High Availability VPN solutions

Guus Sliepen & Ivo Timmermans

23 april 2001

1 Inleiding

Er bestaat geen twijfel over dat Virtual Private Networks steeds belangrijker zullen worden. Er is een groot aantal VPN software pakketten te krijgen, zowel commercieel als Open Source, dat in die behoefte voorziet. Tevens wordt VPN functionaliteit al standaard in veel Operating Systems meegeleverd. Maar veel van deze oplossingen waren ontworpen met een kleine, statische netwerken als uitgangspunt en zijn gebaseerd op point-to-point verbindingen. In grote VPNs is performance en High Availability veelal een schalingsprobleem.

1.1 Begrippen

We zullen de volgende termen gebruiken:

Virtueel adres Een IP adres in het VPN. VPNs gebruiken meestal adressen uit één van de drie gereserveerde subnetten voor privé gebruik: `10/8`, `172.16/12` en `192.168/16`. De adressen in deze subnetten bestaan niet voor het Internet, en zullen dus ook nooit door het Internet gerouteerd worden.

Host Een computer die meedoet in een VPN. Deze heeft een virtueel IP adres, maar hoeft zelf geen VPN software te draaien. Een host kan bijvoorbeeld in een LAN zitten waar een gateway staat die het lokale subnet met het VPN verbindt.

Node Een computer die meedoet in een VPN, VPN software draait en normaliter een eindpunt is van een tunnel.

Tunnel Een verbinding tussen twee nodes.

2 High Availability problemen in netwerken

2.1 "Echte" netwerken

In echte netwerken wordt vooral gekeken naar de *verbindingen* tussen twee punten in een netwerk. High Availability betekent dan vooral het aanleggen van extra kabels en zorgen dat er routing daemons draaien. Op deze manier zal de uitval van één verbinding niet zorgen voor een onderbreking van het netwerkverkeer, omdat het omgeleid wordt via een andere kabel of zelfs een geheel andere route.

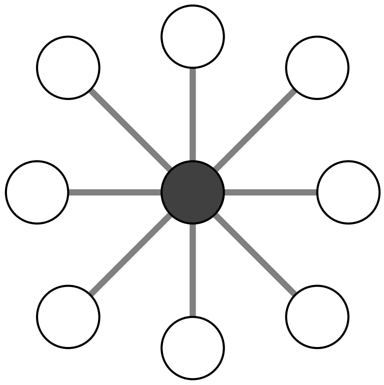
Bekende routing protocollen zijn onder andere RIP, BGP en OSPF.

2.2 VPNs

De analogie van een netwerkkabel in VPNs is natuurlijk de tunnel. Dit is echter geen perfecte analogie. Terwijl tussen twee punten in een echt netwerk duidelijk vele kabels kunnen liggen, zal het aanmaken van meerdere tunnels tussen twee nodes in een VPN geen redundantie geven. Waarom niet? Dit komt omdat een tunnel alleen gedefinieerd is door zijn twee eindpunten. De eindpunten worden gekenmerkt door IP adressen. De route die het VPN verkeer over het echte netwerk zal nemen wordt dan ook niet bepaald door de tunnel of de VPN software in het algemeen, maar het wordt bepaald door de route die "echt" netwerkverkeer zou nemen van het ene IP adres naar het andere IP adres. Hier wordt dus duidelijk dat tussen twee gegeven punten de redundantie alleen door redundantie in het echte netwerk gegarandeerd kan worden.

Het probleem bij VPNs komt naar voren wanneer nodes zelf uitvallen. Op het Internet is het uitvallen van een computer pech, maar zal de bereikbaarheid van andere computers meestal niet veranderen als dat gebeurt. Dit komt omdat op het Internet de meeste computers "leaf nodes" zijn, ofwel kort gezegd geen router zijn. Bij VPNs zal het echter zo zijn dat een aanzienlijk gedeelte van de nodes een routerfunctie hebben. Als een node het verkeer tussen twee andere nodes moet routeren, dan zal door de uitval van die ene node geen verbinding meer opgezet kunnen worden tussen de twee andere nodes.

Voorbeeld 1 (Sternetwerk) Neem als VPN topologie een sternetwerk. Deze wordt gekenmerkt door het feit dat er één node in het centrum is die een verbinding heeft met alle andere nodes, die op hun beurt alleen een verbinding met de node in het centrum hebben (zie figuur 1). In de praktijk komt deze topologie in VPNs vrij vaak voor, denk aan de VPN server van een klein bedrijf in het centrum en inbellende thuiswerkers die een verbinding met de server aangaan.



Figuur 1: Sternetwerk

Als alles goed functioneert zullen alle nodes met elkaar kunnen communiceren. Als er een willekeurige node (behalve die in het centrum) uitvalt, zal dat geen negatieve effecten hebben op de rest van het VPN. Als echter de node in het centrum uitvalt, dan is er met geen enkele node communicatie meer mogelijk!

2.3 Spanning tree vs. full mesh

Om er nu voor te zorgen dat geen enkele node verantwoordelijk kan zijn voor de uitval van het hele of een gedeelte van het VPN, moeten alle nodes er dus voor zorgen dat ze met minstens twee en liefst meer andere nodes een verbinding hebben. Om te garanderen dat voor elke mogelijk subset van nodes die uitvalt de rest (het complement van die subset) gewoon door kan werken, moet elke node een verbinding hebben met alle andere nodes in het VPN! Dit wordt ook wel een "full mesh" netwerk genoemd, in tegenstelling tot een "spanning tree", waarbij zo min mogelijk verbindingen zijn om toch een coherent netwerk te vormen.

Elke verbinding tussen twee nodes wordt gerealiseerd als een tunnel. Om een idee te geven van het aantal tunnels dat nodig is voor zowel een spanning tree als een full mesh netwerk is de volgende tabel: Waarom zorgen we niet gewoon dat alle VPNs een full mesh topologie hebben? Het probleem

# nodes	# tunnels (spanning tree)	# tunnels (full mesh)
2	1	1
3	2	3
4	3	6
5	4	10
10	9	45
100	99	4950
1000	999	499500
n	$n-1$	$\frac{1}{2}n(n-1)$
orde:	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$

Tabel 1: Schaalbaarheid van VPNs

is dat normaal gesproken elke tunnel apart geconfigureerd moet worden. Het aanmaken van één tunnel betekent meestal dat er op twee nodes configuratie gewijzigd moet worden.

Voorbeeld 2 (Uitbreiden VPN) Gegeven een bestaand netwerk van $n = 10$ nodes. Indien er een spanning tree topologie gebruikt wordt, zal het toevoegen van een nieuwe node in het VPN een kwestie zijn van het opzetten van een tunnel tussen een reeds bestaande node en de nieuwe node. Dit is een $\mathcal{O}(1)$, dus makkelijk op te lossen, probleem.

Indien er een full mesh topologie gebruikt wordt, zal het toevoegen van een nieuwe node betekenen dat er voor elke reeds bestaande node een tunnel moet worden opgezet naar de nieuwe node. Dit is een $\mathcal{O}(n)$ probleem, nog te doen bij kleine aantallen, maar erg lastig als het er meer worden.

Het probleem met VPNs is dat het bijhouden van redundante tunnels bij grote VPNs erg tijdrovend is (zie voorbeeld 2). Om precies te zijn: gegeven n nodes en m redundante tunnels per node (waarbij $1 \leq m < n$) is het een $\mathcal{O}(n \cdot m)$ probleem. We zien ook dat het zich hier om een *administratief* probleem betreft.

2.4 Routering VPN verkeer

Ook al wordt redundantie veelal geassocieerd met extra overhead, in het geval van VPNs zal een full mesh netwerk juist leiden tot *minder* overhead dan een spanning tree topologie. We nemen hier aan dat de hoeveelheid netwerkverkeer voor het opzetten en onderhouden van een tunnel natuurlijk verwaarloosbaar is tegenover de hoeveelheid daadwerkelijk VPN verkeer dat er doorheen gestuurd wordt.

In het geval van een spanning tree topologie zal niet elk paar nodes een directe verbinding hebben. In plaats daarvan wordt een route gebruikt die over een aantal andere nodes loopt. Deze tussenliggende nodes zullen het verkeer dan moeten ontvangen, verwerken en doorzenden. Dit kost de tussenliggende nodes CPU tijd en zal zowel de latency verhogen als de bandbreedte[1] verlagen. De route van het

[1] Vanwege het principe van de zwakste schakel.

VPN verkeer over het echte netwerk zal dan waarschijnlijk veel langer zijn dan dat van niet-VPN verkeer dat tussen de twee nodes over het echte netwerk zou gaan.

Bij full mesh netwerken is er tussen elk paar nodes een directe verbinding. Het VPN verkeer neemt daardoor de kortste route over het echte netwerk. Het resultaat is een verbeterde latency en een vergrote bandbreedte.

2.5 High Availability oplossingen

Het eerste idee wat men krijgt om VPNs van redundantie te voorzien is om gewoon extra tunnels aan te maken en om over het gehele VPN routing daemons te draaien om het routing probleem op te lossen. Dat werkt natuurlijk wel, maar dit is een oplossing soortgelijk aan protocol stapeling (zie 3.1). Er moet dan namelijk een heleboel ineens dubbel gedaan worden. Zowel de VPN software moet op elke node geconfigureerd worden voor elke tunnel die ze aanmaken, als de routing software.

Routing daemons genereren per link ruwweg 1 kb/s aan netwerkverkeer[2], enkel en alleen om te controleren of alles nog werkt. Dit betekent voor het gehele netwerk dat er $1 \cdot n \cdot m$ kb/s aan verkeer geproduceerd wordt. Deze hoeveelheid verkeer lijkt klein maar is constant aanwezig, wat wellicht niet erg praktisch is als men zo'n VPN over lage bandbreedte (bijvoorbeeld telefoon- of kabelmodem) links draait, of als er kosten per hoeveelheid netwerkverkeer gerekend worden. Ook betekent dit misschien dat de kosten van het onderhouden van een tunnel niet meer verwaarloosbaar zijn tegenover de hoeveelheid verkeer die er doorheen gestuurd wordt.

Voorts is het verkeer dat een routing daemon gegenereerd vaak volkomen zinloos. De meeste VPN software zelf weet immers ook wanneer een tunnel down is. De VPN software zelf zou veel efficiënter als routing daemon op redundante links kunnen fungeren. Ook hier geld weer dat voor een full mesh netwerk dit voor de VPN software een gedegenereerd geval is: als in een full mesh netwerk een tunnel down gaat hoeft er immers toch geen verkeer omgeleid te worden, want dit komt omdat de node aan de andere kant zelf down is. Vergelijk dit met het gebruik van routing daemons op een full mesh netwerk, dan zou er tevergeefs geprobeerd worden om toch nog een route naar de onbereikbare node op te zetten, terwijl daarvoor ondertussen $1 \cdot n^2$ kb/s aan verkeer door diezelfde daemons geproduceerd werd.

3 Performance problemen

Ook al heeft performance strikt gesproken niets met High Availability te maken, toch zijn High Availability systemen in de praktijk meestal ook High Performance systemen. Tevens is er wel degelijk sprake van Availability problemen in netwerken als de performance erg slecht is. Als bijvoorbeeld de round-trip tijd van een pakket langer is dan de time-out die er voor staat, dan is er duidelijk een probleem. TCP is redelijk bestendig tegen lange round-trip tijden, maar sommige op UDP gebaseerde protocollen als DNS en RADIUS zijn minder vergevingsgezind.

3.1 Protocol stapeling

Veel VPN software hergebruikt reeds bestaande netwerkprotocollen om te kunnen functioneren. Zo is het onder alle UNIX varianten is het een koud kunstje om even een PPP over SSH tunnel op te zetten, en zo snel een klein VPN te vormen. SSH zorgt voor de tunnel, authenticatie en encryptie,

[2] Men kan bijvoorbeeld simpel alle tunnels d.m.v. bridging aan elkaar knopen en daarover het 802.1d Spanning Tree Protocol draaien.

PPP zorgt voor het transport van IP pakketten (en in sommige gevallen ook voor authenticatie). Ook andere VPN oplossingen gebruiken dit soort technieken.

Voorbeeld 3 (PPP over SSH) Het opzetten van een PPP over SSH verbinding is erg simpel. De techniek staat beschreven in de VPN mini-howto die je kan vinden op `http://www.linux.org/docs/ldp/howto/mini/VPN.html`. In het kort komt het erop neer dat met de hulp van een klein programmaatje genaamd `pty-redir` een tunnel maakt met slechts twee commando's:

```
# ./pty-redir ssh remotehost pppd local passive 192.168.1.1:
# pppd /dev/ttyp0 local 192.168.1.2:
```

Helaas is het stapelen van protocollen toch erg inefficiënt. Ten eerste moet tijdens het opzetten van een tunnel elk protocol een start-up fase doorlopen. Voor PPP over SSH moet er een TCP verbinding worden opgezet, SSH uitwisseling van capabilities, authenticatie en aanvraag van pty's, en PPP authenticatie (PAP/CHAP), uitwisseling IP adressen (IPCP), link configuratie (LCP) en compressie.

Ten tweede wordt voor elk gebruikte protocol de header van een VPN pakket een stukje groter gemaakt. Tevens zijn er protocollen die voor elk ontvangen pakketje een ACK terugsturen. In het geval van PPP over SSH gaat de tunnel zelf over een TCP verbinding. Stel nu dat we op het VPN ook een TCP verbinding starten, die over deze tunnel loopt. Dan zal voor elk VPN-TCP pakket een tunnel-TCP pakket verstuurd worden. Na ontvangst van het tunnel-TCP pakket zal er een tunnel-ACK teruggestuurd worden. Tevens zal uit het tunnel-TCP pakket het VPN-TCP pakket gehaald worden. Na het verwerken van het VPN-TCP pakket zal er een VPN-ACK pakket teruggestuurd worden. Het VPN-ACK pakket zit ingekapseld in een tunnel-TCP pakket, waarop na ontvangst weer een tunnel-ACK wordt teruggezonden!

Ten derde zal elk protocol in meer of mindere mate proberen de verbinding in stand te houden, sommige protocollen zullen ook proberen om verloren gewaande pakketten opnieuw te verzenden. Bedenk wat er gebeurt als er op het laagste niveau een pakket dat bijvoorbeeld een ACK van het hoogste niveau verloren gaat. In het geval van PPP over SSH zal na een time-out de onderste laag proberen deze ACK nogmaals te verzenden, terwijl op de bovenste laag door *dezelfde* time-out de andere kant op het hoogste niveau zal proberen om het datapakket opnieuw te verzenden. De door de onderste laag opnieuw gestuurde ACK moet aan de andere kant weer geACKt worden, en tevens wordt er opnieuw voor één datapakket 4 netwerkpakketjes verzonden.

Het hierboven beschreven probleem is zelfs iets erger omdat ook PPP in enige mate aan resends doet. Als iemand deze PPP over SSH verbinding over een modemverbinding heeft opgezet, komen er dus nog twee lagen bij.

Indien men een goede netwerkverbinding heeft zonder al te veel verlies van pakketten is er niet zoveel aan de hand. Als het netwerk echter van wat minder goede kwaliteit is (en helaas is het bij de meeste ISPs in de piekuren geen pretje) zorgt de combinatie van vele resends en lange headers ervoor dat een groot deel van de beschikbare bandbreedte verspild wordt.

3.2 Efficiëntere protocollen

De oplossing voor het probleem van protocol stapeling is eenvoudig: Gebruik zo min mogelijk protocollen. Liefst stuur je de VPN pakketjes zonder extra headers direct naar de bestemming. Raw IP is dan voor tunnels het meest ideale, UDP is echter ook een goede oplossing.

Het klinkt misschien tegenstrijdig dat een tunnel met minder garanties en foutcorrecties betere en meer foutloze prestaties zal leveren. Het maakt echter niet uit op welke laag er aan foutcorrectie wordt gedaan. De applicaties van de eindgebruiker zullen meestal al het juiste protocol (vaak TCP) gebruiken, dus hoeft de VPN software hier geen moeite voor te doen.

4 Bestaande oplossingen

Er zijn veel softwarepakketten en implementaties van VPNs beschikbaar. Het gaat te ver om ze allemaal een eerlijke kans te geven en hier te beschrijven, daarom worden er hier slechts twee beschreven, één vanwege het feit dat deze zeer bekend is, de ander omdat de auteurs van dit artikel het geschreven hebben.

4.1 IPsec

IPsec is voor kleine VPNs het meest efficiënte protocol[3]. Door direct op de IP laag te werken is er geen onnodige overhead of foutcorrecties. Bij grotere VPNs heeft IPsec (in de tunnel mode) dezelfde problemen als beschreven in 2.3. Het configureren van alle tunnels en eventueel het moeten draaien van routing daemons zorgt voor de nodige configuratieproblemen.

4.2 TINC

TINC is een userspace VPN implementatie voor Linux[4], die gekenmerkt wordt door het feit dat zonder zelf veel te hoeven configureren er toch een full mesh netwerk opgebouwd wordt. De huidige versie zet controleverbindingen op via TCP. Deze TCP verbindingen dienen wel zelf geconfigureerd te worden. Hierbij kunnen voor redundantie meerdere controleverbindingen in de configuratie gedefinieerd worden. TINC zal zelf als routing daemon fungeren over deze controleverbindingen. De TCP verbinding wordt gebruikt om informatie over andere nodes uit te wisselen. Elke TINC node weet daardoor het IP adres van alle andere nodes. Via UDP worden VPN netwerkpakketten direct van de ene node naar de andere gestuurd. Er wordt dus een efficiënt netwerkprotocol gebruikt, en tevens ontstaat er zo een full mesh voor het VPN verkeer.

TINC is volop in ontwikkeling, maar tevens bruikbaar. De website met informatie en source code is te vinden op `http://tinc.nl.linux.org/`.

5 Conclusie

Het klakkeloos gebruiken van standaardprotocollen om VPNs mee te bouwen kan vanwege effecten door stapeling tot slechte performance leiden. De oplossing is om juist zo spartaans mogelijk te zijn.

High Availability in VPNs wordt gerealiseerd door tunnels tussen zoveel mogelijk paren nodes aan te leggen. Een full mesh netwerk is het beste, zowel voor de High Availability als wel de performance zelf.

[3] N.B.: Er zijn ook VPN pakketten die IPsec gebruiken voor encryptie alleen. Het gaat hier echter om enkel en alleen het direct gebruik van IPsec.

[4] Met kleine aanpassingen kan het ook op FreeBSD en Solaris draaien.