches.

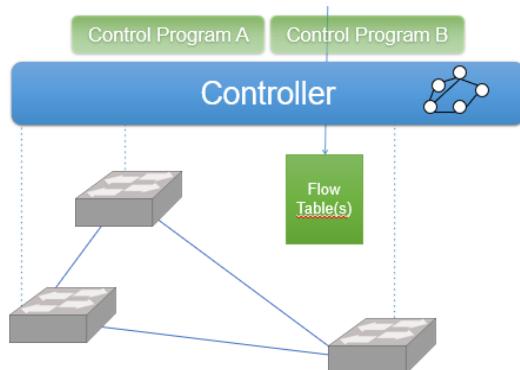
OpenFlow maakt gebruik van een **SSL** verbinding met de **OpenFlow agent** van de switch, die een aantal eenvoudige berichten kan uitwisselen met de controller. Op basis daarvan kan de controller een beeld krijgen van het netwerk en zijn eigen logica gaan voorzien. Om dat hergebruik toe te laten is ook de controller uit **verschillende lagen** opgebouwd. Een

aantal **basisfunctionaliteiten** worden door het netwerk-OS aangeboden, zoals bv. het ontdekken van de netwerktopologie. Op basis van die basisfunctionaliteiten kunnen dan **verdere of nieuwe applicaties** gebouwd worden.

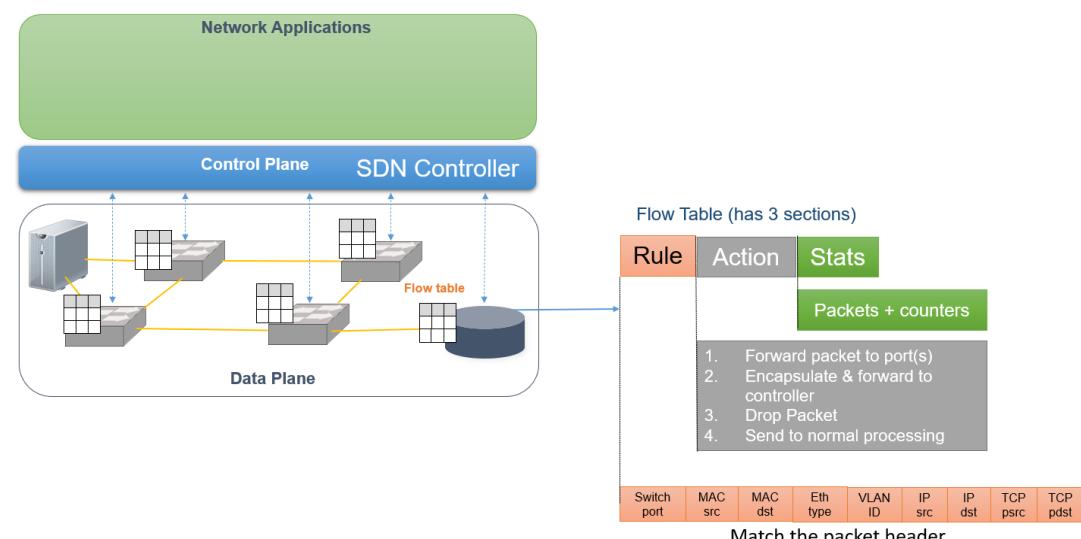
Openflow forwarding abstraction

Alle **configuratie in een programmeerbare switch** wordt door OpenFlow geabstraheerd in een **flow table**. De **controller** kan die flow table gaan **configureren** aan de hand van een OpenFlow instructie. Bijvoorbeeld "als de header van een inkomend pakket de waarde 'p' heeft, zendt dan door naar poort x". "Als de header 'q' bevat, overschrijf header met 'r' en voeg header toe, stuurt naar poort x,y".

De instructies lijken eenvoudig, maar ze maken heel veel mogelijk voor een netwerkbeheerder.



OpenFlow data plane



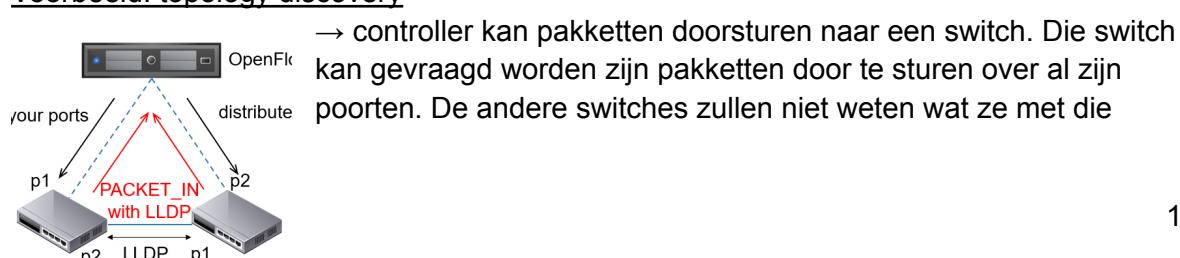
OpenFlow definieert de interface tussen de controller en de switch, maar steunt daarbij ook op een abstractie van de switch. Een regel in zo'n **programmeerbare flow table** heeft drie onderdelen.

Een **matching regel**, op eender welke laag 2 tot laag 4 velden.

Een **actie**, die kan eruit bestaan om een pakket door te sturen naar een of meerdere poorten (1), een pakket te encapsuleren en te forwarden naar een controller (2), of het pakket te droppen (3).

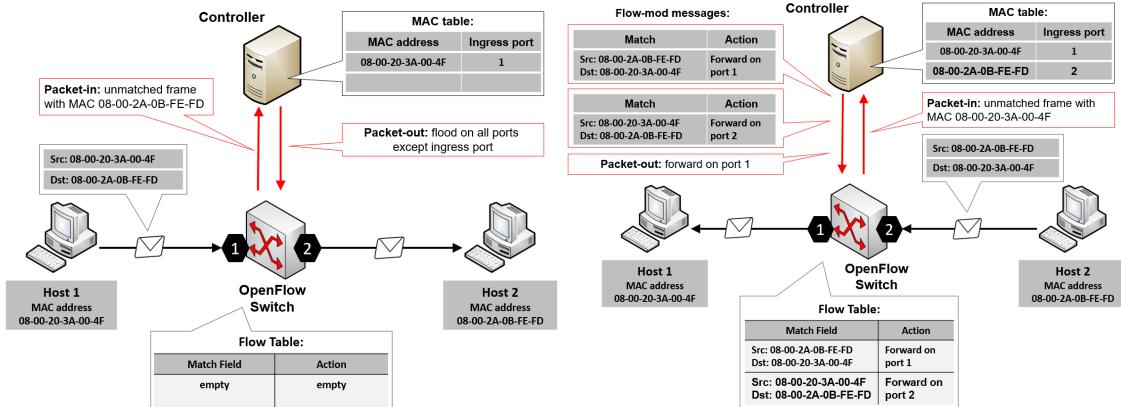
Voor iedere regelen wordt aan aantal **statistieken** bijgehouden, hoeveel keer werd die regel toegepast,

Voorbeeld: topology discovery

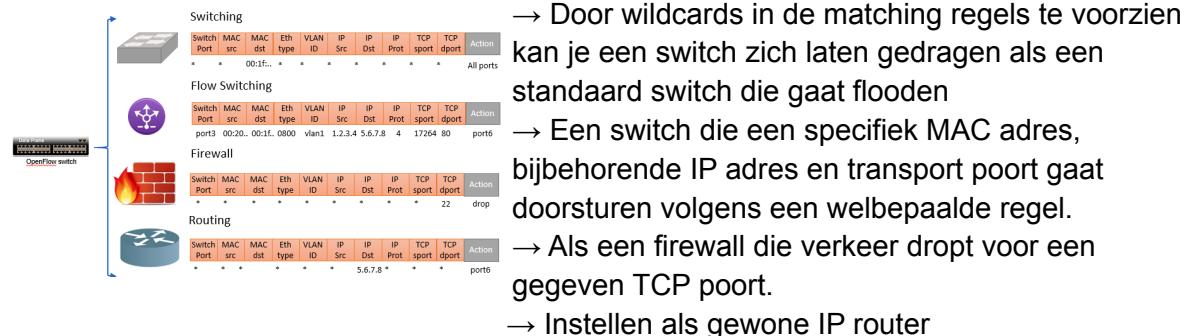


ontvangen pakketten moeten doen. Er is namelijk geen flow geconfigureerd in zijn tabel. Daarom gaat die de pakketten opnieuw doorsturen naar de controller. Door dit mechanisme kan de controller de links in het netwerk ontdekken en een overzicht krijgen.

Voorbeeld: MAC learning in controller



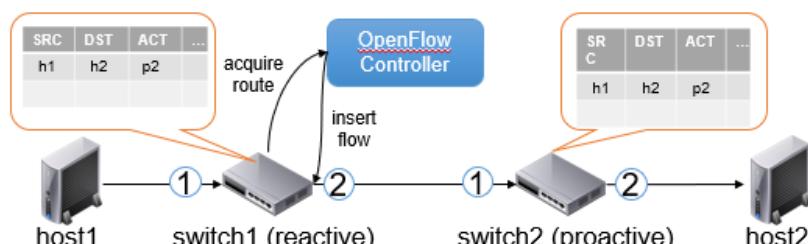
Op basis van deze eenvoudige abstractie en OpenFlow interface kon een "goedkope witte doos" met switching hardware ingezet worden als programmeerbaar netwerk toestel.



- Door wildcards in de matching regels te voorzien kan je een switch zich laten gedragen als een standaard switch die gaat flooden
- Een switch die een specifiek MAC adres, bijbehorende IP adres en transport poort gaat doorsturen volgens een welbepaalde regel.
- Als een firewall die verkeer dropt voor een gegeven TCP poort.
- Instellen als gewone IP router

Control mechanisms - flow insertion

Het voorziet twee mechanismen om flows toe te voegen aan een tabel.



Reactive flow insertion

Hierbij zal pas een regel aan de flow table toegevoegd worden **nadat** er eerst een pakket van die flow werd ontvangen in een switch. Dit eerste pakket wordt immers doorgestuurd naar de controller die op basis daarvan een nieuwe regel toegevoegd in de flow table van de switch. Vervolgens gaat de switch alle pakketten van diezelfde flow behandelen volgens diezelfde regel.

Proactive flow insertion

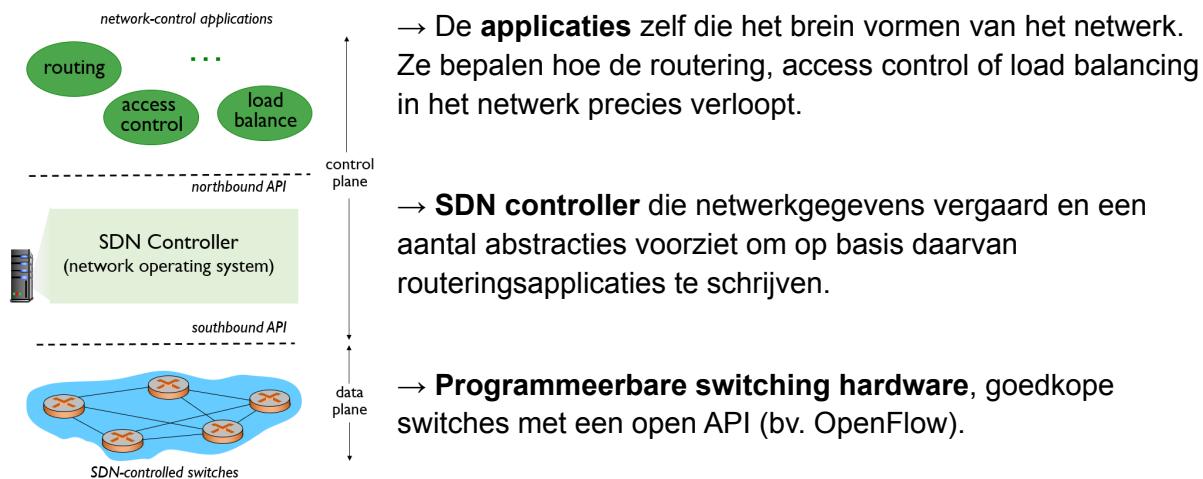
De switch kan **proactief** regels configureren in de flow table, zonder dat een pakket reeds

Robin De Bock

werd ontvangen in de switch. Zo kan ook het eerste pakket van die flow **meteen** in de data plane afgehandeld worden.

SDN architectuur componenten

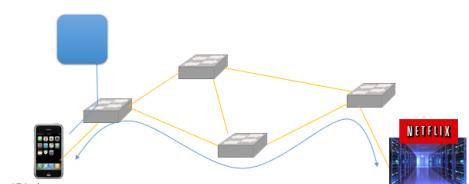
De netwerkarchitectuur bestaat uit 3 delen.



Dit zijn allemaal zaken die nu in software geschreven kunnen worden en aangepast aan eigen eisen.

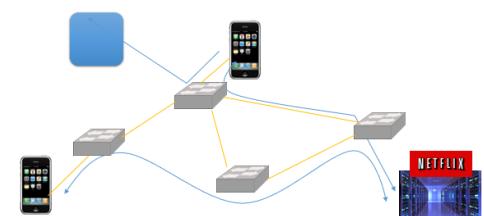
Applications

Dynamic Access Control



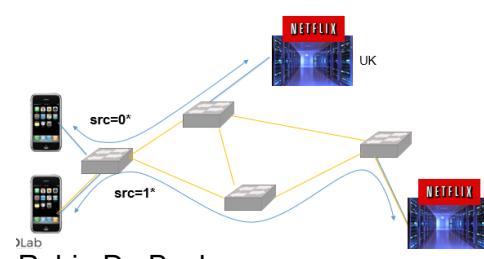
Een mogelijke controle applicatie kan er bijvoorbeeld uit bestaan om op basis van het eerste pakket die je krijgt van een end-device, forwarding regels toe te voegen in je netwerk. Zo kan je een eerste security check doen en op basis daarvan als alles oké is, je netwerk configureren om verkeer van dat specifieke toestel verder toe te laten in je netwerk en te routeren volgens een bepaalde route.

Reroute upon Mobility



Een mobiel apparaat kan je detecteren op een nieuwe locatie en verkeer kan rerouted worden volgens de policy van de controller die je instelt.

Server Load Balancing



Load balancing kan geïmplementeerd worden op basis van het source IP adres. Iets dat niet eenvoudig is in traditionele IP routering.

Sectie 5.5 Programming the network: Netwerkvirtualisatie

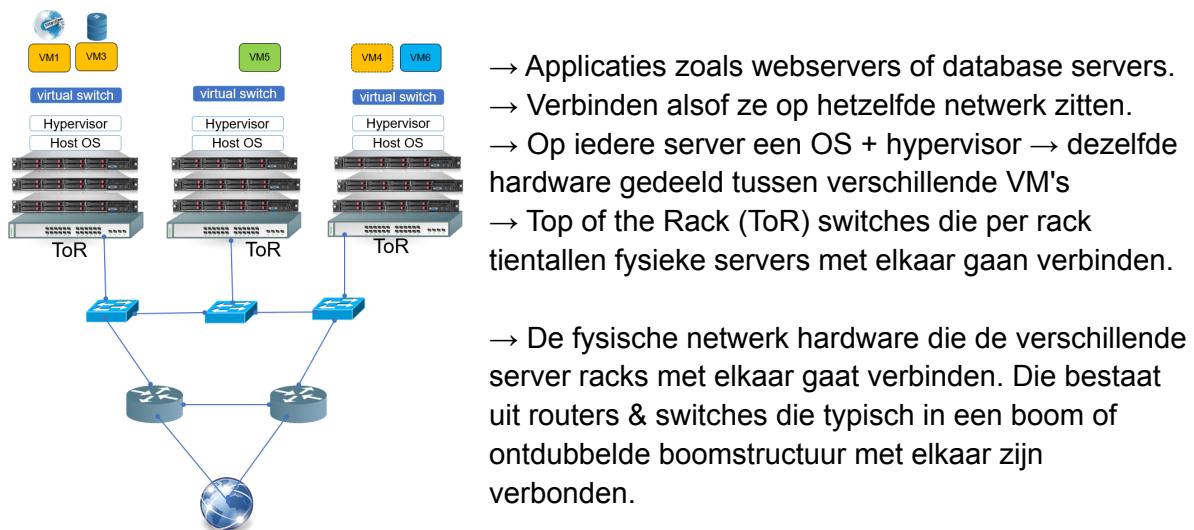
Virtualisatie is het principe waarin een bepaalde **functie** wordt **ontkoppeld** van specifieke **hardware** waarop het wordt uitgevoerd. Door die ontkoppeling kunnen we sneller en robuuster omgaan met bepaalde omstandigheden. We kunnen makkelijk installaties **migreren** naar een andere machine. Op die manier kunnen we makkelijk installaties **produceren** en hoeven we weinig tot geen manuele configuraties om te herstellen van een bepaalde fout.

Computernetwerken bestaan uit veel meer dan alleen switches en routers. Er bestaan ook middle boxes, ofwel netwerkfunctie. Dit is een toestel die het netwerk beter of veiliger moet maken of specifieke ondersteuning bieden voor bepaalde applicaties zoals voice- of media verkeer.

De uitdaging om virtuele applicaties voor netwerken te gebruiken, bestaat uit de functie zo efficiënt mogelijk in software te implementeren en schaalbaar te maken.

Datacenter components

Deze manier van werken werd eerst toegepast in datacenters.

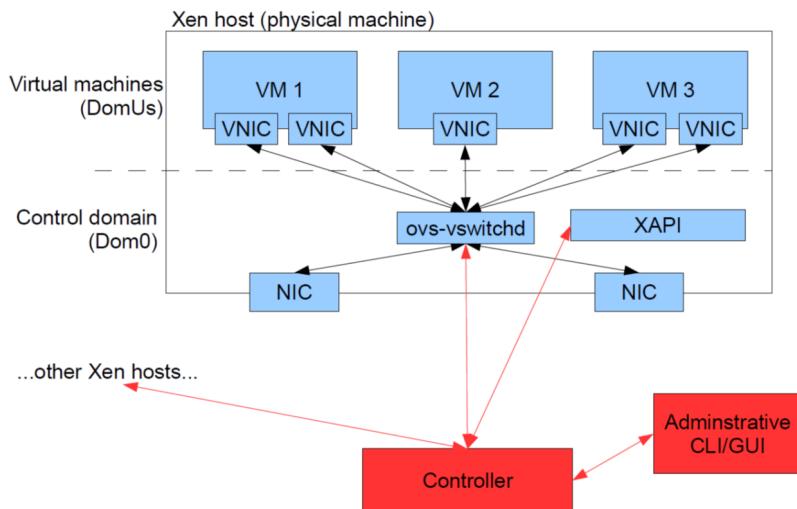


Datacenter components - Virtual switch

Een switch die louter in **software** is geïmplementeerd en die verschillende **virtuele interfaces** van verschillende **virtuele machines** met elkaar kan gaan verbinden alsof het een echte ethernet switch is.

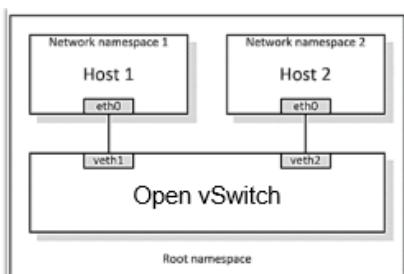
Een veelgebruikte gratis open-source en redelijk recente virtuele switch wordt **Open vSwitch** of OvS genoemd. OvS werkt op Linux systemen en bestaat uit twee grote delen. Eén deel met de pure **data plane**, die deel uitmaakt van de kernel space, om snel pakketten te kunnen ontvangen en doorsturen.

Een ander deel is geïmplementeerd als **applicatie** in de user space en dient om de switch te kunnen configureren. OvS maakt nu standaard deel uit van de Linux kernel, en was ook een van de eerste die zich liet configureren met OpenFlow.



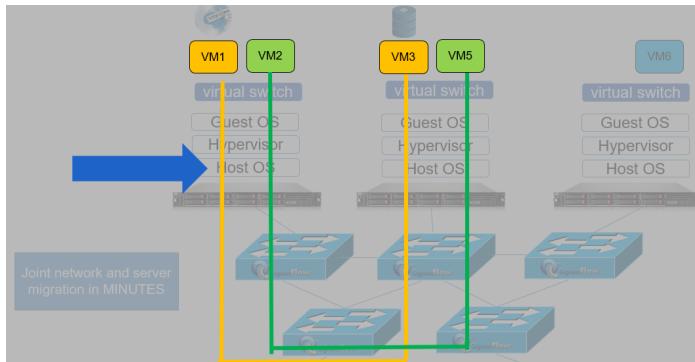
Op de figuur zien we hoe een fysieke machine gebruik maakt van Xen als **hypervisor** om 3 virtuele machines te hosten. Ieder van die VM's heeft virtuele hardware in de vorm van een gegeven CPU, RAM, disk, maar ook een aantal **virtuele netwerk interfaces** (VNIC) zoals ondersteund door de hypervisor. De VNIC's van verschillende VM's, maar ook fysieke interfaces van de host kunnen nu aangesloten worden op dezelfde OvS instantie en de switch kan geconfigureerd worden om verkeer van de ene VM door te sturen naar de andere aan de hand van de controller. De rode controller kan OpenFlow gebruiken als interface naar de switch om zo specifieke regels te configureren in de switch, zodat deze het werk doet in het verkeer tussen de verschillende VM's. Met die controller kan je dan interageren via de CLI of GUI om je netwerk te programmeren.

OVS in mininet



Mininet maakt gebruik van Open vSwitch om een of meerdere LAN netwerken in je netwerk experimenten toe te laten. Mininet maakt geen gebruik van virtuele machines, maar van **network namespaces**. Dit is een Linux functionaliteit die toelaat om een groep processen te **isoleren** van de network stack van de host of van andere network namespaces. Iedere network namespace of node, bevat virtuele interfaces en die interfaces worden via virtuele ethernet links, als Linux dat toelaat, met de switch verbonden. Doordat zowel de network namespaces als de Open vSwitch heel erg efficiënt werken, kan je vrij grote netwerken emuleren.

Virtual networking & tunneling



In datacenters en op de eigen laptop, maken we gebruik van virtuele switches om virtuele netwerken met elkaar te verbinden. Op die manier kunnen verschillende Vm's op dezelfde fysieke host met elkaar communiceren alsof ze op hetzelfde LAN netwerk zijn aangesloten.

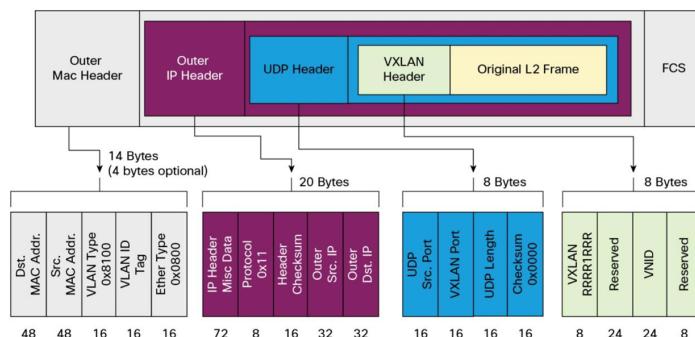
L2 Ethernet network over IP

Om toe te laten dat VM's vanop verschillende hosts met elkaar kunnen communiceren alsof ze in dezelfde LAN zitten, moeten **virtuele links** voorzien worden. Zo'n virtuele link kan op verschillende manieren geïmplementeerd worden. In een datacenter zitten er waarschijnlijk IP routers tussen verschillende server racks. Wanneer dit het geval is kan je geen VLAN's gebruiken, want dan zou je al snel met twee geïsoleerde LAN subnetten zitten die **dezelfde prefix** wensen te gebruiken.

Om zo'n virtuele laag 2 link toe te laten over een IP netwerk heen, wordt vaak **VXLAN tunneling** gebruikt. Het voorziet een tunnel van de ene host node van het datacenter naar een andere host node in hetzelfde of een ander datacenter. De eindpunten van de tunnel worden **VTEP's of tunnel endpoints** genoemd. Zo kan je VM's laten communiceren die verbonden zijn via een extern IP netwerk, alsof tot tot dezelfde LAN behoren.

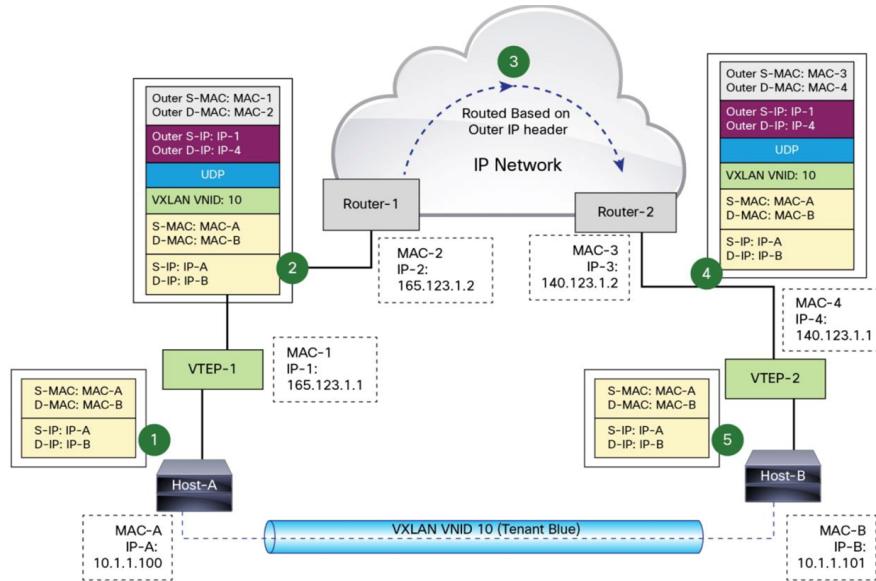
Het **voordeel** van VXLAN is dat je veel (2^{24}) LAN netwerken kunt definiëren tussen VXLAN segmenten (veel meer dan de gebruikelijk 4096 VLAN's).

VXLAN packet header



→ VXLAN tunneling encapsuleert een origineel **laag 2 frame** met de gegeven VXLAN header in een **UDP pakket** dat geëncapsuleerd wordt in een **IP pakket** dat verder getransporteerd kan worden.

VXLAN Packet Forwarding Flow voorbeeld



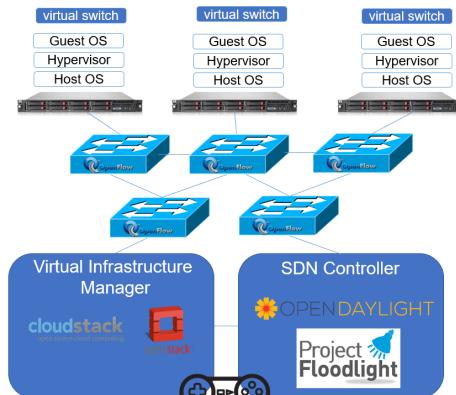
Links onder zien we Host A die we kunnen beschouwen als de ene VM die wenst te communiceren met Host B. Dit is een andere VM die we rechtsonder zien, die zich mogelijk bevindt in een ander datacenter gescheiden door een netwerk waarin zich routers bevinden.
(1) De Host A **genereert** IP pakketten met een gegeven IP adres en MAC adres naar zijn bestemming, Host B, in hetzelfde logische subnet.

(2) De tunnel endpoint **encapsuleert** de pakketten met een VXLAN tag in een UDP header en (3) **transporteert** ze aan de hand van een nieuwe IP header, gericht van het ene tunnel endpoint naar het andere tunnel endpoint. Onderweg kan de daaraan toegevoegde outer ethernet header gewijzigd worden van link tot link, tot het bij de ander tunnel endpoint terechtkomt.

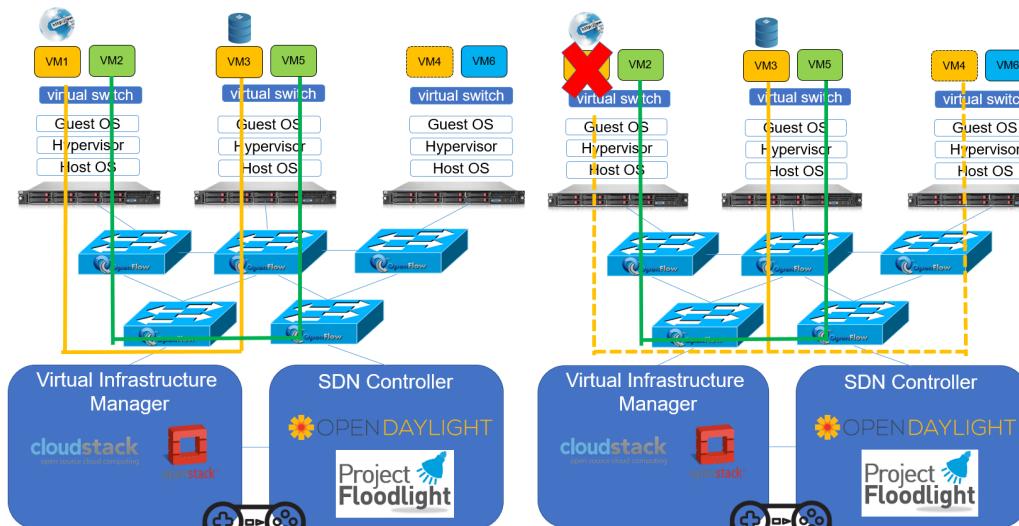
(4) Daar wordt het pakket **uitgepakt** en (5) **afgeleverd** alsof de twee hosts tot hetzelfde LAN netwerk behoren. VXLAN voorziet dus aan de hand van een tunneling mechanisme, een manier om een LAN link te virtualiseren over een IP netwerk.

Network recovery in the modern datacenter

Software heeft een steeds groter aandeel in de manier waarop netwerken worden gebruikt en aangestuurd. Een modern datacenter combineert de elementen van SDN en netwerk virtualisatie.



→ Een datacenter bestaande uit SDN switches en een aantal racks van servers. Op die servers draaien een aantal webapplicaties in de vorm van **VM's**, die met elkaar verbonden zijn in een **virtueel LAN netwerk**. Onderaan is er een **SDN netwerk controller** en anderzijds een **orchestrator** die de **hypervisors** van verschillende servers moet gaan aansturen. Zo'n orchestrator moet dan bijvoorbeeld gaan beslissen voor een nieuwe service-aanvraag welke server VM's zullen gebruikt worden.



→ De gele VM's draaien een webserver en database server. Ze zijn met elkaar verbonden zijn via de gele LAN. Analoog draait een andere service op een groene verzameling van VM's verbonden via hun eigen virtuele netwerk.

→ Als in zo'n datacenter zich een fout voordoet en een VM of onderliggende host crashed, wordt de **orchestrator** hiervan op de hoogte gebracht en gaat die en andere server selecteren om dezelfde VM daarop herop te starten.

Daarnaast gaat de **orchestrator** de **SDN controller** aansturen zodat ook het **netwerk** tussen de gele VM's aangepast wordt om de nieuwe VM erin op te nemen en de oude uit het netwerk te verwijderen. Dit alles gebeurt in enkele seconden, omdat het netwerk geautomatiseerd is en geen manuele configuratie meer vergt van de netwerkbeheerder.

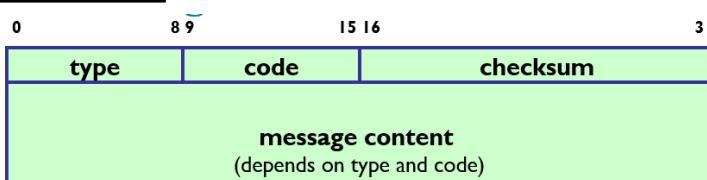
Sectie 5.6 Internet Control Message Protocol (ICMP) (control level)

Het is een stukje functionaliteit in 'the control plane' dat **diagnostische mechanismen** voorziet om:

- (1: **Ping**) Te zien of een gegeven host **bereikbaar** is en de **RTT** (round-trip-time) te meten vanaf een andere host.
- (2: **ICMP redirect**) Te detecteren of bepaalde **routes** in het netwerk **korter** kunnen aan de hand van **ICMP redirect**.
- (3: **Traceroute**) **Tussenliggende hops** detecteren van een route tussen twee hosts.

Voor elk van deze functionaliteiten maakt ICMP gebruik van bijzondere pakketten **bovenop de IP header** (Network layer (3)). Het maakt dus geen gebruik van een Transport Layer (4) protocol (TCP/UDP).

ICMP bericht



De checksum wordt enkel op de ICMP velden en inhoud berekend, met het checksum veld zelf op 0. Daarna kan er een body worden toegevoegd die afhankelijk is van de code en type van de eerdere velden.

Er zijn grofweg twee types in ICMP berichten: enerzijds **errors** die gesignaliseerd kunnen worden (ICMP redirect, Traceroute) en anderzijds **query response** ICMP berichten (ping).

Voorbeeld ICMP query: Ping

De bron stuurt een **ICMP echo Request** naar de bestemming, die antwoordt met een **ICMP echo Reply**.

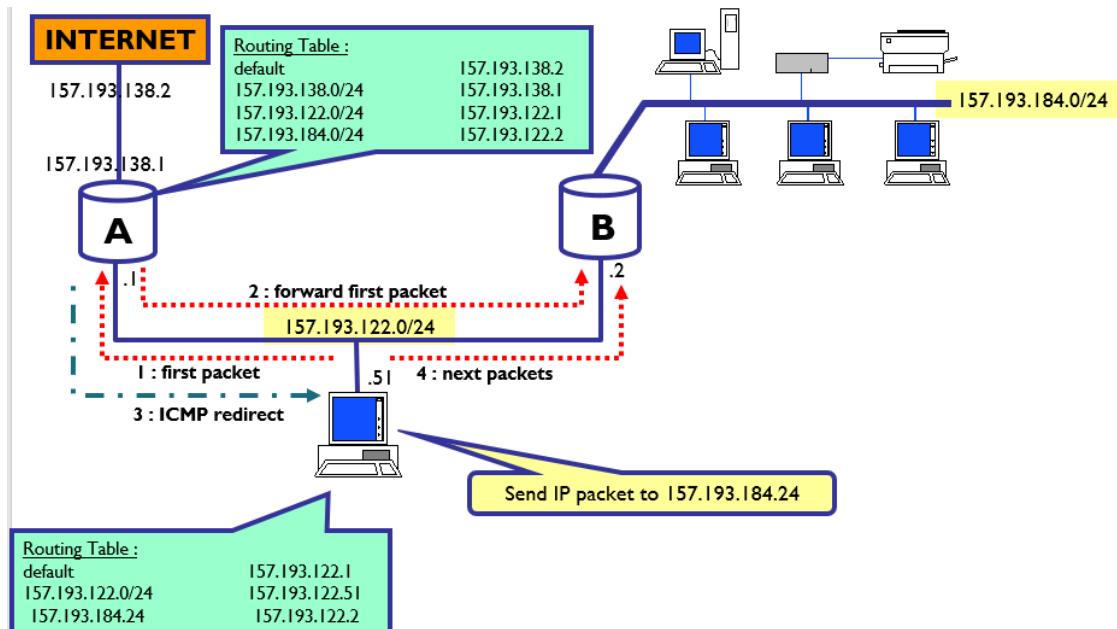
Een **ICMP server** wordt meestal geïmplementeerd als deel van de **kernel** van het besturingssysteem en niet in een traditionele user-space daemon zoals een webserver. De ICMP module van de kernel luistert dan naar ICMP berichten in de IP header.

Bij het ontvangen van de echo response kan de bron de RTT berekenen. ICMP wordt op sommige servers geblokkeerd om attacks te vermijden waar massaal ICMP berichten worden gestuurd naar een gegeven doelwit.

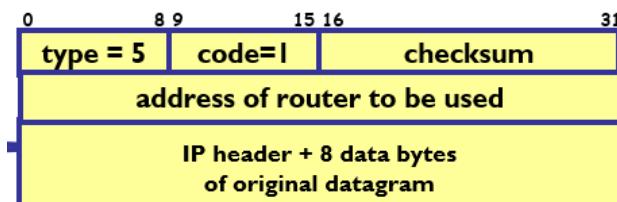
Wanneer **ping** wordt gebruikt langs een **router die NAT gebruikt**, zal ipv de poortnummers de **ICMP Query Identifier** gebruikt worden.

Voorbeeld ICMP: Redirect

Een ander belangrijke functie van ICMP is het detecteren van specifieke fouten in de forwarding van het netwerk.



→ De router stuurt het bericht terug op het netwerk vanwaar het kwam, dus stuurt die een ICMP redirect naar de host. Hierin wordt de correcte gateway B meegegeven. Aan de hand van dit ICMP redirect pakket gaat de host zijn **forwarding tabel uitbreiden** voor het subnet achter B.

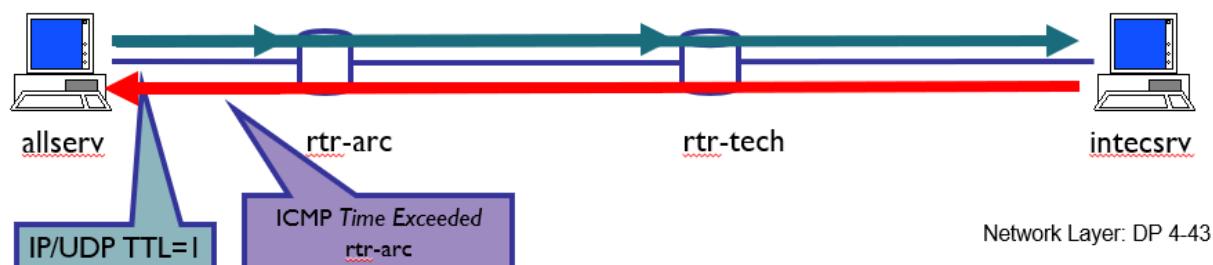


→ In de payload staat zowel het adres van de correcte gateway als de originele IP header en de eerste 8 bytes van het pakket meegestuurd.

ICMP error: Time exceeded (Traceroute)

Wanneer de TTL op 0 komt te staan wordt er een ICMP naar de afzender gestuurd om te vermelden dat het pakket zijn bestemming nooit heeft bereikt. **Traceroute** maakt hier gebruik van om de onderliggende hops naar een host te onderzoeken. Het stuurt ping berichten met een oplopende TTL tot het aantal hops dat nodig is om tot bij de bestemming te geraken.

Niet alle routers zijn geconfigureerd om een Time Exceeded te sturen, dit toont als een sterretje bij de output in de CLI.



Network Layer: DP 4-43

Sectie 5.7 Netwerkbeheer en SNMP (control level)

Een **netwerk** dat wordt beheerd door **dezelfde partij** wordt ook een **autonomus systeem** of **ASM** genoemd.

Netwerkbeheer omvat het gebruik, integratie en coördinatie van de hardware, software en de menselijke gebruiker voor het monitoren, testen, opvragen (pollen), configureren, analyseren, evalueren, en controleren van het netwerk en de onderdelen waaruit het bestaat om tegen redelijke kosten de doelstellingen op het gebied van realtime operationele performance en QoS te bereiken.

Die structuur wordt vaak door het zogenoemde FCAPS model beschreven.

FCAPS (acroniem)

Fault management houdt zich bezig met het **opsporen van fouten** en bijhorende **troubleshooting** en mechanismen om die **problemen op te lossen**. Typische netwerkproblemen zijn dat servers niet beschikbaar zijn, nodes geen netwerktoegang meer hebben, ...

Configuration management houdt zich bezig met het systematisch **configureren** en **update**n van zowel **hardware** als **software** toestellen. Een uitdaging voor grote instellingen is het bijhouden van een **inventaris** van alle toestellen.

Accounting management moet verschillende **gebruikersprofielen** gaan definiëren en per profiel bepalen welke toegang gebruikers mogen hebben, hoeveel quota, ...

Performance management houdt zich bezig met het **bijhouden, meten, bijsturen** van de performance van je netwerk. Dit kan gemeten worden in termen van bandbreedte, delay, uptime, ... Voor service providers worden die zaken meestal in contracten/**SLAs (Service Level Agreements)** vastgelegd. Performance management moet er dan voor zorgen dat die contractuele vereisten kunnen nagekomen worden (QoS). Om die performance te **meten** maakt men vaak gebruik van **telemetrie** diensten. Dit zijn netwerkonderdelen die metingen kunnen doen op de packageflow die ze zien passeren op bepaalde punten of monitors die van tijd tot tijd meten hoe snel een server reageert op requests. Deze meettoestellen noemt men probes. **Passieve probes** luisteren enkel naar bestaand netwerkverkeer terwijl **actieve probes** zelf verkeer introduceren in je netwerk om metingen te doen.

Security management gaat bepalen hoe je je netwerk gaat **beveiligen** tegen attacks of ongewenst gebruik. Hiervoor ga je bijvoorbeeld firewalls configureren die bepaald verkeer gaan blokkeren.

Network management architecture

Wanneer we spreken over network management maken we meestal onderscheid tussen twee zaken.

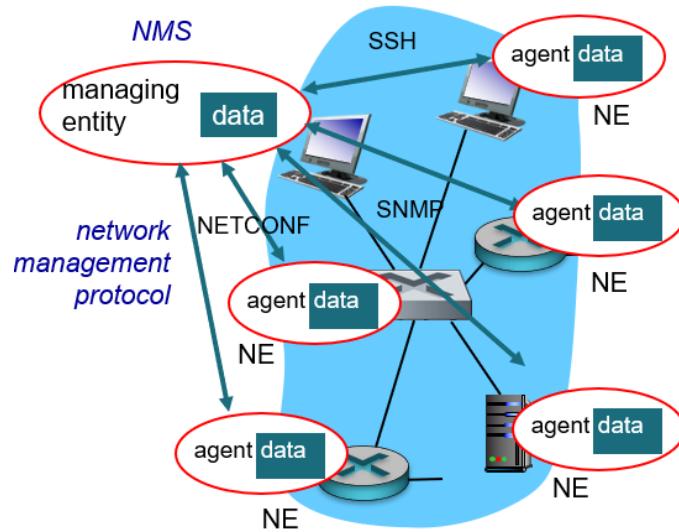
Network element

Enerzijds hebben we de **verschillende toestellen die we wensen te beheren** (routers, switches, hosts, servers, ...). Die noemen we de **network elements**. Elk van die network elements is verantwoordelijk voor een aantal zaken die **gemonitord** of **geconfigureerd** moeten worden. Dit zijn de **network management objects**. Dit kan bv. een route zijn op een router of een teller op een server die bepaald hoeveel connecties toegelaten worden. Om die toegang te geven tot network management objects kan een netwerk element gebruik maken van een stuk software, de **management agent**, dat lokaal op het toestel draait. Die agent moet er dan voor zorgen dat beheerders met het toestel kunnen praten en objects opvragen en aanpassen.

Network Management Station (NMS)

Dit is een toestel om te **interageren met de verschillende agents** van de beheerde toestellen. Zo'n NMS bevindt zich meestal in een controlekamer of een controlecentrum van het bedrijf waar de beheerders zich bevinden. Zo'n centrum wordt ook het NOC of het **Network Operation Centrum** genoemd. Op het NMS draaien verschillende **controle toepassingen**, ofwel **Network Management Applications** (NMA).

Zo'n NMS interageert met agents en maakt daarvoor gebruik van bepaalde management protocollen. Vroeger werd TelNet gebruikt, nadien SSH, tegenwoordig SNMP en NETCONF.



Korte geschiedenis van netwerk management (niet zo belangrijk)

Eerst het ARPA net met zeer minimalistische handmatige management. In de jaren 80-2000 groeide het internet en de onderliggende netwerken gigantisch. Het werd duidelijk dat er een meer systematische en geautomatiseerde manier nodig was voor netwerk management. Tot op de dag van vandaag maakt men echter nog steeds gebruik van SSH, waarbij men via een cli aan management doet. In het beste geval probeert men die interactie te scripten. In de jaren 90 probeerde men dit te harmoniseren in een protocol, het Simple Network Management Protocol (SNMP). Het werd ontworpen als een lightweight connectionless protocol waarbij management software zelf nog veel functionaliteit moest voorzien om er zeker van te zijn dat de configuratie gebeurde zoals gewenst. Ook is er een wildgroei aan manieren waarop fabrikanten SNMP implementeren, gevolg is dat SNMP in de praktijk vaak enkel gebruikt wordt voor monitoring en niet voor configuratie.

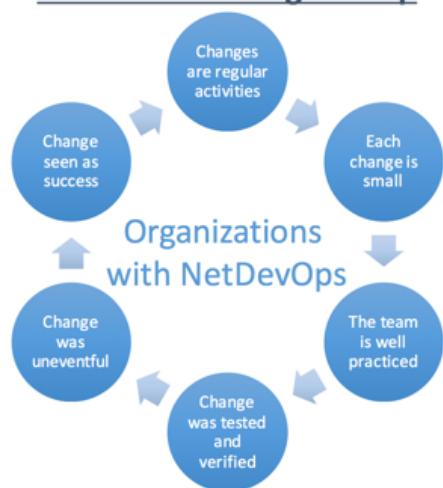
In het laatste decennium hebben ze door dat automatisering en betrouwbare workflows cruciaal zijn in het beheer van netwerken. Nieuwe management workflows zijn dan ook sterk geïnspireerd op de praktijken van cloud computing, software engineering en protocollen zoals NETCONF en tools zoals Ansible.

Een van de belangrijkste zaken om in die mindset te komen is de mentaliteit veranderen van de huidige mensen van network management.

"Culture of Fear" Reinforcing Loop



"Culture of Change" Loop



Het netwerk wordt in code gezet en dan met source control.

Simple Network Management Protocol (SNMP)

Het werd ontworpen als een protocol tussen **Network Management Stations** en de beheerde toestellen of **Network Elements**. Hierbij zijn er 4 cruciale onderdelen te onderscheiden.

1. Management Information Base (MIB)

Dit is de belangrijkste structuur om te managen. Het is een datastructuur verspreid over de verschillende netwerkelementen die alle mogelijke network management data moet gaan bijhouden. bv. configuratie parameters van een router, monitor metrieken op een webserver. Alle gemanagede data op een device wordt in een MIB ondergebracht. Een MIB bestaat zelf uit een **module** waartoe verschillende **objecten** behoren. Zowel de module als de individuele objecten worden gedefinieerd aan de hand van de **SMI taal**. De definities en standaarden worden in **RFC's** gezet.

MIB naming

Ieder object in de tabel heeft een **identifier**, zo'n identifier is een **adres in een boomstructuur** die SNMP voor MIBs definieert.

Het ISO (een andere standaardisatie organisatie) wou de volledige structuur uniformiseren waarin alle mogelijke managementinformatie een plekje kreeg. Aan de top van die structuur zijn er verschillende standaardisatie organisaties. Onder de ISO tak is een tak voor ISO specifieke standaard objecten. Daarnaast heb je een ISO number body, dit kunnen standaarden zijn die gekoppeld zijn aan verschillende landen of zelfs door verschillende bedrijven. Wij zijn vooral geïnteresseerd in het internet onder **US DOD** (hieruit komt het Arpanet). Iedere node in de boomstructuur heeft een **naam** maar ook een **numerike voorstelling**. Een node kan beschreven worden als het **pad** van **root tot leaf** met **punt** als separator. In de praktijk wordt vaak een hybride aanpak gebruikt tussen namen en cijfers.

2. Structure of Management Information (SMI): data definition language

De elementen binnen een MIP volgen een bepaalde structuur en taal waarin ze gedefinieerd zijn. Voor ieder management object in de boomstructuur moet vastgelegd worden over **welke soort data** het precies gaat. Dit gebeurt aan de hand van **SMI als data definition language**. In die taal kan een object een **basistype** hebben (string, integer, ...) en daarnaast wordt ook een **status** of semantiek vastgelegd per object. Verschillende objecten kunnen **gegroeped** worden in **MIP modules**. Dat kan een lijst van dergelijke objecten vastleggen en aangeven voor welke doeleinden die kunnen gebruikt worden.

OBJECT-TYPE: ipInDelivers

```
ipInDelivers OBJECT TYPE
  SYNTAX      Counter32
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The total number of input
     datagrams successfully
     delivered to IP user-
     protocols (including ICMP)"
  ::= { ip 9}
```

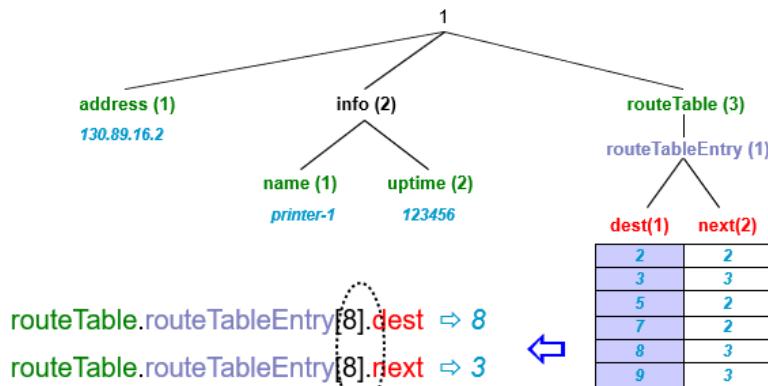
MODULE-IDENTITY: ipMIB

```
ipMIB MODULE-IDENTITY
  LAST-UPDATED "941101000Z"
  ORGANIZATION "IETF SNPv2
                Working Group"
  CONTACT-INFO
    " Keith McCloghrie
      ...."
  DESCRIPTION
    "The MIB module for managing IP
     and ICMP implementations, but
     excluding their management of
     IP routes."
  REVISION "019331000Z"
  ....
  ::= {mib-2 48}
```


SMI definitie van tabellen

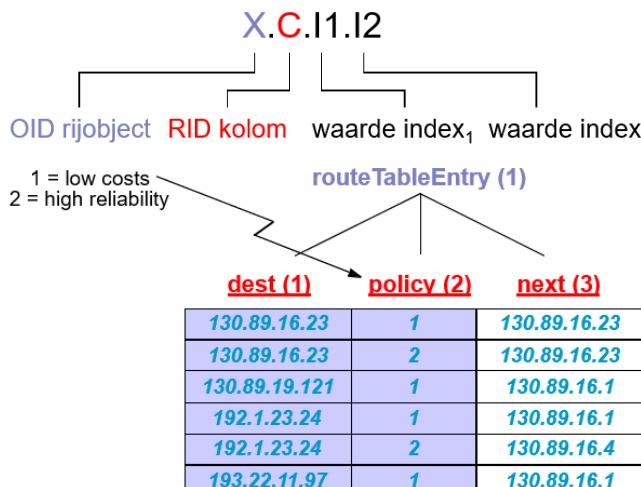
De enige vorm van structureren die SMI aanbiedt, is het creëren van tweedimensionale tabellen van cellen met **rijen** voor elke **instantie** van een of ander fysieke component, en **kolommen** voor elk van de **eigenschappen** ervan.

Bijvoorbeeld een routingtabel: het zal evenveel rijen bevatten als er entries zijn in routingtabel, kolommen voor onder meer het doeladres, de eerste hop, de te gebruiken interface, de metriek, de ttl, ...



Men indexeert een rij op zijn hashes, via de waarde van een of andere kolom als een sleutelattribuut. Het **INDEX attribuut** van het rij object, geeft aan **welke kolom** de **sleutelkolom** is.

Voorbeeld: Indices kunnen bestaan uit **meerdere velden**



3. SNMP protocol

Om die netwerkobjecten van network elements te kunnen opvragen en configureren, maken we gebruik van een **connectionless protocol** dat gebruik maakt van **UDP** en daarover een aantal berichten definieert tussen de Network Management Stations (NMS) en de ME (Managing Entity). Men definieert twee mechanismen van interactie.

Request/response

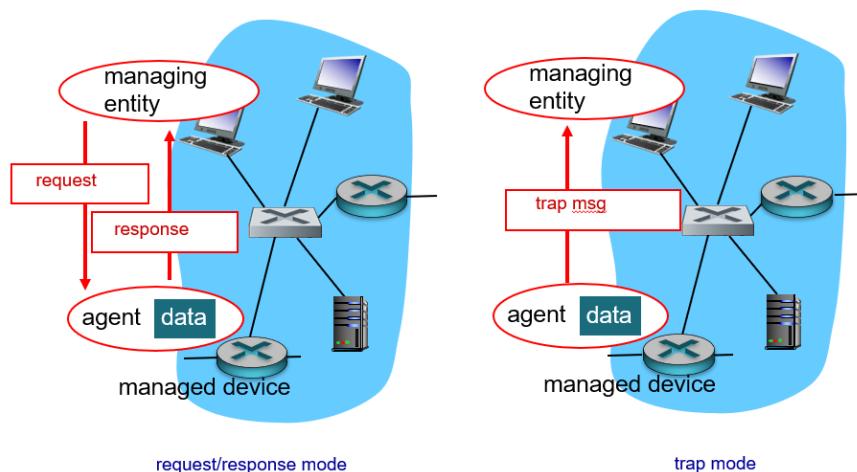
Het basismechanisme is dat de **NMS een vraag stelt** en dat die beantwoord wordt door de ME (Managing Entity). Zo kan de vraag gesteld worden om data van een object op te halen of om aan te passen. Nadien kan de ME de data of bevestiging terugsturen naar de NMS.

Trap mechanisme

Het laat toe berichten vanuit de ME automatisch te **pushen naar het NMS** via een **trap message**.

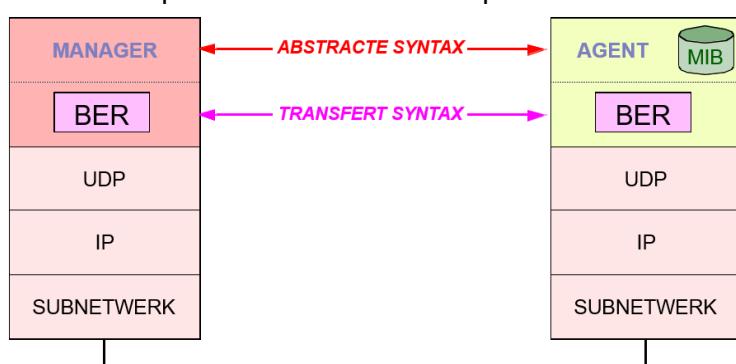
Dit kan nuttig zijn wanneer bijvoorbeeld lokaal er een alarm afgaat hiervan de NMS te informeren zonder dat de RMS continu requests moet uitsturen.

Afbeelding



Basic Encoding Rules (BER)

Niet alleen voor identificatie en definitie van objecten, maar ook voor het formaat van de berichtenuitwisseling. BERs vertalen de **abstracte** syntax in een **transfert** syntax:
tekstuele representatie → **binaire** representatie



Overzicht ondersteunde berichten SMNP

→ De eerste drie types message berichten laten managers toe om data op te vragen van een network element. 'GetRequest' voor een enkelvoudig object, 'GetNextRequest' voorziet een mechanisme om ook meteen de volgende te vragen. Via 'GetBulkRequest' kunnen hele tabellen met info worden opgevraagd.

→ Ze kunnen onderling data uitwisselen via 'InformRequests'.

→ Configuratie aan de hand van 'SetRequest'. Zoals eerder vermeld is er vanuit de fabrikant een wildgroei aan MIB definities zodat er voor ieder toestel andere MIBs moeten aangesproken worden voor configuratie.

→ Een 'Response' bericht bevat de nodige data of bevestiging van herconfiguratie die een netwerkelement terugstuurt naar de manager.

→ Aan de hand van een 'Trap' kan alarm informatie verstuurd worden de manager.

<u>Message type</u>	<u>Function</u>
GetRequest GetNextRequest GetBulkRequest	manager-to-agent: "get me data" (data instance, next data in list, block of data)
InformRequest	manager-to-manager: here's MIB value
SetRequest	manager-to-agent: set MIB value
Response	Agent-to-manager: value, response to Request
Trap	Agent-to-manager: inform manager of exceptional event

4. SNMP security and administration

Vanaf versie 3 werden security features toegevoegd. Het zorgt dat communicatie tussen de manager en een element **geëncrypteerd** kan gebeuren, dat **authenticatie** mogelijk is en **access control lists** kunnen bepalen wie toegang heeft tot welke data.

Net-SNMP

Open-source SNMP software. Het bevat 4 onderdelen, enerzijds een aantal **command line tools** om een aantal SNMP requests uit te sturen vanuit een network element. Daarnaast bevat het ook een **GUI** om de MIB te browsen van een gegeven element. Het bevat ook een **daemon** die kan luisteren naar trap signalen van bepaalde network elements. Ten slotte kan je een **agent** maken die je kan uitvoeren op een systeem zodat het zich kan voordoen als een network element met enkele MIPS die het aanbiedt naar het management systeem. Bovendien kan je het zelf uitbreiden met eigen mechanismen om de data voor bepaalde objecten aan te leveren.

```
# snmpget -d 192.168.0.1 hrMemorySize.0
Sending 42 bytes to 192.168.16.100:161
0000: 30 28 02 01 00 04 06 70 75 62 6C 69 63 A0 1B 02 0(.....public...
0016: 02 40 AD 02 01 00 02 01 00 30 0F 30 0D 06 09 2B .@.....0.0...+
0032: 06 01 02 01 19 02 02 00 05 00 ......

Received 45 bytes from 192.168.16.100:161
0000: 30 2B 02 01 00 04 06 70 75 62 6C 69 63 A2 1E 02 0+....public...
0016: 02 40 AD 02 01 00 02 01 00 30 12 30 10 06 09 2B .@.....0.0...+
0032: 06 01 02 01 19 02 02 00 02 03 03 FD B8 ......

host.hrStorage.hrMemorySize.0 = 261560 KBytes
```

→ Request voor grootte RAM geheugen

Configuratie gebeurt in '/etc/snmp/snmpd.conf'. Sommige objecten worden **automatisch** ondersteund en vanuit het besturingssysteem doorgesluisd naar het bijhorende management object. Andere object data kan **statisch** vastgelegd worden. Tot slot kunnen

bepaalde management objecten laten invullen door **dynamische scripts** die je lokaal uitvoert.

Netwerk automatisering met Ansible

Het is cruciaal voor een netwerkbeheerder om enerzijds je **configuratie** vast te leggen **als code** in bestanden, maar anderzijds ook om over automatisering mechanismen te beschikken die deze code of **configuraties automatisch** kunnen **testen of uitrollen** over verschillende netwerktoestellen.

Ansible is een van de frameworks die toelaat ieder individueel stapje in dat proces te automatiseren. Ansible kan zowel gebruikt worden om tests uit te voeren alsook geautomatiseerd een configuratie uit te rollen.

Het is bruikbaar voor vele zaken.

Orchestratie mechanismen om automatisch ervoor te zorgen dat VM's worden getriggerd als er een hoge load komt op het netwerk. **Configuration management** om ervoor te zorgen dat een gegeven toestel of reeks toestellen automatisch een gegeven configuratie opgelegd krijgen. De uitrol van bepaalde applicaties en bijbehorende resources (**Application deployment**). Het uitvoeren van checks en automatische uitrol van succesvolle componenten (**provisioning & continuous delivery**). Alsook voor de uitvoer van enkele automatische security procedures (**security and compliance**). Elk van die procedures kan zowel op servers als op netwerk toestellen uitgevoerd worden.

Ansible execution modes

Ansible is opgebouwd rond **plugins** die zich kunnen toespitsen op de interactie met eender welk systeem, gaande van SSH tot SNMP. Ansible zelf is een Python applicatie die bepaalde playbooks kan uitvoeren. We maken hier onderscheid tussen twee modes waarin Ansible kan werken.

De eerste is de **local execution mode**, waarin de playbooks worden uitgevoerd vanop een controller of network management node en die remote nodes aansturen gebruik maken van specifieke protocollen zoals SNMP. Voor netwerkmanagement wordt hier vooral op teruggevallen.

Anderzijds is er de **remote execution mode** waarin alle Ansible code en logica wordt gekopieerd naar de specifieke nodes om daar uitgevoerd te worden. Dit gebeurt meer op servers dan op netwerktoestellen.

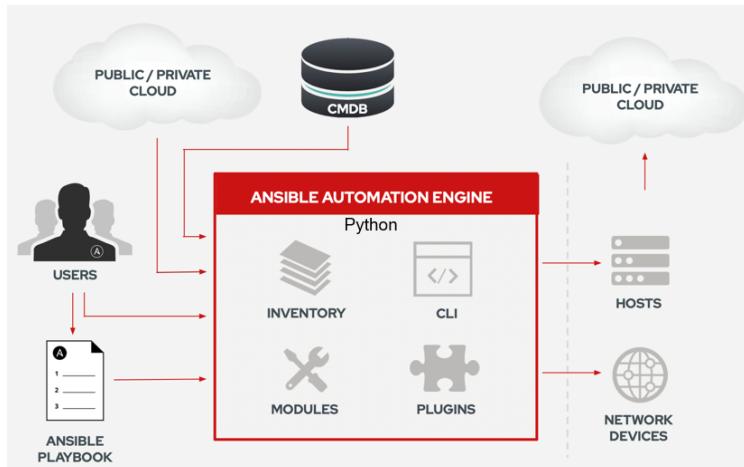
Typische use cases van Ansible

Een eerste scenario kan bestaan uit het gebruik van Ansible voor **backups and restore**. In dit geval kan je bijvoorbeeld een aantal Ansible scripts/playbooks definiëren om van gegeven servers een backup te nemen. Alsook playbooks die voor een gegeven timestamp die backup terugzetten of een rollback moet dit falen.

Een tweede scenario voorziet Ansible voor **configuration management**. Zo kan een lijst van configuratiebestanden gechecked worden aan de hand van een playbook, tegen een

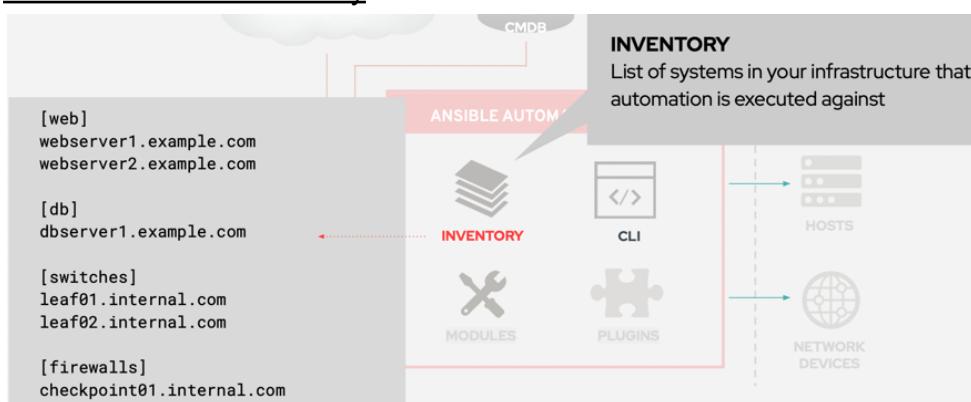
aantal condities of tegen een model configuratie. Wanneer die afwijken van een aantal vereisten kan automatisch een playbook uitgevoerd worden die default settings terug plaatst. Het laatste scenario kan zich focussen op het discoveren van aanwezige systemen een daarvoor automatisch documentatie genereren die bijvoorbeeld versies en de status van de systemen samenvat in een aantal rapporten.

Ansible architecture



Bovenaan zien we een **configuratie database** die alle configuratie op een version controlled manier bijhoudt. Rechts zien we de devices die we wensen te beheren of aan te sturen. Links zien we de netwerkmanager of user, die aan de hand van playbooks de devices aanstuurt. Het belangrijkste staat centraal, namelijk de **Ansible Automation Engine**, die verwerkt playbooks en interageert op basis daarvan met de devices. De Ansible Automation Engine bevat een inventory/lijst van devices die hij moet beheren. Daarnaast zijn er plugins waarmee de basisfunctionaliteit van Ansible kan uitgebreid worden en modules die toelaten met verschillende remote systemen te interageren.

Voorbeeld Ansible inventory



→ Een lijst van IP adressen of hostnames die mogelijk verdeeld zijn in verschillende groepen. Bijvoorbeeld een groep webservers, een groep databases, groep switches, ...

Ansible scripts

Om de Ansible engine zijn werk te laten doen, moeten we die voorzien van scripts of playbooks die de engine vertellen wat hij moet doen. Ansible playbooks zijn YAML bestanden, ze zijn relatief leesbaar. Zo'n playbook bevat een lijst van "**Play's**" die moeten uitgevoerd worden. Iedere play specificeert een **aantal taken** en een **lijst van hosts** waarop

die moeten uitgevoerd worden. Een taak is een **call naar een Ansible module**, dit valt te zien als een functie die kan uitgevoerd worden.

Ansible modules

Ansible voorziet een heleboel modules die het eenvoudiger maakt met een bepaald systeem te interageren. Vaak zijn ze geïmplementeerd in Python en retourneren ze JSON data. *Voor netwerk automatisering zijn er bijvoorbeeld modules die toelaten om zaken te configureren op CISCO, Juniper of Arista systemen.* Die kan je dan rechtstreeks vanuit het playbook aanroepen.

Voorbeeld Ansible tasks

```
- name: configure interface settings
  ios_config:
    lines:
      - description test interface
      - ip address 172.31.1.1 255.255.255.0
    parents: interface Ethernet1
```

Eerst en vooral een naam, dan definieer je de gebruikte module. Daarna geef je de argumenten waarmee je de module wenst aan te roepen.

Key Ansible features

Er is **geen nood aan een agent** die draait op de devices die gemanaged moeten worden. Ansible kan **zelf uitgebreid worden** aan de hand van **modules** die met verschillende types devices kunnen samenwerken.

Daarnaast maakt Ansible het makkelijk om **eenzelfde configuratie** of taak te parametriseren voor toestellen van hetzelfde type, dit is vaak iets wat heel handig is. Bv. een datacenter dat duizenden switches van hetzelfde type heeft. Ansible laat toe om een **template** te maken van de configuratie van zo'n switch en vervolgens de logica te voorzien om voor iedere individuele switch de parameters daarvoor in te vullen. Die template taal is de populaire J2 (Janja2) taal.

Tot slot zijn de belangrijkste sterkepunten van Ansible de manieren waarop **Ansible modules** ontworpen zijn.

Modules zijn zo opgesteld dat ze de beheerder dwingen om te definiëren wat de gewenste eindtoestand is (**declarative**), eerder dan de methode om bij die eindtoestand te geraken. Zo'n declaratieve aanpak laat toe om alle logica in de module zelf te implementeren en af te schermen voor de gebruiker.

Idempotentie is daarboven de belangrijke eigenschap waaraan modules moeten voldoen. Dit houdt in dat als je een gegeven taak die gebruik maakt van een module enig aantal keer na elkaar uitvoert, het resultaat altijd hetzelfde zal zijn. Modules moeten dus de logica voorzien zodat enkel die zaken die nog gedaan moeten worden, worden uitgevoerd. De eindtoestand zal op die manier steeds gecheckt worden tegen het doel dat gedefinieerd werd in het playbook.

Die combinatie van ansible features maakt dat het een robuust en betrouwbaar automatiseringsplatform is en ondersteuning biedt voor heel veel hardware en software platformen.

H6 Linklaag

Sectie 6.4 Local-Area netwerken met switches

ARP (Address Resolution Protocol)

Het is een protocol dat de lijm vormt tussen de netwerklaag en de datalink laag.

Interfaces krijgen een achterliggend 48-bit MAC adres, ook wel fysisch adres, LAN adres, ethernet adres genoemd. Dit zijn adressen die enkel gebruikt worden tussen hosts die rechtstreeks verbonden zijn aan de hand van deze 'link' of 'link technologie' (binnen hetzelfde subnet dus). MAC adressen volgen **geen bepaalde structuur** en worden toegekend door de fabrikant van het netwerk interface. Het is onveranderbaar.

Wanneer we een pakket via een router naar een ander netwerk willen sturen staat er in dat pakket enkel het IP adres van de zender en ontvanger. In de netwerklaag beschikken we niet over een mechanisme om de router te vertellen dat het pakket voor hem bestemd is in de volgende stap (gateway naar dat andere netwerk).

Een pakket zal het **bron MAC-adres (S-MAC)** en **bestemmings MAC-adres (D-MAC)** bevatten. Eenmaal de router het pakket heeft ontvangen van A, zal de router het frame decapsuleren en afleveren aan de netwerklaag. Die gaat een lookup doen in zijn forwarding tabel en het pakket doorsturen via de link die toegang biedt tot B. De **IP header** van het pakket **blijft onveranderd**, maar opnieuw wordt hiervoor een **frame header** toegevoegd met een ander MAC adres voor bron en bestemming.

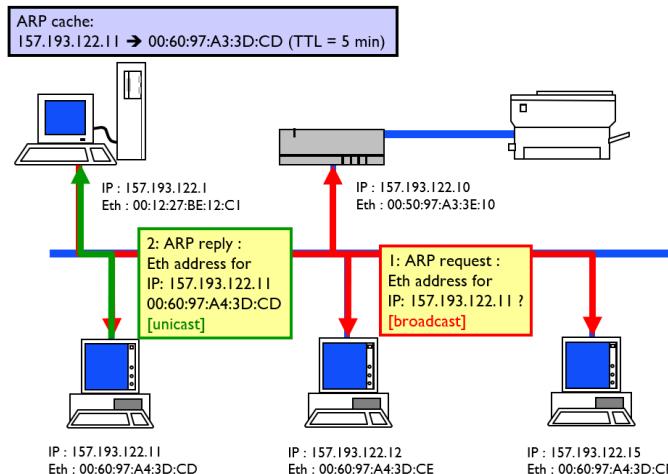
IP adressen worden globaal gebruikt en over verschillende LAN's of subnetten heen, terwijl MAC adressen enkel gebruikt worden binnen gegeven subnetten.

Hoe bekomen nodes de MAC adressen van gateways of andere nodes op het LAN netwerk?

Via ARP kunnen we voor een gegeven IP adres in een gegeven subnet MAC-adressen opvragen.

ARP werkt *plug-and-play* en vereist dus **geen bijkomend beheer**. Iedere node maakt gebruik van een **lokale tabel/cache** die bijhoudt welk MAC adres hoort bij een gegeven IP adres op het gegeven subnet. Wanneer een MAC adres niet gekend is, kan je het als node vragen op het gegeven subnet aan de hand van een **broadcast bericht**. Iedere entry in de tabel heeft een **TTL**, bijgevolg moeten de entries verlengd worden aan de hand van **refresh messages** → **soft-state protocol**.

ARP voorbeeld



ARP cache management

Door verkeerde configuratie of spoofen kan het zijn dat er **foute entries** zijn met hetzelfde IP adres.

Met het 'arp' commando kunnen we een overzicht krijgen per interface van alle entries. Per interface, want normaal gezien hoort elke interface tot een eigen subnet.

Het kan handig zijn om de hele ARP cache te wissen 'arp -d -a'. Dit kan helpen wanneer bepaald verkeer niet aankomt bij zijn bestemming of wanneer onzeker is of nieuwe adressen wel goed zijn geüpdateert.

Een specifiek IP adres wissen kan met 'arp -d cnet.ugent.be'. Dit kan gedaan worden wanneer je weet dat een bepaald IP adres dubbel gebruikt wordt. Je kan ook zelf een entry toevoegen die niet vervalt. Dit kan gebruikt worden een MAC adres te specificeren waarvan het IP adres gespoofed wordt.

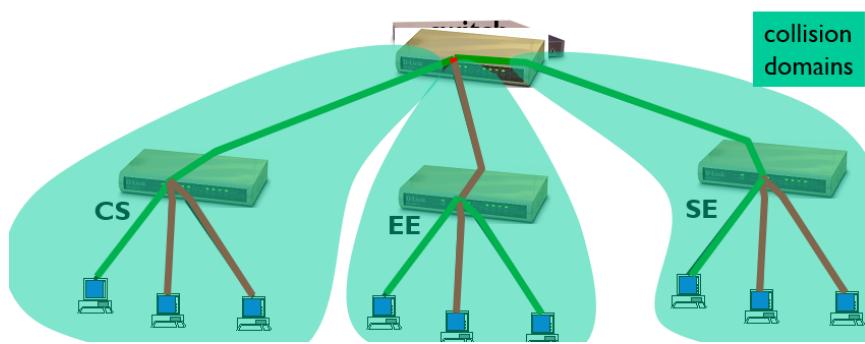
Ethernet switching

Een **hub** is een passieve repeater op de fysische laag zodat het lijkt dat alle toestellen die op de hub zijn aangesloten op 1 shared medium. IN die context kunnen we gebruik maken van protocollen zoals CSMA/CD om ervoor te zorgen dat het kanaal gebruikt kan worden door meerdere partijen.

Een shared medium heeft enkele nadelen. Er kan slechts 1 toestel op een moment gebruik maken van het medium.

Een ethernet switch is een actief store-and-forward toestel. Het **ontvangt** frames, **slaat** ze **op** in een buffer, doet een **lookup** in een MAC forwarding tabel en **stuurt** het **door**. Het werkt op laag 2 (Link Layer) en is grotendeels vrij van manuele configuratie. Een ethernet switch is een plug-and-play oplossing om meerdere toestellen in hetzelfde LAN netwerk met elkaar te verbinden. Ze werken **transparant** omdat hosts niet zien dat ze zijn verbonden door een switch.

Hub kunnen nog steeds gebruikt worden. Het breidt de collision domain of shared medium uit, terwijl de switch het shared medium opbrekent in verschillende **segmenten**. Een ethernet switch gaat selectief verkeer doorsturen naar een of meerdere poorten.

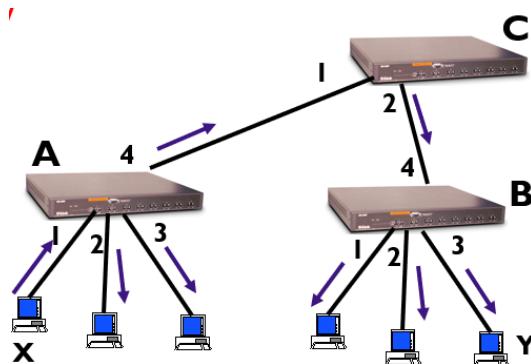


Forwarding

Forwarding in een switch heeft 3 functies.

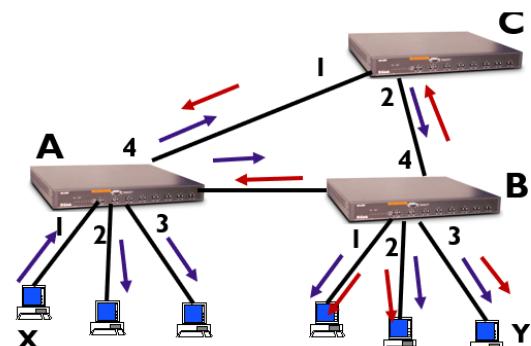
1. Flooding

De eerste is **flooding**, zonder extra informatie gaat de switch alle verkeer ontvangen via een interface en flooden naar alle andere interfaces. Dit is **geen repeating** op de fysische laag zoals een hub, een switch gaat namelijk eerst het frame inspecteren en wanneer er geen verdere informatie vorhanden is gaat hij het flooden. Dit lijkt op het gedrag van een hub, omdat alle poorten verkeer ontvangen. Het is echter verschillend, omdat er **geen shared medium** is, alle hosts kunnen informatie tegelijk verzenden.



2. Lussen voorkomen: Spanning Tree Protocol (STP)

Een tweede functie zorgt dat complexe topologieën met meerdere switches, waarin mogelijk lussen zitten, toch vlekkeloos werken. Hiervoor maken switches gebruik van een **Spanning Tree Protocol**. Zoals de naam aangeeft gaat het STP de topologie omzetten in een boomstructuur **zonder lussen**. Dit voorkomt dat frames eindeloos geflood worden in een netwerk.



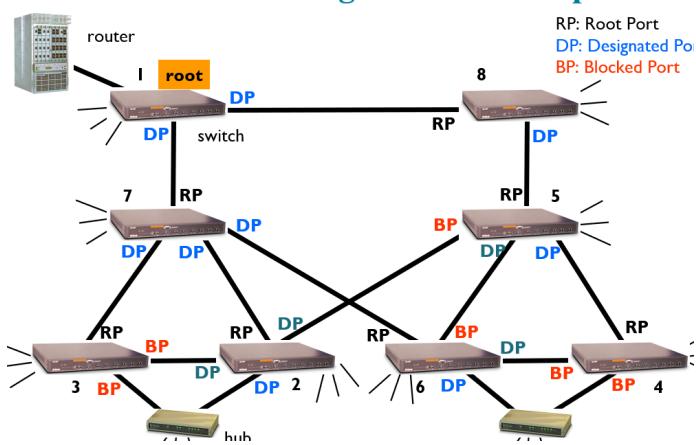
Het leert volledig automatisch welke poorten van switches geblokkeerd moeten worden. Hiervoor maakt het gebruik van controleberichten die switches met elkaar uitwisselen, Bridge Protocol Data Unit (BPDU). Het maakt gebruik van **4 stappen**.

Eerst worden **alle poorten geblokkeerd** voor dataverkeer. Vervolgens wordt een root switch geselecteerd. Dit wordt de root of de bovenste switch van de boomstructuur. In de derde stap wordt de **minimale boomstructuur** opgesteld op basis van de '**kortste pad kosten**' naar de geselecteerde root node. Tot slot gaan poorten hun toestand veranderen naar 'forwarding mode' als dit wordt toegelaten door de berekende boomstructuur.

De **root node** wordt bepaald aan de hand van een **switch identifier (switch ID)**. Het wordt opgebouwd uit twee delen, enerzijds een **prioriteitswaarde** van 2 bytes die de

netwerkbeheerder kan instellen en daarnaast een **MAC adres**. Hoe **kleiner de waarde**, hoe **groter de kans** dat de switch de root switch wordt.

De poort die tot het kortste pad leidt, wordt de **root poort**. Nadien wordt voor elk netwerk segment bepaald wat de poort is die leidt naar het kortste pad naar de root switch. De segmenten zijn de links die switches verbinden. Ieder segment vormt op zich een shared medium en kan dus ook hubs bevatten. Het segment is verbonden met een of meerdere switches en elk van die switches had in de voorgaande stap reeds bepaald wat de root poort was. In deze stap gaan switches samen bepalen **per segment** welke poort leidt tot het kortste pad naar de root, de **designated port** van het segment. Iedere poort die geen root port of designated port is, wordt geblokkeerd. Het kan voorkomen op een gegeven segment of switch meerdere poorten leiden naar een pad met dezelfde kortste route naar de root. In dat geval bepaalt de kleinste switch ID welke poort de designated poort wordt, de andere poorten worden geblokkeerd.



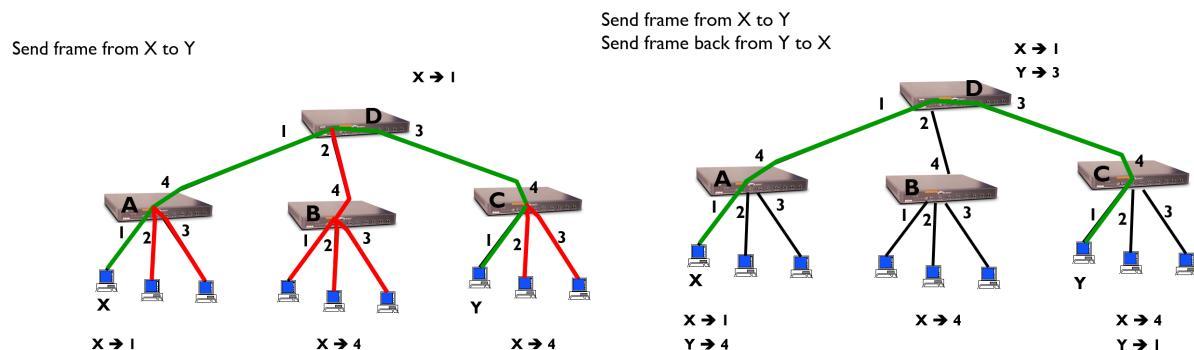
Zelfs in relatief complexe topologieën gaat het STP convergeren in 30-60 seconden. Het betekent dat alle poorten hun finale staat hebben bereikt en dat het forwarden van data frames van start kan gaan. Dit proces gebeurt bij het opzetten van een nieuw netwerk, maar ook bijvoorbeeld bij het veranderen van een bestaand netwerk door een kabelbreuk of uitbreiding met nieuwe nodes.

De convergentietijd is acceptabel voor kleinere LAN omgevingen, maar niet voor grotere ethernet netwerken. Hiervoor werd het **Rapid Spanning Tree Protocol ontworpen (RSTP)** dat dit in enkele seconden kan doen. Nog een uitbreiding laat toe om meerdere spanning trees te gebruiken voor verschillende virtuele LAN netwerken (VLAN).

3. Automatisch leren van MAC

Tot slot zal de switch **automatisch leren** waar bepaalde MAC adressen zich in het netwerk bevinden. Eenmaal die geleerd zijn, zal de switch geen flooding meer gebruiken naar die bestemmingen. Een ethernet switch maakt gebruik van een eigen forwarding tabel, waarin hij matched op het MAC adres van zijn bestemming. Iedere entry heeft een TTL, zo heeft de switch weinig ruimte nodig voor zijn lookup tabel.

De forwarding regels in een switch moeten niet ingegeven worden, maar worden geleerd door de switch zelf. Dit leerproces is gedreven door het netwerkverkeer zelf. Wanneer een frame ontvangen wordt van een gegeven bron, gaat de switch een entry toevoegen voor de **bron** (MAC-adres) van dat frame die het koppelt aan de **interface** waarop het ontvangen werd. Dit heet **MAC learning**. Zolang er geen entry geleerd werd voor een gegeven MAC adres, zal de switch het verkeer naar die bestemming flooden.



Verschil tussen routers en switches

Ze maken beiden gebruik van het store-and-forward principe: pakket ontvangen, opslaan in buffer, route bepalen en doorsturen. Routers doen dat aan de hand van het **IP bestemmingsadres** in de **network layer**. Switches doen dat aan de hand van het **MAC adres** die zich bevindt in de **Link Layer**.

De **lookup** bij routers gebeurt aan de hand van **longest-prefix matching** in forwarding tabellen van mogelijke prefixen. De hardware hiervoor nodig is relatief complex.

De lookup van switches zijn relatief eenvoudig en werken enkel op een exacte match op het MAC adres van de bestemming. Om het geheugen te beperken wordt gebruik gemaakt van een TTL.

Virtual Local Area Network (VLAN)

Voorzien een mechanisme om een LAN netwerk op te splitsen in logische netwerken gebruikmakend van **dezelfde ethernet hardware**. Ze gedragen zich alsof het **fysische andere netwerken** betreft. MAC learning en flooding gebeurt volledig gescheiden in de verschillende VLAN's.

Wanneer gebruik gemaakt wordt van standaard ethernet switches zoals vaak het geval kan dit **inefficiënties** en **risico's** met zich meebrengen. Het heeft een **single broadcast domein**. Zolang switches geen entries hebben ingevuld voor gegeven bestemmingen wordt verkeer geflood over het hele netwerk. Dit betekent dat bepaalde delen (afdeling 1) van het netwerk verkeer kunnen waarnemen van andere delen (afdeling 2). Dit is **weinig efficiënt** aangezien bandbreedte verspild wordt, maar ook **onveilig**. Een oplossing zou zijn om de netwerken van de afdelingen compleet fysiek te ontkoppelen. Dit is duur en vaak onnodig omdat de capaciteit van het netwerk vaak voldoende is.

Er zijn een aantal mechanismen om VLAN's toe te laten.

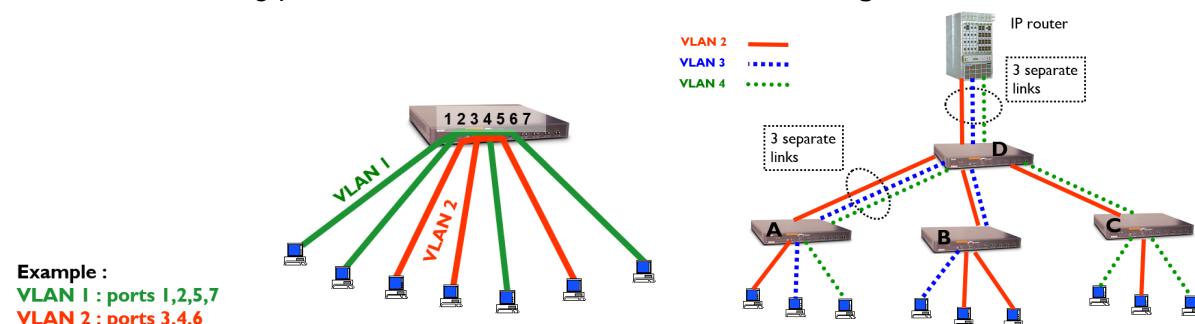
De twee meest gebruikte mechanismen zijn port-based VLAN's en tag-based VLAN's.

Daarnaast zijn er ook VLAN's per protocol of per host, maar die worden in de praktijk weinig gebruikt.

VLAN's worden bij voorkeur statisch geconfigureerd voor maximale controle. Port-based en tag-based VLAN's zijn makkelijk statisch te configureren.

Port-based VLAN's

Bij **port-based VLAN's** wordt iedere poort of interface van een ethernet switch gekoppeld aan een VLAN waartoe die behoort. VLAN's worden geïdentificeerd door een nummer (ID), standaard nummer 1. Enkel poorten die tot dezelfde VLAN (ID) behoren delen de flooding en de MAC learning processen. Dit resulteert in **verschillende logische netwerken**.

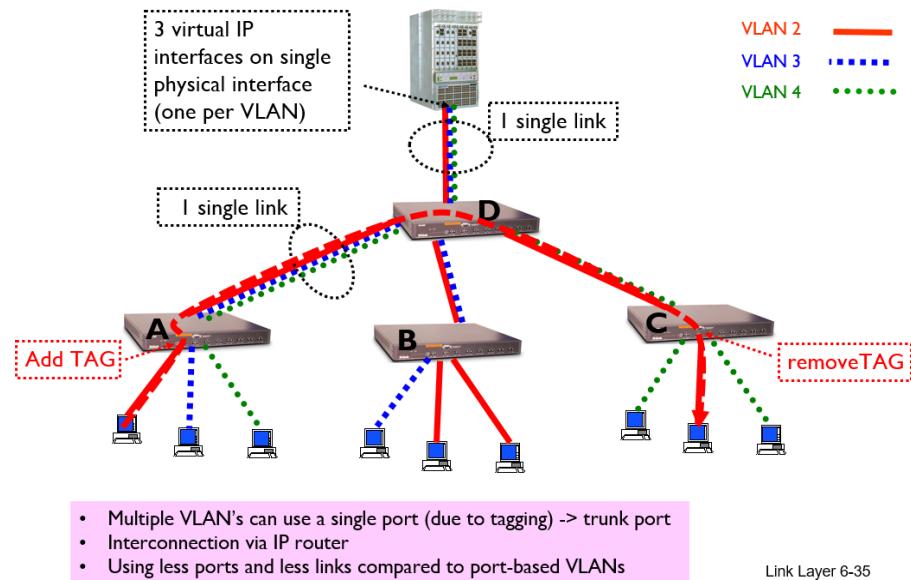


Elk LAN netwerk vormt zijn eigen subnet en de IP adressen in bijkomende subnetten zullen verschillend zijn. Wanneer een host uit de ene VLAN wenst te communiceren met een host uit een andere VLAN (ander subnet), dan kan dit enkel via een **router** die als gateway fungeert. Deze router moet dan **interfaces** hebben voor elke VLAN. Er zijn ook **meerdere links** nodig tussen de **switches** en **switch en router**. Tag-based VLAN's probeert zit op te lossen.

Tag-based VLAN's

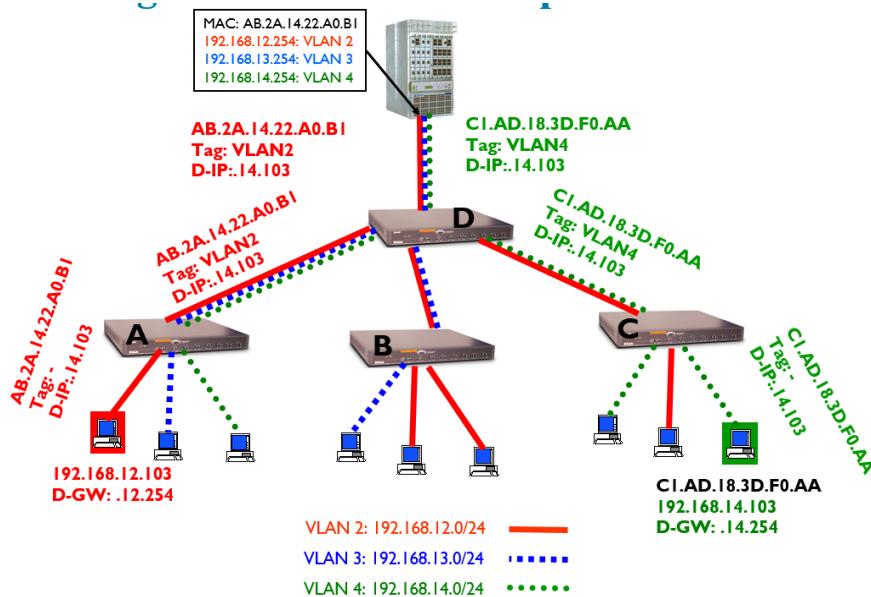
Bij **tag-based VLAN's** wordt aan de hand van een tag in een uitgebreide header (aan het ethernet frame) aangegeven tot welke VLAN het frame toebehoort. Op die manier ontstaan tagged en untagged frames. Een **switch** die kan omgaan met zulke tagged headers wordt **VLAN aware** genoemd. In de header wordt het type veld vervangen door een nieuwe type (8100) en de tag informatie.

Aan de hand van tags kan het eerdere voorbeeld efficiënter ingevuld worden.



De router is slechts met 1 interface verbonden en moet dus op een andere manier aangeven over welke VLAN hij verkeer wenst te versturen. Dit doet de router aan de hand van 3 virtuele interfaces, 1 per VLAN, waaraan elk een verschillend IP-adres wordt toegekend. Hosts zijn zich niet bewust van het feit dat er VLAN's gebruikt worden, dit verloopt volledig transparant. De switches moeten dus het frame taggen met de corresponderende VLAN. Hierdoor kunnen poorten gebruikt worden voor meerdere VLAN's, deze poorten worden **trunk poorten** genoemd.

Voorbeeld Tag-based VLAN



Iedere host in het netwerk is voorzien van een IP adres. Wanneer de host linksonder een pakket wenst te versturen naar de host rechtsonder, dan is te zien dat die hosts zich bevinden in verschillende subnetten. De subnetten worden geïmplementeerd aan de hand van verschillende VLAN's. De bron behoort tot de rode VLAN, de bestemming in de groene. Om een pakket van de bron naar de bestemming in een **ander subnet** te sturen moet gebruik gemaakt worden van de **gateway**. Wanneer het frame binnenkomt in router A wordt het getagged. De router ontvangt het pakket, stript de ethernet header en bekijkt in de netwerklaag hoe het pakket verder moet doorgestuurd worden. MAC adres wordt geresolved en dan toegevoegd aan een header en opgestuurd in het juiste subnet.

Gratuitous ARP

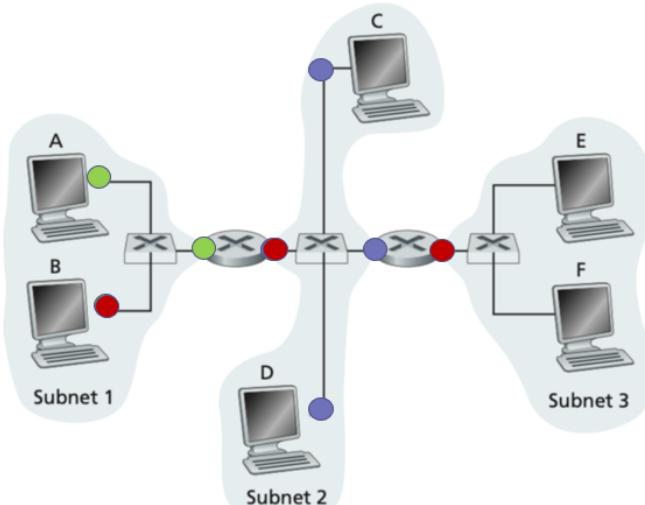
Mechanisme waarbij bepaalde ARP requests worden **gebroadcast** op een gegeven LAN netwerk. Bepaalde besturingssystemen zoals Windows gaan dat typisch doen wanneer je hen opstart. Niet voor een bepaalde onbekende bestemming, maar voor het IP adres waarvoor zij zelf eigenaar van zijn. Het misbruikt het systeem om andere apparaten **op de hoogte te stellen** (announcement) van het MAC adres met IP adres. Wanneer nodes een ARP request ontvangen gaan ze hun ARP cache lokaal updaten. Dit gaat niet met ARP responses, want die zijn unicast.

Een andere use case is wanneer je een server hebt die wordt gebacked up door een andere server. Dan kan het zijn als de primary server down gaat dat de backup server automatisch een ARP request gaat flooden met zijn eigen IP en MAC adres mapping. De andere nodes moeten dan enkel de gratuitous ARP gebruiken om hun caches te updaten.

Response college

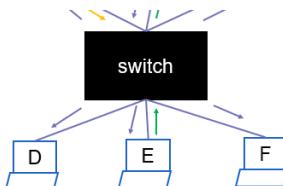
Gegeven het onderstaande netwerk. Beantwoord de volgende vragen.

- a) Hoeveel interfaces moeten een IP adres krijgen in Subnet 2? 4
- b) Hoeveel interfaces moeten een MAC adres krijgen in Subnet 1? 3
- c) Wanneer host E een pakket wenst te versturen naar host A. Veronderstel dat alle ARP tabellen/caches leeg zijn. Door hoeveel toestellen moet een ARP request uitgestuurd worden vooraleer verkeer in B zal ontvangen worden? 3
- d) Veronderstel dat alle ARP tabellen/caches leeg zijn. Wanneer host C een pakket wenst te versturen naar host F, zal C dan het MAC adres voor F opvragen via ARP? Antwoord met "JA" of "NEE". NEE



Vraag 4 Moeilijkheid: 1

Gegeven een Ethernet switch die 6 hosts A t.e.m. F in een stertopologie met elkaar verbindt. Veronderstel dat de switch net is opgestart.



a) Wanneer B een frame zendt naar E. Welke andere hosts ontvangen dan ook dit frame? Antwoord met hoofdletters in de vorm van een gesorteerde lijst zonder spaties met hoofdletters, bvb. [A,C] of [F]. flooding -> alle andere hosts

b) Als in een volgende stap (nadat B een frame naar E heeft verzonden) E antwoordt met een frame naar B, welke andere hosts ontvangen dan ook dit frame? Antwoord met hoofdletters in de vorm van een gesorteerde lijst zonder spaties met hoofdletters, bvb. [A,C] of [F]. reeds interface van B geleerd door eerste stap -> enkel B

c) Als in een volgende stap (nadat B naar E en E naar B een frame heeft verzonden) A een frame zendt naar B, welke andere hosts ontvangen dan ook dit frame? Antwoord met hoofdletters in de vorm van een gesorteerde lijst zonder spaties met hoofdletters, bvb. [A,C] of [F]. reeds interface van B geleerd door eerste stap -> enkel B

d) Als tot slot (na de vorige stappen), B antwoordt naar A, welke andere hosts ontvangen dan ook dit frame? Antwoord met hoofdletters in de vorm van een gesorteerde lijst zonder spaties met hoofdletters, bvb. [A,C] of [F].

reeds interface van A geleerd door eerste stap -> enkel A

Vraag 5 Moeilijkheid: 1

Gegeven een port-based VLAN switch met 6 poorten. Poort 1, 2 en 3 zijn toegekend aan VLAN 1, poort 4 en 5 aan VLAN 2 en poort 6 is toegekend aan VLAN 3.

Aan poort 1, 2, 4 en 6 zijn corresponderend hosts a, b, c en d verbonden. Eenzelfde router is met poort 3 en 5 van de switch verkeerverbonden, alsook met het Internet via een andere interface.

Welk van de volgende uitspraken is correct?

- ⇒ De hosts a en b zitten in hetzelfde subnet en hebben dus eenzelfde netwerkprefix.

3 (100 %)

- ⇒ Host d kan geen verkeer sturen naar a en b.

3 (100 %)

Om verkeer te versturen naar van a naar c, moet a een ARP request doen voor c's MAC adres

1 (33,33 %) / 2 (66,67 %)

Standaardafwijking: 0,4 %

Punt-biserial correlatie: nvt

Discriminatie-index: 0,00 %

a en c zitten in andere VLANs = ander subnets/link -> router nodig -> ARP enkel op zelfde link

Een ARP request van de router naar het MAC adres van host c, zal ontvangen worden door hosts a en b.

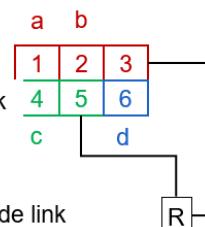
0 (0 %)

VLAN 1 : subnet 1

VLAN 2 : subnet 2

VLAN 3 : subnet 3

Virtual Local Area Network = andere link = ander subnet = ander net prefix



a and b in andere VLANs dan c = ander subnets/link -> router nodig -> ARP enkel op zelfde link

Vraag 6 Moeilijkheid: 1

Geef aan welk van de volgende uitspraken correct is met betrekking tot het Spanning Tree Protocol (STP).

Wanneer STP gebruikt wordt is MAC adress learning niet meer nodig.

0 (0 %)

MAC address learning is onafhank. van STP

- ⇒ STP zorgt ervoor dat je netwerktopologie beperkt wordt, waardoor je eigenlijk capaciteit verliest.

Shortcuts in boom kunnen niet gebruikt worden = verlies aan capaciteit

Standaardafwijking: 0,00 %

Kortste padroutering met STP is niet mogelijk.

2 (66,67 %)

Standaardafwijking: 0,00 %

Hangt af van topologie -> als bv. topologie reeds boom is, dan is STP kortste pad

Discriminatie-index: 0,00 %

- ⇒ STP zorgt ervoor dat je niet noodzakelijk het kortste pad gebruikt in je topologie.

Hangt af van topologie -> als bv. topologie reeds boom is, dan is STP kortste pad

- ⇒ Als je topologie van switches uit zichzelf een boom vormt, heb je eigenlijk geen STP nodig.

3 (100 %)

Hangt af van topologie -> als bv. topologie reeds boom is, dan is STP kortste pad

3 (100 %)

Oefeningen vanaf slide 29. Oplossing belangrijke oefening H5:

response-college-week6-net-mgt

H8: beveiliging

8.9 Firewall and IDS

Firewalls zijn toestellen die beslissen welk verkeer er wel of niet mag binnengaan in ons netwerk. Firewall is een soort van router, maar eentje die niet alle IP pakketten doorstuurt. Het zal elk IP pakket bekijken om te zien of het verkeer is dat we wensen in ons netwerk, of dat het verkeer is dat we liever blokkeren in ons netwerk.

We kunnen ervan uitgaan dat bv. verkeer dat komt vanuit ons netwerk dat dat het te vertrouwen verkeer is en dat we regels gaan opstellen van wat de gebruikers mogen doen op het internet. Het verkeer dat komt van het internet gaan we proberen vorm geven, zodat enkel het gewenste verkeer binnengaat in ons netwerk en dat het ongewenste verkeer van de 'bad guys' die zich bevinden op andere plaatsen te gaan limiteren.

Waarom firewalls (onzichtbaar)

Voorkomen van **Denial Of Service** (DOS) attacks. Bv. SYN flooding, TCP overladen zodat er geen connectie meer vrij is voor gewone gebruiker.

Voorkomen van **illegal modificatie/toegang van/tot interne data**. Bv. aanpassen van site. Enkel **geautoriseerde toegang** toestaan binnen het **interne netwerk**.

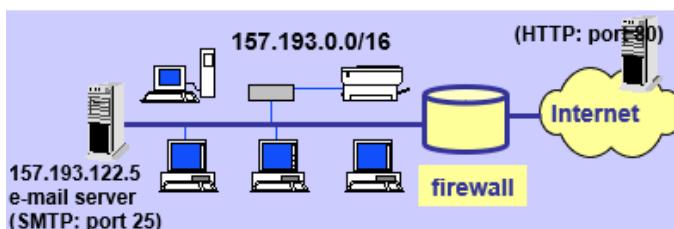
Er zijn 3 types van firewall

- stateless packet filters
- stateful packet filters
- application gateways

1) Stateless packet filters

Het gaat elk IP pakket dat passeert door de firewall, **onafhankelijk** van elkaar, **individueel**, gaan bekijken. Je kan daar **regels** voor opstellen die beslissingen gaan maken op basis van IP adressen (source en destination), port numbers (destination, source) (TCP/UDP), TCP SYN & ACK bits, ICMP message type.

Voorbeeld:



We gaan er een **Access Control List (ACL)** voor ontwerpen. Dat zijn regels die bepalen welk verkeer er wel of niet door deze firewall mag passeren.

		Forward policy:	DENY		
	Protocol	Source IP	Destination IP	Ports: src -> dest	Flag bit
ACCEPT	TCP	157.193.0.0/16	0.0.0.0/0	1024:65535 -> 80	all
ACCEPT	TCP	0.0.0.0/0	157.193.0.0/16	80 -> 1024:65535	ACK
ACCEPT	TCP	157.193.122.5/32	0.0.0.0/0	25 -> 1024:65535	all
ACCEPT	TCP	0.0.0.0/0	157.193.122.5/32	1024:65535 -> 25	all
DENY	all	all	all	all	all

→ Alle verbindingen naar de buitenwereld zijn toegestaan, op voorwaarde dat het op TCP poort 80 gebeurt. De client moet een poortnummer boven de 1024 (standaard) nemen. Dit laat toe dat een client een TCP verbinding start met bv. een webserver, ergens op het internet.

→ Omgekeerd, als we een three-way-handshake starten zal deze server naar een SYN, een SYN ACK terugsturen en ook dat verkeer moet door de firewall heen. Dus als het verkeer terugkomt van eender welk IP adres op de wereld, een webserver die verkeer stuurt vanaf poort 80, dan zal deze firewall dat binnenlaten zolang het over client ports gaat. We kunnen daar zelfs aan toevoegen dat de **ACK bit** altijd aan moet zijn. In TCP is er maar 1 boodschap die geen ACK bit heeft, en dat is het eerste SYN bericht.

We laten geen inkomende SYN berichten toe zonder ACK flag, niemand kan hier een three-way-handshake starten naar een van onze clients.

→ Stel we hebben een e-mail server staan. Dat betekent dat mailservers van over heel de wereld moeten kunnen hun e-mails pushen tot op onze server. Dat pushen gebeurt aan de hand van het SMTP protocol. Als die antwoorden stuurt, moet deze server dat kunnen sturen naar eender welk IP adres op heel de wereld. En de antwoorden sturen naar de client poortnummers die corresponderen met die verbindingen. Hier komen de SYN flags van buitenaf en kunnen we geen extra voorwaarden hieraan koppelen.

→ Eender welk IP adres vanop de wereld moet vanaf een bepaald client poortnummer verbinding opzetten naar onze server. Met zijn ene IP adres op poort 25.

→ Deny alle andere verkeer.

Deze regels worden **in volgorde** gelezen, een fall-through chain. Dat betekent dat als bv. een pakket voldoet aan regel twee, dit voldoende is en de andere regels niet meer aan bod komen.

From policy to technology

In bedrijven maken ze van bepaalde wensen, policies bepaalde regels in de firewall.

<u>Policy</u>	<u>Firewall Setting</u>
No outside Web access.	Drop all outgoing packets to any IP address, port 80
No incoming TCP connections, except those for institution's public Web server.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
Prevent Web-radios from eating up the available bandwidth.	Drop all incoming UDP packets - except DNS and router broadcasts.
Prevent your network from being used for a <u>smurf</u> DoS attack.	Drop all ICMP packets going to a "broadcast" address (eg 12.207.255.255).
Prevent your network from being tracerouted	Drop all outgoing ICMP TTL expired traffic

Stateful packet filtering

Kan onderzoeken of verkeer 'goed' verkeer is.

Het gaat niet elk pakket individueel bekijken, maar probeert **TCP verbindingen te monitoren** en op basis daarvan beslissen of verkeer wel of niet toegelaten is in het netwerk.

Voorbeeld

action	source address	dest address	proto	source port	dest port	flag bit	check connection
allow	157.193/16	0.0.0.0	TCP	> 1023	80	any	
allow	0.0.0.0	157.193/16	TCP	80	> 1023	ACK	X
deny	all	all	all	all	all	all	

→ Iedereen in het netwerk mag verkeer sturen naar de buitenwereld op poort 80.

→ Verkeer dat terugkomt moet de ACK flag hebben, maar **check connection**. Alle verkeer dat voldoet aan deze regel moeten we controleren of dat wel degelijk een TCP verbinding is afkomstig uit het interne netwerk.

Op het moment dat een van de clients een verbinding maakt naar de buitenwereld en er gaat een **SYN pakket** door die firewall, dan kunnen we vaststellen dat het een startende TCP verbinding is en het opnemen in de **allowed list**. Als er een tweede client een verbinding opstart komt die ook in de lijst terecht van gekende TCP connecties.

Source address	Destination address	Source port	Destination port
157.193.3.5	157.192.2.1	10455	80
157.193.34.4	124.23.45.2	24356	80

Verkeer dat terugkomt, dat gericht is aan een van onze clients, komende vanaf poort 80. Als het de eerste tabel is gepasseerd zal het de tweede tabel passeren om te verifiëren of dat het verkeer is dat correspondeert met een van de gekende TCP verbindingen uit dit netwerk.

Allowed: S-IP 157.192.2.1, D-IP 157.193.3.5, TCP, S-P 80, D-P 10455, ACK
 Not Allowed: S-IP 130.12.5.12, D-IP 157.193.3.5, TCP, S-P 80, D-P 20345, ACK

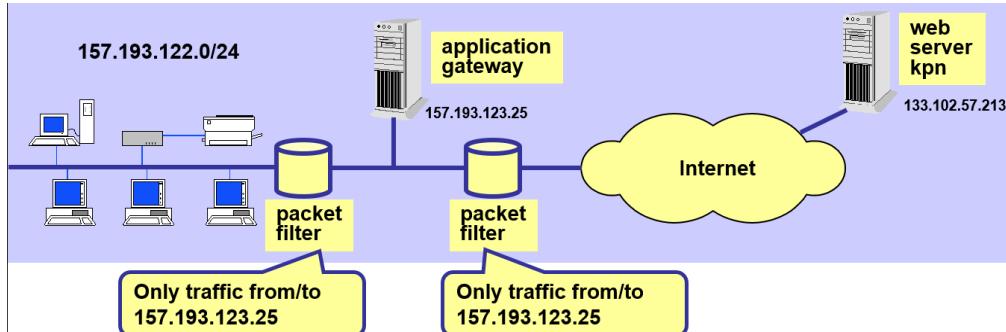
De records verdwijnen bij een FIN ACK, wanneer de TCP verbinding wordt stopgezet.

Robin De Bock

Application gateways

Enkel heel specifieke **applicatie data** die naar buiten mag. Typisch worden die **gecombineerd** met een stateless/stateful packet filter.

Voorbeeld: Web proxy

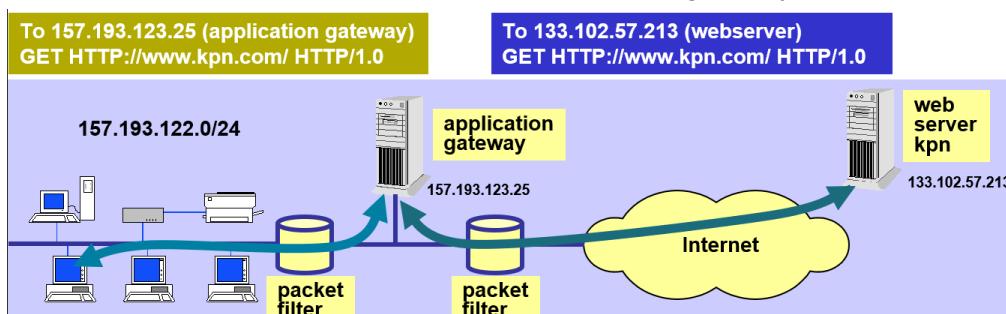


Stel dat er in ons netwerk een application gateway toestel hebben staan voor webverkeer (HTTP/HTTPS). Dan kunnen we dat gaan combineren met twee packet filters.

Eentje in het interne netwerk die verkeer naar het internet verbiedt behalve als je het stuurt naar de application gateway. Iemand die wil browsen moet de **application gateway** gebruiken (/instellen).

Een tweede regel in de **firewalls** zegt dat enkel dit application gateway toestel mag surfen op het internet.

Zo'n application gateway kan gebruikt worden als cache (niet meer relevant door hoge bandbreedtes). Ook kan het gebruikt worden e-mail server (mails scannen). Een ander voordeel is dat de buitenwereld enkel de application gateway ziet surfen.



Elke client uit het interne netwerk moet zijn verbinding doorsturen naar deze application gateway (Proxy server) en daarin meedelen dat hij wil surfen naar een webpagina. Dan gaat die server voor de client een HTTP verbinding opzetten naar de externe server. Als er een antwoord komt gaat hij dat terugsturen naar de client zelf. Een soort 'toegelaten man-in-the-middle' voor het webverkeer.

Limitations of Firewalls, Gateways (onzichtbare slide)

IP-spoofing: router kan niet weten of gegevens "echt" afkomstig zijn van geclaimde bron.

Als meerdere toepassingen een speciale behandeling nodig hebben, heeft elk een eigen toepassings gateway.

Client software moet weten **hoe ze contact moet opnemen** met de gateway. B.v. het IP-adres van de proxy in de webbrowser instellen.

Filters gebruiken vaak 'all or nothing' beleid voor **UDP**.

Afweging: mate van **communicatie** met de buitenwereld, niveau van **beveiliging**.

Veel sterk beschermd sites hebben nog steeds last van aanvallen.

Intrusion Detection Systems (IDS)

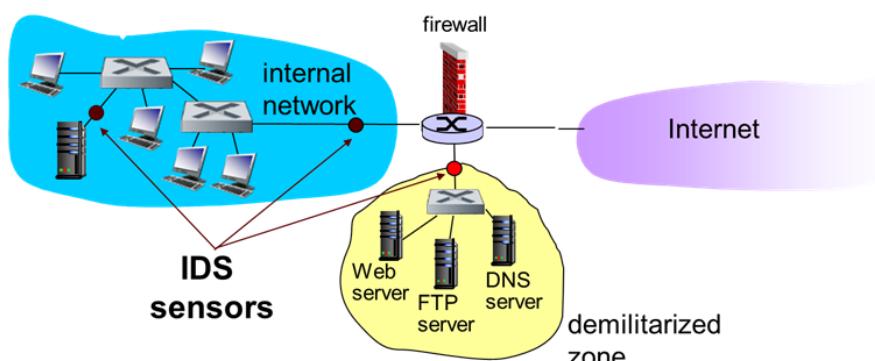
Deze drie soorten gateways beschermen het interne netwerk, omdat ze beslissen welk verkeer en wel of niet door het netwerk mag passeren. De vraag is of het verkeer dat wordt toegelaten allemaal wel 'goed' verkeer is, of gebeurt er nog iets anders dat nog niet helemaal wordt begrepen door de regels die zijn opgesteld.

Als we dit willen bewaken in het netwerk, kunnen we ook opteren om een **Intrusion Detection System (IDS)** te installeren. Deze gaan nog een stuk verder. Ze gaan niet enkel kijken naar de TCP header of de SYN flags.

Ze gaan ook proberen **correleren** tussen opeenvolgende sessies. Er zijn mogelijkheden om verkeer te gaan monitoren, het bekijken en daar statistische analyses op uitvoeren. (port scanning, network mapping, DoS attack)

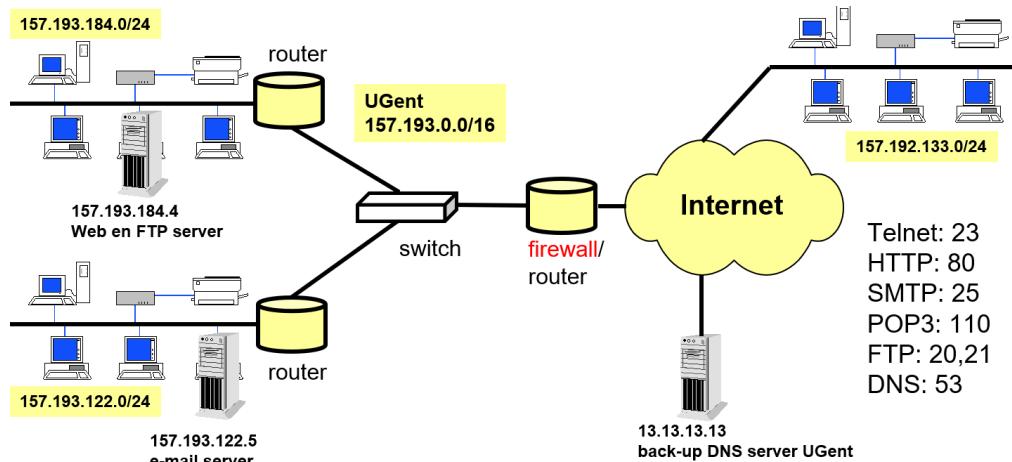
Een tweede ding dat het kan doen is om niet enkel het verkeer te laten passeren dat voldoet aan TCP en IP header, maar ook **dieper (deep packet inspection)** daarin te checken en de inhoud van het verkeer te controleren.

Plaatsing



Die network Intrusion Detection Systems kunnen op meerdere plaatsen geplaatst worden in het netwerk. Typisch worden ze geplaatst na de firewall. Als er een zone voor de servers is, een 'demilitarized' zone, dan wordt er vaak daar ook nog een voor geplaatst.

Oefening instellen packet filtering (firewall)



Standard policy : deny all traffic

Allow : - access to public web

- access to UGent web server and FTP server
- access to UGent back-up DNS server
- sending and receiving e-mail between internal and external e-mail server (from/to UGent)
- full access between 157.192.133.0/24 and 157.193.122.0/24 (except sending e-mail from e-mail server 157.193.122.5 to any server in 157.192.133.0/24)

Oplossing:

	protocol	source	destination	ports	
ACCEPT	TCP	157.193.0.0/16	0.0.0.0/0	1024:65535 > 80	public web
CCEPT	TCP	0.0.0.0/0	157.193.0.0/16	80 > 1024:65535	UGent web
CCEPT	TCP	157.193.184.4/32	0.0.0.0/0	80 > 1024:65535	UGent FTP
CCEPT	TCP	0.0.0.0/0	157.193.184.4/32	1024:65535 > 80	UGent FTP
CCEPT	TCP	157.193.184.4/32	0.0.0.0/0	20 > 1024:65535	Receive e-mail
CCEPT	TCP	0.0.0.0/0	157.193.184.4/32	1024:65535 > 20	Send e-mail
CCEPT	TCP	157.193.184.4/32	0.0.0.0/0	21 > 1024:65535	Between 2 subnets
CCEPT	TCP	0.0.0.0/0	157.193.184.4/32	1024:65535 > 21	UGent DNS back-up
CCEPT	TCP	157.193.122.5/32	0.0.0.0/0	1024:65535 > 25	No e-mail to 157.192.133.0/24
CCEPT	TCP	0.0.0.0/0	157.193.122.5/32	25 > 1024:65535	Policy: DENY
DENY	TCP	157.193.122.5/32	157.192.133.0/24	1024:65535 > 25	
DENY	TCP	157.192.133.0/24	157.193.122.5/32	25 > 1024:65535	
DENY	ALL	0.0.0.0/0	0.0.0.0/0	N/A	

ACCEPT	TCP	157.193.122.5/32	0.0.0.0/0	1024:65535 -> 25
ACCEPT	TCP	0.0.0.0/0	157.193.122.5/32	25 -> 1024:65535
ACCEPT	ALL	157.193.122.0/24	157.192.133.0/24	N/A
ACCEPT	ALL	157.192.133.0/24	157.193.122.0/24	N/A
ACCEPT	UDP	157.193.0.0/16	13.13.13.13/32	1024:65535 -> 53
ACCEPT	UDP	13.13.13.13/32	157.193.0.0/16	53 -> 1024:65535
DENY	TCP	157.193.122.5/32	157.192.133.0/24	
DENY	TCP	157.192.133.0/24	157.193.122.5/32	

Overruled by

Send e-mail

Between 2 subnets

UGent DNS back-up

No e-mail to
157.192.133.0/24



public web

UGent web

UGent FTP

UGent FTP

No e-mail to
157.192.133.0/24

Receive e-mail

Send e-mail

Between 2 subnets

UGent DNS back-up

Policy: DENY

	protocol	source	destination	ports
ACCEPT	TCP	157.193.0.0/16	0.0.0.0/0	1024:65535 -> 80
ACCEPT	TCP	0.0.0.0/0	157.193.0.0/16	80 -> 1024:65535
ACCEPT	TCP	157.193.184.4/32	0.0.0.0/0	80 -> 1024:65535
ACCEPT	TCP	0.0.0.0/0	157.193.184.4/32	1024:65535 -> 80
ACCEPT	TCP	157.193.184.4/32	0.0.0.0/0	20 -> 1024:65535
ACCEPT	TCP	0.0.0.0/0	157.193.184.4/32	1024:65535 -> 20
ACCEPT	TCP	157.193.184.4/32	0.0.0.0/0	21 -> 1024:65535
ACCEPT	TCP	0.0.0.0/0	157.193.184.4/32	1024:65535 -> 21
DENY	TCP	157.193.122.5/32	157.192.133.0/24	1024:65535 -> 25
DENY	TCP	157.192.133.0/24	157.193.122.5/32	25 -> 1024:65535
ACCEPT	TCP	157.193.122.5/32	0.0.0.0/0	25 -> 1024:65535
ACCEPT	TCP	0.0.0.0/0	157.193.122.5/32	1024:65535 -> 25
ACCEPT	TCP	157.193.122.5/32	0.0.0.0/0	1024:65535 -> 25
ACCEPT	TCP	0.0.0.0/0	157.193.122.5/32	25 -> 1024:65535
ACCEPT	ALL	157.193.122.0/24	157.192.133.0/24	N/A
ACCEPT	ALL	157.192.133.0/24	157.193.122.0/24	N/A
ACCEPT	UDP	157.193.0.0/16	13.13.13.13/32	1024:65535 -> 53
ACCEPT	UDP	13.13.13.13/32	157.193.0.0/16	53 -> 1024:65535
DENY	ALL	0.0.0.0/0	0.0.0.0/0	N/A