Evaluatie van de schaalbaarheid van SNMP-gebaseerde netwerkbevraging

Daan Volcke

Promotoren: prof. Joris Moreau, dhr. Nico Wauters (Networkmining)

Begeleiders: dr. Wouter Tavernier, prof. dr. ir. Didier Colle, dhr. Leo Nederlof (NetworkMining)

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Vakgroep Industriële Technologie en Constructie
Voorzitter: prof. Marc Vanhaelst
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2014-2015

UNIVERSITEIT GENT

Evaluatie van de schaalbaarheid van SNMP-gebaseerde netwerkbevraging

Daan Volcke

Promotoren: prof. Joris Moreau, dhr. Nico Wauters (Networkmining)

Begeleiders: dr. Wouter Tavernier, prof. dr. ir. Didier Colle, dhr. Leo Nederlof (NetworkMining)

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Vakgroep Industriële Technologie en Constructie
Voorzitter: prof. Marc Vanhaelst
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2014-2015

UNIVERSITEIT GENT

Woord vooraf

Alvorens we beginnen zou ik graag een aantal mensen bedanken. Vooreerst mijn dank aan mijn promotoren Joris Moreau, Didier Colle en Nico Wauters als partner van NetworkMining. Zonder hen was deze masterproef niet mogelijk geweest.

Ook mijn begeleiders Wouter Tavernier en Leo Nederlof, tevens partner van NetworkMining, wil ik bedanken voor hun technische ondersteuning en advies. Mijn dank gaat verder uit naar alle andere mensen bij iMinds die mij geholpen hebben, maar in het bijzonder naar Sahel Sahhaf voor haar cruciale hulp bij het opzetten van een grootschalige testopstelling.

Tenslotte gaat mijn dank nog uit naar mijn familie en vrienden voor hun steun en de nodige luchtige momenten. Onder mijn vrienden vormden Bart, Ineke en Stijn Doyen bovendien ook mijn bevoorradingslijn vanuit het thuisfront, toen ik als een kluizenaar in Gent verbleef om mijn masterproef af te werken. En zonder mijn ouders had ik natuurlijk niet de middelen gehad om te kunnen studeren.

Aan iedereen hierboven gaat mijn oprechte dank uit!

Daan Volcke
Gent, 2015

Abstract

Nederlands

SNMP biedt een uniforme manier aan om informatie op te halen over netwerkcomponenten zoals routers en switches, ongeacht de fabrikant. Er zijn veel toepassingen van SNMP maar enkele belangrijke zijn de inventarisatie en configuratie- en performantiebeheer ervan. In een grootschalig netwerk is het ondenkbaar dat er geen gebruik gemaakt wordt van tools die van deze gegevens gebruik maken.

In dit werk wordt onderzocht hoe een bestaande tool die momenteel gebruikt wordt om kleine tot middelgrote netwerken te bevragen via SNMP, kan ingezet worden op grootschalige netwerken. Daarvoor wordt een diepgaande analyse uitgevoerd van alle aspecten die hierbij betrokken zijn, van software- tot netwerkniveau, om problemen bloot te leggen die de schaalbaarheid negatief beïnvloeden.

Eenmaal die problemen vastgesteld zijn worden mogelijke oplossingen besproken, geïmplementeerd en wordt de effectiviteit ervan geëvalueerd.

Kernwoorden: SNMP, bevraging, retrieval, retriever, schaalbaarheid, Net-SNMP, analyse, performantie, berichtstructuur, profiler, netwerkvertraging, bulk, optimalisatie

English

SNMP offers a uniform way to retrieve information about network components such as routers and switches, regardless of the manufacturer. There are many applications of SNMP, but a few important ones are stocktaking and the configuration and performance management of those network components. In a large scale network it is unthinkable that tools which make use of this information would not be used.

This work researches how an existing tool that's currently used to query small to medium sized networks via SNMP, can be used in large scale networks. To that end, a thorough analysis is done of all aspects involved, from software to network level, to find problems that negatively affect the scalability.

Once those problems are identified, solutions are discussed, implemented and their effectiveness evaluated.

Keywords: SNMP, querying, retrieval, retriever, scalability, Net-SNMP, analysis, performance, message structure, profiler, network latency, bulk, optimisation

Inhoudsopgave

Lijst van tabellen

Lijst van figuren

Codefragmenten

Woordenlijst

ARP-tabel ARP staat voor het Address Resolution Protocol. Met behulp van ARP kan een toestel het unieke hardware-adres (MAC-adres) te weten komen dat bij een IP-adres hoort van een ander toestel.[1]. 15, 28

SNMP-agent Een SNMP-agent vervult de serverrol bij SNMP-protocol. Ze houdt de informatie bij over een toestel en beantwoordt requests die om die informatie vragen. Zie ook paragraaf 2.2.2.. 16, 17, 20, 22, 23, 25, 32, 36–39, 41, 42, 55, 56, 63

Lijst van gebruikte afkortingen

cgroups control groups. 56

DoS Denial of Service. 68

LLDP Link Layer Discovery Protocol. 33, 56

LXC Linux Containers. 55, 69

MDB Manageable Objects Database. 17

MIB Management Information Base. 17, 18, 20, 21, 29, 30, 34, 41, 52

MTU Maximum Transmission Unit. 25, 52, 53

NAT Network Address Translation. 33

NMS Network Management Systeem. 16, 17, 20, 21

OID Object Identifier. 17–27, 29–32, 34, 36–38, 41, 44, 45, 48, 50, 52, 53, 57, 66, 67

PDU Protocol Data Unit. 20, 21, 26, 27, 53

RID Relative Identifier. 17, 19

STP Spanning Tree Protocol. 33

TLV Tag-Length-Value. 26, 27, 53

TPL Task Parallel Library. 71

Hoofdstuk 1

Inleiding

1.1 Situering

Bestaande (open-source) softwaretools voor het opvragen van netwerkinformatie via het SNMP-protocol beschikken over te weinig intelligentie om grootschalige netwerken te ondervragen. Hierdoor is het moeilijk om op frequente basis datamining van het netwerk te doen, om bijvoorbeeld de netwerkconnectiviteit of netwerkroutering in kaart te brengen. Het efficiënt kunnen nagaan van configuratiefouten van grootschalige Ethernet- en IP-netwerken is nochtans een must voor elke netwerkbeheerder.

Bij het bedrijf NetworkMining, dat zich profileert als een onafhankelijke softwareleverancier voor transportnetwerken, wordt er aan netwerkbevraging gedaan via het SNMP-protocol met behulp van een intern ontwikkelde tool. Bij het aanvatten van de masterproef werd deze tool enkel ingezet op kleine en middelgrote netwerken waarbij performantie geen belangrijke rol speelde. Daarom werd de tool zo eenvoudig mogelijk gehouden. NetworkMining zou echter de tool willen inzetten op grootschalige netwerken, waardoor de nodige aanpassingen moeten gedaan worden om de bevraging zo efficiënt mogelijk te maken. Naast de omvang van het netwerk komt daar ook bij dat het interessant zou zijn om de bevraging periodiek te kunnen uitvoeren, zodat historische data over het netwerk kan bijgehouden worden.

1.2 Probleemstelling

Fabrikanten van routers en switches voorzien nu al network management systemen die het leven van een netwerkbeheerder makkelijker maken. Deze systemen kunnen onder andere geaggregeerde informatie van de verschillende netwerkelementen rapporteren. Het probleem hierbij echter is dat deze managementsystemen meestal enkel werken voor netwerkelementen van dezelfde fabrikant. Grootschalige netwerken bestaan doorgaans echteruit apparatuur van verschillende fabrikanten, typisch bepaald door het afwegen van de kostprijs en features die een bepaalde fabrikant aanbiedt op het moment dat een netwerk uitgebreid wordt. Maar ook het raadplegen van verschillende network management systemen is niet uniform: er wordt gebruik gemaakt van verschillende API's en technologieën zoals XML SOAP en CORBA. Ze bieden bovendien niet altijd alle informatie aan die de netwerkbeheerder wenst. Vanwege dit heterogeen landschap werd ervoor geopteerd om gebruik te maken van het SNMP-protocol voor het opvragen van netwerkinformatie. Dit protocol wordt wel door apparatuur van alle fabrikanten ondersteund en biedt een enigszins uniform alternatief. Via het SNMP-protocol worden echter niet dezelfde aggregatiemogelijkheden aangeboden als door

een network management systeem. Enkel de ruwe informatie van individuele netwerkcomponenten kan via SNMP opgevraagd worden, die dan verder verwerkt en geaggregeerd moet worden.

1.3 Doelstelling

De SNMP Data Retriever die hierboven werd beschreven, wordt vandaag enkel gebruikt op kleine- tot middelgrote netwerken. De bedoeling is om deze software ook te kunnen inzetten op grootschalige netwerken, waar ze haar nut het meest kan bewijzen. Men kan echter niet zondermeer de bestaande software inzetten op die grote netwerken: er komen veel zaken bij kijken waar dat bij kleinere netwerken niet het geval was. Deze moet men dan vooral zoeken in de richting van performantieproblemen. In het verleden was er weinig noodzaak aan performantie en opteerde men eerder voor simpliciteit. De bedoeling van de masterproef bestaat erin om de schaalbaarheid te onderzoeken van de SNMP-bevragingen met de bestaande software. Hierbij moet er gezocht worden naar mogelijke bottlenecks die zich voordoen. Dit kan gaan om de CPU van de client, bandbreedteproblemen, de databank die de opgevraagde gegevens moet opslaan maar niet kan volgen, of het netwerkelement zelf die niet snel genoeg is. Eens de bottlenecks geïdentificeerd zijn, moet er gezocht worden naar oplossingen om de bottlenecks te verhelpen. Denk aan aanpassingen aan de software, zoals het implementeren van multithreading, gelijktijdig gebruik van meerdere SNMP clients of het opzetten van een databankcluster. Om te zien hoe effectief de oplossingen zijn, zal er ook een testmethode/benchmark opgesteld moeten worden om dit te meten.

Hoofdstuk 2

SNMP

2.1 Inleiding

SNMP staat voor het Simple Network Management Protocol. De naam vertelt je meteen al waarvoor het protocol dient: het beheren van je netwerk. De 'S' van SNMP betekent niet simpel als in simpel in gebruik (alhoewel het niet moeilijk in gebruik is), maar dat het protocol zo eenvoudig mogelijk gehouden is. Met SNMP kun je allerhande informatie opvragen van verschillende soorten toestellen. In principe is er geen limiet op hetgeen je kunt opvragen van een toestel, maar de functionaliteit om die informatie op te vragen moet wel geïmplementeerd worden. Normaal gezien gebeurt dat door de fabrikant van het toestel.

Er zijn maar een paar voorwaarden om met behulp van SNMP informatie van een toestel op te kunnen vragen. Eerst en vooral moet het toestel natuurlijk SNMP ondersteunen. Ze moet ook de functionaliteit bezitten om de gevraagde informatie te kunnen aanbieden en tenslotte moet de vrager ook kennis hebben van welke informatie er allemaal opgevraagd kan worden. Zo zal een router niet dezelfde informatie kunnen aanbieden als bijvoorbeeld een switch. Er is wel een minimale verzameling van gegevens die door alle toestellen ondersteund moet worden om als SNMP-compatibel bestempeld te kunnen worden. Denk hierbij aan de naam van het systeem, hoe lang het systeem al online is, welke netwerkinterfaces ze heeft, enzovoort.

Er zijn een groot aantal use cases te bedenken voor het gebruik van SNMP. Hier zijn enkele van de belangrijkste.

- Inventarisatie: met behulp van discovery procedures en SNMP kun je een overzicht krijgen van alle toestellen aangesloten op het netwerk, en hoe ze met elkaar verbonden zijn. Netwerken zijn niet statisch: er worden toestellen toegevoegd en verwijderd. Met SNMP kun je een actueel beeld van de aangesloten hardware en de netwerktopologie verkrijgen. Dit is zeer handig voor kleine en middelgrote netwerken, en zelfs onmisbaar voor grote netwerken!
- Configuratiebeheer: met SNMP kun je de configuratieinstellingen van toestellen opvragen en controleren. Bijvoorbeeld: voor Windowstoestellen kun je de lijst opvragen van geïnstalleerde updates, van routers kun je de routetabel controleren, van switches de ARP-tabel, enzovoort.
- Performantiebeheer: met SNMP kun je toezicht houden op de netwerkverzadiging. Zo kun je de huidige toestand in de gaten houden, onregelmatigheden vaststellen en trends volgen. Op basis van deze trends kunnen dan plannen opgesteld worden voor toekomstige uitbreidingen van de netwerkcapaciteit op knelpunten in het netwerk.

Voor kleine netwerken kan SNMP het leven van een systeembeheerder een stuk gemakkelijker maken. Om grote netwerken te beheren is SNMP quasi een vereiste. De conclusie is duidelijk:

SNMP zou onderdeel moeten uitmaken van de gereedschapskit van elke systeembeheerder.

2.2 SNMP meer in detail

2.2.1 De S van SNMP

Zoals gezegd staat de 'S' van SNMP voor simple. Hiermee wordt vooral bedoeld dat men SNMP zo simpel mogelijk heeft gehouden. De reden waarom dit zo is, is tweeledig: eenvoud van implementatie en met het oog op performantie. Omdat SNMP zo simpel in elkaar steekt, kunnen fabrikanten zonder al te veel moeite SNMP implementeren op hun hardware. Door de simpliciteit van SNMP moet er bovendien slechts weinig werk verricht worden om SNMP-requests te beantwoorden. Zodoende kunnen zelfs toestellen met zeer elementaire hardware, zoals embedded systemen, ook SNMP ondersteunen.

2.2.2 SNMP-toestelrollen

SNMP kent twee soorten toestellen: SNMP-agents en SNMP-managers. In het client-server model wordt de serverrol verricht door de SNMP-agent en de client door de SNMP-manager.

- De SNMP-agent is een stuk software dat draait op de netwerkcomponenten en informatie bijhoudt over het toestel. Andere toestellen kunnen dan die gegevens opvragen en de SNMP-agent zal die aanvraag beantwoorden.
- De SNMP-manager, vaak een Network Management Systeem (NMS) genoemd, is de software die de SNMP-requests genereert, verstuurt naar de SNMP-agents en uiteindelijk de antwoorden verwerkt. Een NMS zal meestal niet één, maar meerdere SNMP-agents van verschillende toestellen ondervragen. Aan de hand van de verkregen informatie kan de NMS beslissen om verdere acties uit te voeren.

2.2.3 Uniformiteit

Een van de grootste voordelen van SNMP is het feit dat ze je een uniforme manier aanbiedt om je netwerk te beheren. Fabrikanten van netwerkapparatuur bieden graag hun eigen oplossing aan om hun toestellen te beheren, maar het probleem is dat netwerken zelden uit apparatuur bestaan van slechts één fabrikant. De managementoplossingen van de ene fabrikant werken meestal niet om ook de toestellen van de andere fabrikant te beheren. Met SNMP zit je niet vast aan een fabrikant en kun je op uniforme wijze gans je netwerk beheren.

SNMP biedt maar weinig functionaliteit aan en dat speelt zowel in zijn voordeel als in zijn nadeel: Het voordeel is het gemak waarmee fabrikanten SNMP kunnen implementeren, het nadeel is het gebrek aan features: met SNMP kun je enkel ruwe data opvragen. Als je meer uit je data wenst te halen, moet je die zelf achteraf verder verwerken. Denk aan het bijhouden van historische data, het combineren van verschillende gegevens om verbanden te zien, grafieken opmaken of zelfs ganse rapporten opstellen. Dergelijke zaken worden wel ondersteund door de managementoplossingen van fabrikanten, maar bij SNMP is het aan de gebruiker om die data uit de SNMP-gegevens te extraheren.

2.2.4 Onderliggende protocols

SNMP steunt op IP en UDP als transportprotocol. Dankzij de IP-laag kan SNMP ook werken op heterogene netwerken. UDP werd verkozen boven TCP vanwege de eenvoudige werking, wat

weer handig is voor low-level netwerkcomponenten. Bovendien heeft UDP een kleinere impact op het netwerkverkeer dan TCP[17]. Het nadeel van UDP is wel dat er geen bevestiging gebeurt van verstuurde pakketten. Als er een pakket verloren gaat, is het dan aan de NMS om dit te detecteren en op te vangen. Normaal zal een NMS simpelweg de SNMP-query opnieuw versturen. Voor het versturen en ontvangen van SNMP-berichten wordt er standaard gebruik gemaakt van poort 161.

2.3 Object Identifiers

Elk mogelijk soort gegeven dat je kunt opvragen via SNMP van een toestel wordt wereldwijd uniek geïdentificeerd[17] door een Object Identifier (OID). OID's worden hiërarchisch ingedeeld in een boomstructuur, vergelijkbaar met DNS. De eindpunten van de boom stellen objecten voor en de knooppunten van de boom worden gebruikt om objecten logisch te groeperen. De Relative Identifiers (RID's) van de knooppunten bepalen de uiteindelijke OID. OID's hebben een dotted-decimal notatie waarbij de knooppunten en het object van elkaar gescheiden worden door punten. Van links naar rechts wordt de OID opgebouwd door het hoogste knooppunt tot het uiteindelijk object. Een voorbeeld van een OID die de uptime van een syteem teruggeeft is *iso.org.dod.internet.mgmt.mib-2.system.sysUpTime*. Het stuk van de boomstructuur waarin die OID valt kun je ook zien in figuur 2.1.

De tekstuele notatie van een OID valt, zoals je ziet, nogal lang uit. Daarom is het ook mogelijk om van een numerieke notatie gebruik te maken. Elk knooppunt heeft behalve de tekstuele naam ook een nummer waar gebruik van kan gemaakt worden, en een veel kortere OID oplevert. De OID die hiervoor als voorbeeld werd gegeven wordt dan 1.3.6.1.2.1.1.3. In de figuur van de boomvoorstelling staat het nummer van elk knooppunt tussen ronde haken. Eventueel kan er ook gebruik gemaakt worden van een hybride notatie waarbij afwisselend gebruik kan gemaakt worden van de tekstuele of numerieke identificatie van een knooppunt. Een mogelijke hybride voorstelling van het vorig voorbeeld is 1.3.6.1.mgmt.1.1.sysUpTime.

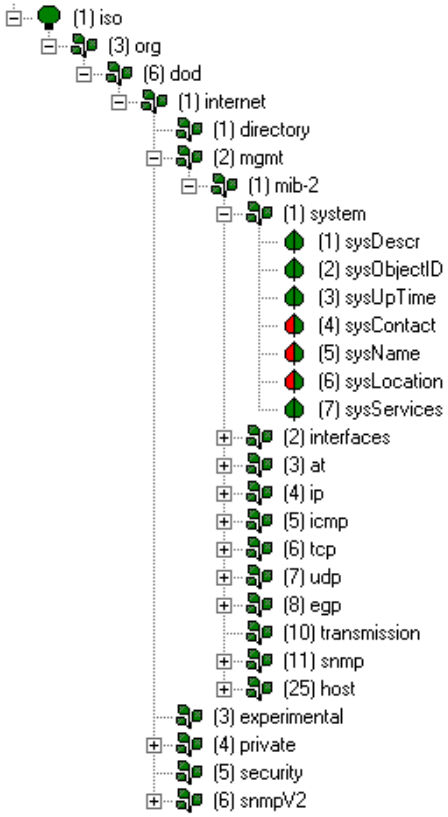
2.4 Management Information Base

NMS'en houden een overzicht bij van alle gegevens die ze kunnen opvragen van een SNMP-agent in een zogeheten Manageable Objects Database (MDB). In die MDB zit een verzameling van Management Information Base (MIB) bestanden.[1] In die MIB-bestanden worden de eigenlijke SNMP-objecten gedefiniëerd. Vaak gaat men samenhorende objecten in hetzelfde MIB-bestand definiëren. Bijvoorbeeld alle objecten die bij een bepaald protocol horen, of alle objecten die worden geïmplementeerd door een bepaalde fabrikant. Voor elk object wordt de naam en het nummer vastgelegd die zullen gebruikt worden in het OID voor dat object, alsook een beschrijving ervan en welke datastructuur er gebruikt wordt om de data voor te stellen (bv. een string of integer).

Als voorbeeld kun je de definitie van hetzelfde *sysUpTime*-object als eerder zien in codefragment 2.1. Het *ACCESS*-attribuut duidt hierbij aan of een gegeven gewijzigd mag worden of niet. *STATUS* geeft aan of het gegeven verplicht geïmplementeerd moet worden. Onderaan zie je ook dat het object onder de *system*-tak valt.

Voor NMS'en zijn MIB-bestanden zeer belangrijk. Zij moeten weten welke gegevens ze precies kunnen opvragen van een bepaalde agent, en vooral: hoe ze die gegevens moeten interpreteren! Een NMS moet dus minstens over de MIB's beschikken die de OID's definiëren die hij wenst op te vragen. Een groot aantal MIB's zijn gestandaardiseerd en worden ook standaard meegeleverd met SNMP-software. Een aantal van die gestandaardiseerde MIB's moeten ook verplicht ondersteund

[1] De MIB-bestanden worden eerst gecompileerd naar een binair formaat alvorens ze in de MDB opgeslagen worden.[17]



Figuur 2.1: Boomstructuur van SNMP-objecten

worden door SNMP-agents om de stempel 'SNMP compatibel' te mogen dragen. Het MIB-2 knooppunt dat je kunt zien in figuur 2.1 is daar een voorbeeld van. Die minimale set van gegevens zal je dus van ieder SNMP toestel kunnen opvragen.

2.5 Tabellen

Behalve scalaire waarden is het ook mogelijk om tabellen op te vragen met SNMP. We maken gebruik van de standaardtabel voor netwerkinterfaces *ifTable* met als OID 1.3.6.1.2.1.2.2 ter illustratie. Een tabel wordt als volgt gedefiniëerd in een MIB-bestand: je hebt enerzijds een tabelobject (hier ifTable) en anderzijds een rijobject (hier ifEntry). De definitie van het tabelobject zie je in codefragment 2.2. De volgende codefragmenten tonen de definities van de overige objecten.

Het tabelobject wordt gedefiniëerd als een sequentie of opeenvolging van rijobjecten. Het rijobject wordt op zijn beurt gedefiniëerd als een sequentie van kolommen. Hiervoor wordt een aparte sequentiedefinitie (IfEntry) gebruikt die bestaat uit een opeenvolging van kolomnamen gevolgd door hun datatype.

Bij de definitie van het rijobject zelf zie je dat er als syntax (datatype) de voorgaande sequentiedefinitie wordt gebruikt. Merk op dat de sequentiedefinitie (IfEntry) conventioneel (maar niet verplict) begint met een hoofdletter maar de definitie van het object zelf niet (ifEntry).

Ook belangrijk is het *INDEX*-attribuut. Deze geeft aan hoe de tabel geïndexeerd moet worden, dus hoe je aan individuele rijen kunt geraken. Dit kan een eenvoudige integer zijn of een samenstelling van kolommen die een rij uniek kan identificeren. Hier wordt de index (ifIndex) gedefiniëerd als een eenvoudige integer.

```
sysUpTime OBJECT-TYPE
     SYNTAX TimeTicks
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION "The time (in hundredths of a second) since the
                  network management portion of the system was last
                  re-initialized."
    ::= { system 3  }
```

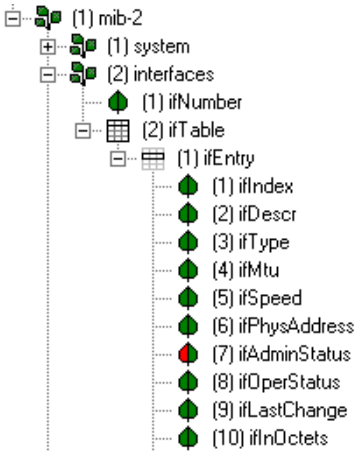
Codefragment 2.1: Definitie van een SNMP-object

```
ifTable OBJECT-TYPE
    SYNTAX  SEQUENCE OF IfEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
            "A list of interface entries.  The number of
            entries is given by the value of ifNumber."
    ::= { interfaces 2 }
```

Codefragment 2.2: Definitie van een tabel

De indexering gebeurt dan als volgt. Om te beginnen heb je de OID van de tabel zelf: 1.3.6.1.2.1.2.2. Het rijobject heeft ook een eigen OID. Zijn RID is 1 en komt achter de OID van de tabel. Daarna komt de RID van de kolom. Laten we als voorbeeld de kolom met de snelheid van de interface nemen (*ifSpeed*). Deze heeft als RID 5. De OID voor die kolom wordt dan 1.3.6.1.2.1.2.2.1.5. Als we alle OID's overlopen die beginnen met de OID van de ifSpeed-kolom, dan krijgen we de waarden van alle rijen voor die ene kolom. Als we ook nog eens een specifieke rij willen opgeven, dan volgt dat na de kolomaanduiding. Vermits er hier gebruik gemaakt wordt van een simpele integer als index kunnen we achteraan de RID 1 toevoegen om de snelheid van de *eerste* interface te weten te komen. De uiteindelijke OID wordt dan 1.3.6.1.2.1.2.2.1.5.1. Je kunt dit visueel ook bevestigen in figuren 2.2 en 2.3.

Omdat de kolomaanduiding voor de rijaanduiding komt in een OID, zal het overlopen van alle OID's die beginnen met de OID van de tabel tot resultaat hebben dat de tabel kolom per kolom wordt overlopen. Deze operatie wordt een SNMP walk van een OID genoemd en wordt verder uitgelegd in paragrafen 2.6.2 en 2.7.2.



Figuur 2.2: Boomstructuur van een tabel

```
IfEntry ::=
    SEQUENCE {
        ifIndex
            INTEGER,
        ifDescr
            DisplayString,
        ifType
            INTEGER,
        ...
    }
```

Codefragment 2.3: Sequentiedefinitie van een tabelrij

```
ifEntry OBJECT-TYPE
    SYNTAX  IfEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
            "An interface entry containing objects at the
            subnetwork layer and below for a particular
            interface."
    INDEX   { ifIndex }
    ::= { ifTable 1 }
```

Codefragment 2.4: Definitie van een rijobject

2.6 SNMP Operaties

SNMP maakt gebruik van het Protocol Data Unit (PDU)-berichtformaat voor al het communicatieverkeer tussen SNMP-agents en NMS'en. Er worden een aantal verschillende operaties ondersteund door SNMP en die hebben elk hun eigen PDU-formaat[12]. Hieronder worden enkel de belangrijkste SNMP-operaties gegeven die relevant zijn voor de masterproef:

- GET
- GETNEXT
- GETBULK

2.6.1 GET

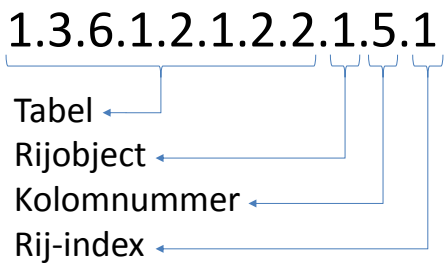
De GET-operatie was de eerste en belangrijkste SNMP operatie. De GET-operatie wordt geïnitieerd door de NMS en laat die toe om een gegeven van een SNMP-agent op te vragen. Om te weten welk gegeven de NMS juist wenst te weten te komen, geeft die de OID mee met de PDU die het gegeven uniek identificeert. Herinner je dat de NMS over een MIB moet beschikken die de OID definiëert, zodat zowel de NMS als SNMP-agent perfect weten waarover het gaat. De SNMP-agent ontvangt de PDU en verwerkt deze. Een nieuwe GET-response-PDU wordt opgesteld voor dat OID, met de waarde ervan ingevuld, en teruggestuurd door de SNMP-agent. De NMS ontvangt de GET-response-PDU en weet nu de waarde voor dat OID.

2.6.2 GETNEXT

De GETNEXT-operatie wordt gebruikt om een verzameling van opeenvolgende OID's op te vragen. Gegeven een bepaalde OID zal de GETNEXT-operatie de eerstvolgende OID met bijhorende waarde teruggeven. Dit gebeurt in lexicografische volgorde en vermits OID's samengesteld zijn door

```
ifIndex OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "A unique value for each interface.  Its value
            ranges between 1 and the value of ifNumber.  The
            value for each interface must remain constant at
            least from one re-initialization of the entity's
            network management system to the next re-
            initialization."
    ::= { ifEntry 1 }
```

Codefragment 2.5: Definitie van een index



Figuur 2.3: OID van een cel uit een SNMP-tabel

integers, kan zo makkelijk een ganse boomtak overlopen worden. Deze manier van werken noemt men diepte-eerst zoeken.[12] Wanneer de NMS het antwoord ontvangt van een GETNEXT-operatie, zal ze een nieuwe sturen voor het volgende OID. De NMS zal blijven GETNEXT-PDU's sturen tot de agent een foutmelding terugstuurt die aangeeft dat het einde van de MIB bereikt is.

De GETNEXT-operatie is vooral handig bij het overlopen van tabellen omdat je geen rekening hoeft te houden met de indexering van de rijen. Tabellen kunnen een index bevatten die samengesteld wordt door meerdere attributen, en zelfs als een tabel numeriek geïndexeerd is, zijn de indexen niet noodzakelijk opeenvolgend. Met de GETNEXT-operatie hoef je je daar geen zorgen over te maken, ze geeft je meteen de volgende index met de bijhorende gegevens. De GETNEXT-operatie uitgevoerd op de OID van een tabel geeft je zijn eerste index, waarmee je de volgende indexen kunt opvragen.

Normaal gezien zal je niet rechtstreeks in contact komen met GETNEXT-operaties, maar zul je gebruik maken van de SNMP walkopdracht. Je hoeft enkel de OID mee te geven met de SNMP walkopdracht waarop deze de nodige GETNEXT-requests zal sturen om de ganse boomtak te overlopen.

2.6.3 GETBULK

Met de tweede versie van SNMP werd de GETBULK-operatie gespecificeerd. Deze operatie laat je toe om in een request meerdere OID's op te vragen. Bij een gewone GET- of GETNEXT-operatie kun je ook meer dan een OID meegeven, maar de berichtgrootte wordt beperkt door de capaciteit van de SNMP-agent. Als een SNMP-agent geen antwoord kan geven op alle OID's die werden gevraagd in de GET-operatie, wordt een foutboodschap teruggestuurd, zonder data. De GETBULK-operatie daarentegen probeert zo veel mogelijk data als het kan terug te sturen. Met deze operatie is het dus wel mogelijk om incomplete antwoorden terug te krijgen.[12]

De belangrijkste feature van de GETBULK-operatie is echter de mogelijkheid om het lexicografisch

overlopen van gegevens op de SNMP-agent te laten gebeuren. Daartoe geef je aan hoeveel GETNEXT-requests er moeten gebeuren en voor welke OID's. Doordat het werk op de SNMP-agent gebeurt, moet er niet gewacht worden op elk individueel antwoord en kunnen de gegevens samengebundeld worden in één antwoord. In paragraaf 2.7.2 zullen we hier verder op ingaan.

2.7 Manieren om SNMP gegevens op te vragen

Aan de hand van de SNMP-operaties besproken in paragraaf 2.6 zijn er verschillende mogelijkheden om gegevens op te vragen, al naar gelang je een of meerdere gegevens wenst op te vragen, en welke gegevens je juist wenst op te vragen (bv. tabellen). Met behulp van Net-SNMP zullen de verschillende mogelijkheden hieronder gedemonstreerd worden. Net-SNMP is een gratis en open-source softwarepakket dat commando's aanbiedt voor de verschillende SNMP-operaties, en is beschikbaar voor de meeste besturingssystemen waaronder Windows en Linux. Wij maken gebruik van de Linux versie.

2.7.1 Opvragen van één enkel gegeven

GET-operatie

Indien je maar geïnteresseerd bent in slechts één gegeven, dan ligt het voor de hand om gebruik te maken van de SNMP GET-operatie. Voorwaarde is wel dat je exact de OID kent van het gegeven dat je wenst op te vragen. Een GET-request in Net-SNMP ziet er zo uit:

```
$ snmpget -v 1 -c public 127.0.0.1 sysDescr.0
SNMPv2-MIB::sysDescr.0 = STRING: Linux debian-vm-01 3.2.0-4-amd64 #1 SMP Debian 3.2.54-2 x86_64
```

Codefragment 2.6: SNMP GET-opdracht

Daarbij geeft de -v optie de te gebruiken versie mee en de -c optie de *communitystring*. De communitystring fungeert als een soort wachtwoord om SNMP-requests te beveiligen. De opties worden gevolgd door het IP adres van het te ondervragen toestel en het OID van het gegeven dat je wenst op te vragen. Zoals je misschien al opgemerkt hebt, wordt de OID hier niet voorafgegaan door iso.org.dod.internet.mgmt.mib-2.system, zoals het zou horen voor een OID die in die subtak valt. De reden daarvoor is dat dit niet vereist is als de naam van de tak uniek is, wat het geval is voor sysDescr.

De volgende vraag die je wellicht stelt is waarom er nog een .0 achter de OID staat. Dit is omdat MIB objecten geïdentificeerd worden door de conventie x.y waarbij x de OID is van het object en y de instantie aanduidt. Normaal wordt y gebruikt bij tabellen om de rij aan te duiden (1 is de eerste rij, 2 de tweede, enzovoort), zoals uitgelegd in paragraaf 2.5. Maar bij scalaire objecten is y altijd 0. De .0 achterwege laten levert een fout op.[12]

GETNEXT-operatie

Je kan ook gebruik maken van de SNMP GETNEXT-operatie als je het *volgende* gegeven wenst te weten te komen. Dit wordt hoofdzakelijk gebruikt bij het overlopen van tabellen zonder dat je moet rekening houden met de indexering van de tabel. Het is mogelijk om zelf een SNMP GETNEXT-operatie uit te voeren, maar normaliter maak je gebruik van de SNMP walkopdracht die hiervan gebruik maakt om een ganse boomtak te overlopen. Hier gaan we verder op in in paragraaf 2.6.2.

```
$ snmpgetnext -v 1 -c public 127.0.0.1 ifTable
IF-MIB::ifIndex.1 = INTEGER: 1
```

Codefragment 2.7: SNMP GETNEXT-opdracht op een tabel

Als je de indexering van een tabel niet kent, dan zou je het volgende kunnen doen:

Dit levert dan de waarde van de eerste kolom (hier ifIndex) van de eerste rij op. Herinner je dat de OID van een tabelelement eerst het kolomnummer bevat en dan de index van de rij. Dus als je de waarde van een andere kolom wenst zonder dat je de index van de eerste rij kent, zou je het volgende kunnen doen:

```
$ snmpgetnext -v 1 -c public 127.0.0.1 ifTable.1.2
IF-MIB::ifDescr.1 = STRING: lo
```

Codefragment 2.8: SNMP GETNEXT-opdracht op een kolom van een tabel

Dit levert dan de tweede kolom (de beschrijving van de interface, hier lo, kort voor loopback interface) op van de eerste rij. Maar zoals gezegd zul je eerder gebruik maken van de SNMP walkopdracht dan van GETNEXT-operaties (zie verder).

2.7.2 Opvragen van meerdere gegevens

SNMP BULK-operatie

Als je meerdere gegevens wenst op te vragen gebruik je best de SNMP BULK-operatie. Deze laat toe om meerdere OID's op te vragen en opeenvolgende GETNEXT-operaties op een SNMP-agent te laten gebeuren, om dan de gegevens samen te bundelen in één antwoordbericht.

Om van SNMP BULK-operaties gebruik te maken moet je ten eerste zorgen dat je gebruik maakt van SNMP versie 2c (zie ook paragraaf 2.8 waarin we de verschillende SNMP versies bespreken). Ten tweede moet je ook twee extra parameters opgeven: *non-repeaters* en *max-repetitions*. Je kunt meerdere OID's meegeven met de BULK-operatie en met die parameters kun je opgeven of er slechts één keer een GETNEXT-operatie op een OID moet uitgevoerd worden of meerdere. Non-repeaters geeft aan hoeveel OID's er zijn meegegeven waarop slechts één keer een GETNEXT-operatie moet uitgevoerd worden. Deze OID's zullen dus elk slechts één antwoord bevatten in het antwoordbericht. Met max-repetitions geef je aan hoeveel keer er een GETNEXT-operatie moet uitgevoerd worden op de overige OID's. Als je max-repetitions op 10 zet, dan zal je voor elk van de overige OID's 10 antwoorden krijgen in het antwoordbericht. Vermits je enkel opgeeft hoeveel OID's niet herhalen (non-repeaters) moet je eerste de niet-herhalende OID's opgeven en dan de herhalende. Wellicht dat een voorbeeld meer duidelijkheid verschaft:

```
$ snmpbulkget -v 2c -c public -Cn1 -Cr3 127.0.0.1 sysDescr ifTable ipAddrTable
SNMPv2-MIB::sysDescr.0 = STRING: Linux debian-vm-01 3.2.0-4-amd64 #1 SMP Debian 3.2.54-2 x86_64
IF-MIB::ifIndex.1 = INTEGER: 1
IP-MIB::ipAdEntAddr.10.0.2.11 = IpAddress: 10.0.2.11
IF-MIB::ifIndex.2 = INTEGER: 2
IP-MIB::ipAdEntAddr.127.0.0.1 = IpAddress: 127.0.0.1
IF-MIB::ifIndex.3 = INTEGER: 3
IP-MIB::ipAdEntIfIndex.10.0.2.11 = INTEGER: 2
```

Codefragment 2.9: SNMP BULK-opdracht

Het aantal non-repeaters en het aantal max-repetitions worden ingesteld met respectievelijk de -Cn en de -Cr optie. Vermits het aantal non-repeaters op 1 staat wil dit zeggen dat enkel de eerste OID, sysDescr, niet herhaald moet worden. En omdat max-repetitions op 3 staat, worden alle overige OID's drie keer herhaald (hier ifTable en ipAddrTable). Als antwoord zie je het volgende staan: de waarde van sysDescr, de eerste drie waarden van ifTable en de eerste drie waarden van ipAddrTable.

Merk op dat de versie nu op 2c staat. Verder zie je dat de OID voor de scalaire waarde sysDescr nu niet gevolgd wordt door .0, want achterliggend maakt de BULK-operatie gebruik van een GETNEXT-operatie. Als je toch .0 toevoegt dan zul je niet de waarde van sysDescr terugkrijgen maar die van de volgende OID. Moest je ten slotte een hoger aantal max-repetitions gebruiken dan een tabel cellen heeft, of per ongeluk een scalaire waarde meerdere keren laten herhalen, dan krijg je ook nog de waarden van de OID's die volgen op de tabel of de scalaire waarde terug.

GET- en GETNEXT-opdrachten

In principe is het ook mogelijk om meerdere gegevens op te vragen met een GET- of GETNEXT-opdracht. Om dat te doen geef je simpelweg meerdere OID's mee in de opdracht:

```
$ snmpget -v 1 -c public 127.0.0.1 sysDescr.0 sysUpTime.0 sysName.0
SNMPv2-MIB::sysDescr.0 = STRING: Linux debian-vm-01 3.2.0-4-amd64 #1 SMP Debian 3.2.54-2 x86_64
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (1451759) 4:01:57.59
SNMPv2-MIB::sysName.0 = STRING: debian-vm-01
```

Codefragment 2.10: Meerdere gegevens opvragen met SNMP GET

Het nadeel van dit te doen met een GET- of GETNEXT-opdracht is dat, als er een fout optreedt bij het opvragen van een van de OID's, je geen gegevens terug zal krijgen over de OID's die wel correct waren:

```
$ snmpget -v 1 -c public 127.0.0.1 sysDescr.0 sysUpTime.0 sysName.0 fouteOID
fouteOID: Unknown Object Identifier (Sub-id not found: (top) -> fouteOID)
```

Codefragment 2.11: Meerdere gegevens opvragen met SNMP GET met een foute OID

Bij een BULK-opdracht krijg je die gegevens wel. Om meerdere gegevens op te vragen maak je dus beter gebruik van BULK-opdrachten.

SNMP walk

Normaal gezien maak je weinig gebruik van de GETNEXT-operatie maar gebruik je in de plaats de SNMP walkopdracht. De SNMP walkopdracht wordt gebruikt om een ganse boomtak te overlopen en maakt daarvoor gebruik van GETNEXT-operaties. Met de volgende SNMP walkopdracht kun je de interfacetabel van een toestel overlopen:

```
$ snmpwalk -v 1 -c public 127.0.0.1 ifTable
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifIndex.2 = INTEGER: 2
IF-MIB::ifIndex.3 = INTEGER: 3
...
IF-MIB::ifSpecific.4 = OID: SNMPv2-SMI::zeroDotZero
IF-MIB::ifSpecific.5 = OID: SNMPv2-SMI::zeroDotZero
IF-MIB::ifSpecific.6 = OID: SNMPv2-SMI::zeroDotZero
```

Codefragment 2.12: SNMP walkopdracht

Merk op dat we hier geen instantie (de .0 bij scalaire gegevens) na de OID hebben opgegeven. Mocht je hier toch een instantie opgeven (de index van een rij van de tabel) dan zou je slechts één kolomwaarde terugkrijgen, horende bij die rij en had je hetzelfde kunnen bereiken met een SNMP GET-opdracht.

Omdat je GETNEXT-operaties op een SNMP-agent kunt laten uitvoeren met BULK-operaties, kun je de SNMP walkopdracht nog sneller laten verlopen door de BULK-variant te gebruiken. Het resultaat blijft natuurlijk hetzelfde.

```
$ snmpbulkwalk -v 2c -c public 127.0.0.1 ifTable
...
```

Codefragment 2.13: SNMP walkopdracht m.b.v. BULK-operaties

Let erop dat je de versie moet veranderen omdat je gebruik maakt van BULK-operaties.

Net als bij het snmpbulkget-commando kun je hier het aantal herhalingen (max-repetitions) opgeven met de -Cr optie. Daarbij komt het erop neer dat we kunnen opgeven hoeveel objecten moeten samengebundeld worden in een pakket. Om een SNMP walk zo snel mogelijk te laten verlopen zijn we geneigd om dit aantal zo hoog mogelijk in te stellen. We moeten echter wel rekening houden met een aantal zaken. Ten eerste is de maximale grootte van een pakket beperkt tot de Maximum Transmission Unit (MTU) van een verbinding. Bij een ethernetverbinding gaat het om 1500 bytes. Als een bericht van meer dan 1500 bytes over ethernet verstuurd moet worden, dan moet deze opgesplitst worden over meerdere berichten, wat de performantie nadelig kan beïnvloeden.

Ten tweede is het bij een groot aantal herhalingen ook mogelijk dat er redelijk wat data verspild wordt. Stel dat we het aantal herhalingen instellen op 50, dan zullen de pakketten steeds 50 objecten bevatten. Ook het laatste pakket zal er 50 bevatten, ook al valt er slechts een object meer binnen de walk. Dan zijn de overige 49 objecten natuurlijk niet relevant voor ons en doen we daar dan ook niks mee.

Als we een walk doen van een deelboom die maar 10 objecten heeft, heeft het ook geen zin om het aantal herhalingen op 50 te zetten. Natuurlijk weet je op voorhand niet hoeveel objecten juist onder een OID vallen, maar je kan meestal wel een schatting doen of dit baseren op het aantal objecten dat in het verleden werd opgehaald voor dat OID van een bepaald toestel.

Het aantal herhalingen is uiteindelijk een beslissing die vooral zal afhangen van het geschatte aantal gegevens dat opgevraagd zal worden. Standaard staat het aantal herhalingen daarom op een conservatieve 10 objecten.

2.8 Versies

Er zijn drie grote versies van SNMP die momenteel in gebruik zijn op netwerktoestellen: SNMPv1, SNMPv2c en SNMPv3. Ondanks dat SNMPv3 de twee voorgaande versies opvolgt kun je nog steeds veel toestellen vinden die enkel met SNMPv2c of zelfs met SNMPv1 werken. Het is aan de fabrikant om voor de ondersteuning van SNMPv3 te zorgen op een toestel.

2.8.1 SNMPv1

De eerste versie van SNMP dateert al van 1988 maar is soms toch nog te vinden als enige ondersteunde versie op een toestel. Een van de grootste gebreken die deze versie kenmerkt is de beveiliging. De originele versie van SNMP maakte gebruik van een zogeheten *community string* om SNMP-requests te beveiligen. Die string fungeert als een wachtwoord die in *cleartext* met ieder SNMP-request wordt meegegeven[20]. Natuurlijk kan iedereen die de requests kan onderscheppen het wachtwoord gewoon uitlezen en zelf requests met dat wachtwoord versturen. SNMP voorziet in

de mogelijkheid om niet alleen data uit te lezen, maar ook om data aan te passen met behulp van de SNMP SET-operatie. Vanwege de slechte beveiliging wordt de SET-operatie echter zo goed als nooit geïmplementeerd in de SNMP-agent zodat de operatie geen effect heeft.

2.8.2 SNMPv2c

De tweede versie van SNMP introduceerde in 1993 [19] oorspronkelijk de GETBULK-operatie en een betere beveiliging. De nieuwe beveiliging werd echter als te complex beschouwd en werd op veel plaatsen niet aanvaard. SNMPv2c werd in 1996 [19] voorgesteld als een alternatief die de verbeteringen van SNMPv2 had, maar de complexe beveiliging ervan achterwege liet ten voordele van de community strings van SNMPv1. Deze versie werd wel aanvaard door de gemeenschap en is nog steeds in gebruik op een groot aantal toestellen. De GETBULK-operatie was een welkome toevoeging aan SNMP omdat ze een veel performanter alternatief bood voor de vele GETNEXT-operaties die voorheen nodig waren om grote hoeveelheden gegevens op te vragen.

2.8.3 SNMPv3

De derde en laatste versie van SNMP werd geaccepteerd als een volwaardige internetstandaard in 2002 [22] en bracht de reeds lang gevraagde verbetering in beveiliging met zich mee. Omdat de vorige versies van SNMP slecht beveiligd waren, werd het protocol enkel gebruikt voor het monitoren van het netwerk en performantiebeheer. Met de verbeterde beveilging in SNMPv3 kan SNMP eindelijk een veilig platform aanbieden om een netwerk niet enkel passief te beheren zoals voorheen, maar ook actief te beheren door configuratiewijzigingen uit te voeren via SNMP. SNMPv3 is echter op veel toestellen nog steeds niet aanwezig en zeker de SNMP SET-operatie wordt in de praktijk maar zelden geïmplementeerd.

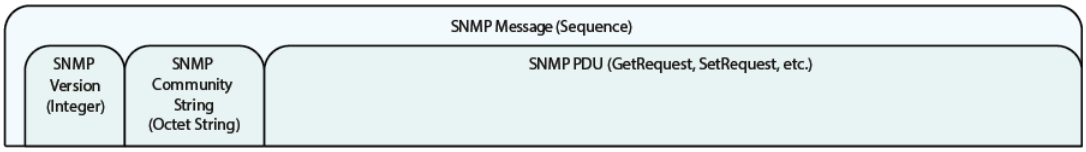
2.9 Berichtstructuur van SNMP

In deze paragraaf bespreken we de berichtstructuur van een SNMP-bericht. In tegenstelling tot bijvoorbeeld IP- en ethernetheaders, hebben de headers van SNMP-berichten geen vaste lengte. In de plaats daarvan bestaat een SNMP-bericht uit Tag-Length-Value (TLV) tripletten[17]. Daarbij geeft de tag aan om wat voor datatype het gaat, length duidt de lengte aan van de data en value bevat de data zelf. Je hebt primitieve datatypes zoals een integer, een octetstring of een OID. Maar je hebt ook complexe datatypes zoals een sequentie of een PDU voor bijvoorbeeld een GET-request of een GET-response. De complexe types zoals de sequentie zijn opgebouwd uit meerdere kleinere velden zodat je een geneste structuur krijgt[2].

Primitieve datatypes	Complexe datatypes
Integer	Sequence
Octet String	GetRequest
Object Identifier	GetResponse
Null	GetBulkRequest

Tabel 2.1: Enkele SNMP-datatypes

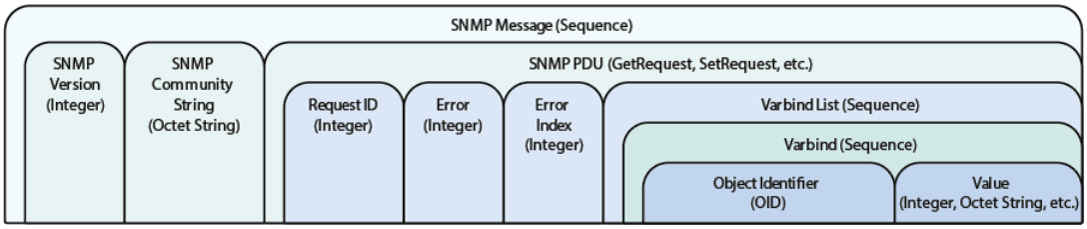
Een SNMP-bericht is dus niks meer dan een geneste structuur van datavelden. Het SNMP-bericht zelf wordt gedefinieerd als een sequentie van drie velden: een integer die de SNMP versie voorstelt, een octetstring die de community voorstelt en de eigenlijke SNMP-PDU (zelf een samengesteld type).



Figuur 2.4: Berichtstructuur van een SNMP-bericht[2]

De SNMP-PDU bestaat uit de volgende velden:

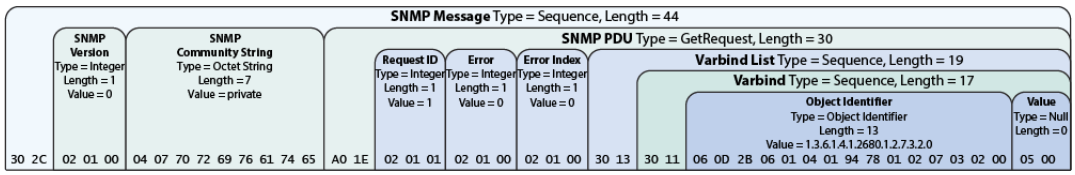
- Request ID (integer): een unieke identificatie van een SNMP-request. Het responsbericht zal dezelfde identificatie gebruiken zodat de ontvanger weet bij welke request het antwoord hoort.
- Error (integer): duidt aan of er een fout is opgetreden. De SNMP-agent zal deze waarde veranderen indien nodig. Als de waarde op nul staat is er geen fout opgetreden.
- Error Index (integer): verwijst naar het object dat de fout heeft veroorzaakt.
- Varbind List (sequentie): de verzameling van alle objecten in de PDU.
  - Varbind (sequentie): komt overeen met een object en bevat zijn OID en waarde. De waarde wordt enkel ingevuld in het responsbericht.
    - Object Identifier (OID)
    - Value (integer, octetstring, . . .)



Figuur 2.5: Berichtstructuur van een SNMP-bericht en zijn SNMP-PDU[2]

Een voorbeeld van een GET-request in detail zie je in figuur 2.6. Onderaan zie je de hexadecimale waarden van de bytes. De eerste byte van een veld duidt steeds de code van het datatype aan en de tweede de lengte van het veld. De volgende bytes vormen dan de inhoud van het veld. Zodoende krijgen we de Tag-Length-Value tripletten waar we over spraken. Merk op dat als de lengte van een veld, dat deel uitmaakt van een complex datatype zoals een sequentie, verandert, dan moet de lengte van het bovenliggende datatype ook aangepast worden.

Omdat het hier om een request gaat, is de waarde van het object nog niet ingevuld. Daarmee zien we meteen ook het doel van het Null-datatype: zo kan er aangegeven worden dat een veld niet is ingevuld en kan er een byte uitgespaard worden. De value van het TLV-triplet is dan ook niet aanwezig en zijn lengte is nul.



Figuur 2.6: Berichtstructuur van een SNMP-bericht in detail[2]

Hoofdstuk 3

Bestaande situatie

3.1 SNMP Data Retriever

De SNMP Data Retriever is het stuk software dat NetworkMining zelf ontwikkeld heeft voor de bevraging van netwerkcomponenten via SNMP. Alhoewel het de bedoeling is om netwerkcomponenten te ondervragen over bijvoorbeeld hun routetabel of ARP-tabel, is het geen probleem om ook alle andere soorten toestellen te ondervragen, bijvoorbeeld werkstations van werknemers of telefoontoestellen die verbonden zijn met het netwerk. De retriever laat je toe om een lijst van op te vragen toestellen op te geven. Om te weten welke gegevens moeten opgevraagd worden bij elk toestel, moet aan elk toestel een toesteltype toegewezen worden. De toesteltypes maak je zelf aan en bevatten een lijst van de op te vragen gegevens voor dat type. Hieronder geven we een overzicht van de verschillende configuratiemogelijkheden die de SNMP Data Retriever aanbiedt.

We bespreken ook de databankstructuur die door de SNMP Data Retriever gebruikt wordt om de resultaten weg te schrijven en geven ook een globaal overzicht van hoe de software te werk gaat.

Bij de aanvang van de masterproef was het zo dat de retriever reeds ingezet werd voor kleine en middelgrote netwerken. De bedoeling van de masterproef is om de nodige aanpassingen te doen zodat de retriever ook vlot kan ingezet worden voor grote netwerken, van 1000 toestellen en meer.

3.1.1 Configuratie

Er zijn drie manieren waarop de retriever kan geconfigureerd worden. De belangrijkste, en wellicht de enige waar de eindgebruiker mee in aanraking zal komen is het XML-configuratiebestand. Een aantal opties kunnen ook doorgegeven worden aan de hand van argumenten bij het oproepen van het programma. De laatste manier is via nog een configuratiebestand: het *AppConfig*-bestand. De laatste twee mogelijkheden zullen hoogstwaarschijnlijk eenmalig geconfigureerd worden bij de installatie van de software en verder nooit meer gewijzigd worden.

XML-configuratiebestand

De twee belangrijkste dingen die je kunt configureren in het XML-configuratiebestand, zijn wat voor types toestellen er zijn en de lijst van IP-adressen van die toestellen. Een voorbeeld van een definitie van een toesteltype zie je in codefragment 3.1. Het toesteltype heet *General* en als er een toestel van dat type bevraagd wordt, moet er een SNMP walk gedaan worden van de

system boomtak met als OID 1.3.6.1.2.1.1. Dit is dezelfde boomtak die we eerder hebben gezien in figuur 2.1. De system boomtak is verplicht aanwezig op alle SNMP-toestellen, vandaar de naam van het toesteltype. Het MIB-bestand waarin die tak gedefiniëerd is heet *RFC1213-MIB*. Deze gegevens vind je terug als de attributen van de SNMP walk-opdracht in het configuratiebestand. Een toesteltype moet niet beperkt zijn tot een enkele SNMP walk. Ze kan ook bestaan uit meerdere SNMP walk-opdrachten, of zelfs een SNMP GET-opdracht voor een enkele OID.

```
<deviceType name="General">
    <snmpWalk oid="1.3.6.1.2.1.1" mib="RFC1213-MIB" name="system" />
</deviceType>
```

Codefragment 3.1: Definitie van een toesteltype in het XML-configuratiebestand

Nadat de toesteltypes gedefiniëerd zijn, kun je de IP-adressen opgeven van alle toestellen die opgevraagd moeten worden. Bij de opsomming van de toestellen hoort natuurlijk ook hun toesteltype die we eerder gedefiniëerd hebben. In codefragment 3.2 zie je dat de definitie van een toestel bestaat uit drie attributen: een arbitrair gekozen naam, zijn IP-adres of hostnaam en zijn toesteltype.

```
<device name="atlas2a1.intec.ugent.be" ip="atlas2a1.intec.ugent.be" type="Bridge" />
```

Codefragment 3.2: Definitie van een toestel in het XML-configuratiebestand

De overige opties zijn die voor een databaseconnectiestring, de communitystring, de locatie van de MIB-bestanden en de SNMP-timeout waarde. Dit is hoelang gewacht wordt (in milliseconden) op een response na het versturen van een SNMP-request.

```
<database value="Database=snmpdb;Data␣Source=localhost;User␣
    Id=networkminer;Password=SomePassword;Port=3306;old␣syntax=yes" />
<snmpCommunity get="public" />
<MIBpath value=".\MIBs" />
<snmpTimeout value="3000" />
```

Codefragment 3.3: Overige opties in het XML-configuratiebestand

Argumenten

Bij het oproepen van de retriever kun je optioneel enkele argumenten meegeven. De belangrijkste twee zijn *inputfile* en *clearresults*. Met het eerste argument geef je de locatie mee van het XML-configuratiebestand. Clearresults zorgt ervoor dat resultaten van een vorige retrieval gewist worden, zodat je met een schone lei begint. In codefragment 3.4 zie je een voorbeeld van hoe je deze argumenten moet gebruiken.

AppConfig

Met het AppConfig-bestand zal de eindgebruiker normaal niet in contact komen. Het bevat de mogelijkheid om het loglevel te veranderen zodat meer of minder logging informatie wordt weggeschreven in het logbestand. De databaseconfiguratiestring kan hier ook opgegeven worden, maar, indien er in het XML-configuratiebestand ook een databank opgegeven wordt, dan wordt de laatste gebruikt. Na het vervangen van het loggingframework (zie paragraaf 4.2.7) komen er enkele loggingopties bij in dit bestand. Zo kun je naast het loglevel ook extra uitvoermogelijkheden opgeven: naar een tekstbestand, naar het consolescherm, naar een databank of een combinatie. Ook het logformaat kan hier desgewenst aangepast worden.

```
SNMPDataRetrieval.exe "-clearresults" "-inputfile=config\snmp.xml"
```

Codefragment 3.4: Oproepen van SNMP Data Retriever met twee argumenten

3.1.2 Databankstructuur

Tijdens het uitvoeren van de SNMP Data Retriever worden er een drietal tabellen aangemaakt: *types*, *devices* en *results*. In de eerste tabel worden de toesteltypes opgeslagen. Een voorbeeld zie je in figuur 3.1. Daar werden twee toesteltypes gedefinieerd: bridge en router. Voor elk toesteltype zijn er een aantal opdrachten (GET- of walkopdrachten), samen met hun datatype, OID, MIB-bestand en naam.

DeviceType	RequestType	AttributeType	oid	mib	name
Bridge	snmpWalk	VARCHAR(50)	1.3.6.1.2.1.1	RFC1213-MIB	system
Bridge	snmpWalk	VARCHAR(50)	1.3.6.1.2.1.2	RFC1213-MIB	interfaces
Bridge	snmpWalk	VARCHAR(50)	1.3.6.1.2.1.4	RFC1213-MIB	ip
Bridge	snmpWalk	VARCHAR(50)	1.3.6.1.2.1.17.1	BRIDGE-MIB	dot1dBase
Bridge	snmpWalk	VARCHAR(50)	1.3.6.1.2.1.17.2	BRIDGE-MIB	dot1dStp
Router	snmpWalk	VARCHAR(50)	1.3.6.1.2.1.1	RFC1213-MIB	system
Router	snmpWalk	VARCHAR(50)	1.3.6.1.2.1.2	RFC1213-MIB	interfaces
Router	snmpWalk	SEQUENCE OF IpRouteEntry	1.3.6.1.2.1.4.21	RFC1213-MIB	ipRouteTable

Figuur 3.1: Tabel van toesteltypes en hun opdrachten

De toestellen zelf worden opgeslagen in de devices tabel. In figuur 3.2 zie je een toestel met zijn IP-adres of hostnaam, het toesteltype en zijn communitystring.

ID	DeviceIPaddress	DeviceType	ReadCommunity	LatestOkRetrieval	LatestStatus	LatestRetrievalAttempt	LatestResult	GNDStatus
1	atlas2a1.intec.ugent.be	Bridge	public	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Figuur 3.2: Tabel van toestellen

De laatste tabel bevat de opgehaalde gegevens. Een aantal resultaten zie je in figuur 3.3. Elk resultaat bevat de host, de OID, het attribuutnaam (dat wordt samengesteld uit de naam van de MIB en de naam van de OID), de waarde van het resultaat en de datum waarop ze werd opgehaald.

3.1.3 Werking

In deze paragraaf geven we een *high-level* overzicht van de werking van de SNMP Data Retriever. Bij het analyseren van de applicatie met een profiler wordt hier ook naar verwezen om te weten wat elke methode juist doet.

De SNMP Data Retriever begint eerst en vooral met de *Initialize*-methode op te roepen. Deze is verantwoordelijk voor de configuratie en het opzetten van de databankverbinding. Zowel de commandlineparameters als het XML-configuratiebestand worden hier verwerkt. Na het inlezen van de configuratie wordt de databankverbinding opgezet met de gegevens uit de configuratie.

Nadat de basisconfiguratie werd ingelezen door de Initialize-methode worden de toesteltypes (devicetypes) ingelezen uit het XML-configuratiebestand. Indien nodig wordt er in de databank een nieuwe devicetypestabel aangemaakt en worden de toesteltypes daarin opgeslagen. Hetzelfde gebeurt voor de te bevragen toestellen. Ten slotte wordt ook de results tabel aangemaakt, indien die nog niet bestond, om de opgehaalde gegevens in weg te schrijven.

Daarna kan het ophalen van de gegevens beginnen. De lijst van de te bevragen toestellen wordt overlopen, en, om meerdere toestellen tegelijkertijd op te vragen, wordt voor elk toestel een nieuwe

DeviceIpAddress	oid	AttributeName	Value	RetrievalDate
atlas2b1.intec.ugent.be	1.3.6.1.2.1.17.1.1.0	BRIDGE-MIB.dot1dBase	00:11:85:A1:0D:00	2014-06-26 17:43:05
atlas2a1.intec.ugent.be	1.3.6.1.2.1.17.1.1.0	BRIDGE-MIB.dot1dBase	00:11:85:A0:A7:00	2014-06-26 17:43:05
atlas2b2.intec.ugent.be	1.3.6.1.2.1.17.1.1.0	BRIDGE-MIB.dot1dBase	00:17:08:F0:DC:00	2014-06-26 17:43:05
atlas2a2.intec.ugent.be	1.3.6.1.2.1.17.1.1.0	BRIDGE-MIB.dot1dBase	00:13:21:83:DE:80	2014-06-26 17:43:05
atlas2b2.intec.ugent.be	1.3.6.1.2.1.17.1.2.0	BRIDGE-MIB.dot1dBase	50	2014-06-26 17:43:05
atlas2b1.intec.ugent.be	1.3.6.1.2.1.17.1.2.0	BRIDGE-MIB.dot1dBase	50	2014-06-26 17:43:05

Figuur 3.3: Tabel van resultaten

thread gestart. Het ophalen zelf gebeurt in de *RetrieveFromDevice*-methode en elke thread voert die methode uit voor een toestel.

Er worden maximaal 50 threads aangemaakt. Bijgevolg kunnen er maar 50 toestellen tegelijkertijd opgevraagd worden. Eens er 50 threads aangemaakt zijn, wordt er gewacht tot een vorige thread klaar is om een nieuwe te starten. De threads worden in een lijst bijgehouden. In een oude versie van de SNMP Data Retriever werd, na het starten van de eerste 50 threads, gewacht op de eerste thread om nieuwe threads aan te maken.

De RetrieveFromDevice-methode overloopt de lijst van instructies die moeten afgehandeld worden voor een bepaald toestel(type). Dat kunnen ofwel enkelvoudige gegevens zijn, die opgehaald moeten worden met een GET-request, ofwel moet een SNMP walk gedaan worden van een bepaalde OID. Vervolgens worden de request(s) gestuurd naar dat toestel. Nadat het antwoord is ontvangen, worden alle variabelen die in dat antwoord zitten overlopen. Er wordt gecontroleerd of de ontvangen gegevens relevant zijn (als ze onder de originele OID vallen bij een SNMP walk), en zoja wordt het gegeven weggeschreven in de databank met de *InsertResultRow*-methode. De InsertResultRow-methode doet niks meer dan het uivoeren van een SQL insert-query. Gegevens worden dus een per een weggeschreven in de databank.

Hoofdstuk 4

Benchmarks en Experimenten

In dit hoofdstuk bespreken we alle uitgevoerde experimenten, zowel op kleine- als grote schaal, om de schaalbaarheid van de SNMP Data Retriever te onderzoeken.

Op kleine schaal maken we gebruik van virtuele machines en een aantal productieswitches om onze testen uit te voeren. We onderzoeken de impact van netwerkvertraging, het nut van het filteren van de gegevens die moeten opgehaald worden, en de impact van bulkrequests. Verder vergelijken we de performantie van de SNMP Data Retriever met de Net-SNMP commandlinetools, analyseren we de SNMP Data Retriever nader met een profiler, onderzoeken we of het beter is om een tabel rij per rij of kolom per kolom op te vragen, en gaan we na wat de impact is van de fragmentatie van pakketten op het netwerk.

Bij het identificeren van problemen die de schaalbaarheid belemmeren, bespreken en implementeren we oplossingen. Achteraf analyseren we ook de effectiviteit van die oplossingen.

Dankzij de samenwerking met het iMinds onderzoekscentrum kunnen we op grote schaal gebruik maken van de Virtual Wall die toelaat om snel geautomatiseerde testnetwerken op te zetten. We maken gebruik van virtualisatie om een groot aantal toestellen te creëren die we kunnen bevragen via SNMP. Met dat groot aantal toestellen kunnen we de uitvoeringstijd, het CPU-, geheugen- en bandbreedtegebruik analyseren die op grote schaal nodig is. We kijken hierbij naar zowel de originele versie van de SNMP Data Retriever, als de versie met onze verbeteringen, om te zien of ze het gewenste effect hebben op de schaalbaarheid.

4.1 Terminologie

Alvorens we beginnen met de benchmarks en experimenten, willen we nog eens de gebruikte terminologie verduidelijken, zodat er zeker geen verwarring ontstaat.

Om te beginnen worden gegevens die een SNMP-agent aanbiedt, voorgesteld door **objecten**. Een object wordt uniek geïdentificeerd door een **OID**. Omdat OID's hiërarchisch zijn opgebouwd, komt een OID niet noodzakelijk overeen met juist één object. Als we een hoger gelegen OID opvragen, krijgen we dus meerdere gegevens terug.

Typisch overlopen we een hoger gelegen OID met een SNMP walkopdracht. Deze blijft gegevens lexicografisch overlopen totdat de opgehaalde gegevens niet meer onder de originele OID vallen. Alle gegevens die wel onder de originele OID vallen, noemen we **relevant**. Maar omdat je extra gegevens moet opvragen om te weten dat er geen gegevens meer onder de originele OID vallen, hebben we ook een aantal **niet-relevante gegevens**.

Ten slotte zijn er twee retrievers waarvan we gebruik maken om SNMP-gegevens op te halen. Enerzijds hebben we de **SNMP Data Retriever** van NetworkMining en anderzijds zijn er de Net-SNMP commandlinetools.

4.2 Kleinschalige benchmarks en experimenten

4.2.1 Virtuele machines

De eerste testopstelling bestaat uit vier virtuele machines die switches in een netwerk nabootsen. Als virtualisatieplatform wordt er gebruik gemaakt van *Oracle VM VirtualBox* (verder gewoon VirtualBox genoemd). VirtualBox is vrij te verkrijgen voor alle gangbare besturingssystemen en is bovendien open-source.

Hardwareconfiguratie

Aan elke node wordt 256 MB geheugen en één CPU-*core* toegewezen. Op de nodes wordt een minimale versie van Debian 7 geïnstalleerd, zonder grafische schil. Hierdoor is zelfs 256 MB een ruime luxe voor de nodes: na het opstarten van een node wordt er amper 70MB geheugen gebruikt.

Alle toestellen zijn rechtstreeks met elkaar verbonden in een privénetwerk. Via Network Address Translation (NAT) kunnen ze via de gastheer toch nog het internet bereiken. Dit werd bewerkstelligd met de *NAT Network mode*, een nieuwe feature in VirtualBox die nog niet in de documentatie staat, maar wel kort beschreven wordt in een nieuwspost (zie[28]). Ter vergelijking: *NAT mode* laat gastsystemen toe om met het internet te communiceren via NAT, maar zitten elk in een apart privénetwerk en kunnen dus niet met elkaar praten. *Host-only mode* laat gastsystemen met elkaar (en het gastheersysteem) communiceren door ze in een gezamelijk privénetwerk te plaatsen, maar communicatie met het internet is niet mogelijk.

Softwareconfiguratie

Hieronder leggen we kort stap voor stap uit hoe je op een Debianinstallatie de nodige SNMP software kunt installeren en configureren. De uitleg is zeer beknopt gehouden en dient enkel om je op weg te helpen. In de testopstelling werden de nodes ook nog geconfigureerd als switches die het Spanning Tree Protocol (STP) uitvoeren, alsook het Link Layer Discovery Protocol (LLDP). Deze extra protocollen bieden informatie aan die via SNMP opgevraagd kan worden en dit zijn ook gegevens die in een realistische situatie opgevraagd kunnen worden. De configuratie als switch en van LLDP wordt hier echter niet besproken.

Om de nodes van een minimale Debian 7 installatie te voorzien, werden geen extra softwarepakketten geselecteerd bij de installatie.

In de veronderstelling dat het internet werkt, beginnen we na de installatie met het updaten van het systeem en het installeren van *sudo*:

```
# apt-get update
# apt-get upgrade
# apt-get install sudo
```

Maak een gebruiker aan en zorg ervoor dat je sudo rechten hebt met behulp van het *visudo* commando of door je gebruiker toe te voegen aan de *sudo* groep.

```
# visudo
# usermod -a -G sudo <jouw gebruikersnaam>
```

Dan installeren we de snmp *daemon* die zal antwoorden op SNMP requests. Om te testen is het ook interessant om de client-side SNMP tools te installeren, alsook een tool om de belangrijkste MIB's te downloaden. Met die MIB's kunnen numerieke OID's omgezet worden naar de meer gebruiksvriendelijke tekstuele voorstelling.

Debian laat standaard niet toe dat niet-vrije (non-free) software[1] geïnstalleerd wordt. De *snmp-mibs-downloader* tool die we nodig hebben om de MIB's te downloaden is daar een voorbeeld van. Om dit toch toe te laten moet je voor elke lijn in /etc/apt/sources.list het non-free-component achteraan toevoegen. Dan krijg je zoiets:

```
deb http://ftp.belnet.be/debian wheezy main non-free
deb-src http://ftp.belnet.be/debian wheezy main non-free
```

Nu kunnen we wel snmp-mibs-downloader en de andere tools installeren.

```
$ sudo apt-get install snmpd snmp snmp-mibs-downloader
```

Het volgende commando zal de MIB's downloaden:

```
$ sudo download-mibs
```

Het gebruik van de MIB's kun je aanzetten door de volgende regel in commentaar te zetten in het bestand /etc/snmp/snmp.conf:

```
mibs :
```

Om *snmpd* te configureren, kun je gebruik maken van het *snmpconf* commando, dat op interactieve wijze configuratiebestanden aanmaakt. We moeten ook nog de locatie van de MIB-bestanden opgeven. Voeg daarom het volgende toe aan /etc/default/snmpd:

```
export MIBS=/usr/share/mibs
```

Vervolgens herstarten we snmpd om de nieuwe configuratie in te laden.

```
$ sudo /etc/init.d/snmpd restart
```

Om te testen of alles werkt, gebruiken we het volgende commando:

```
$ snmpwalk -v 1 -c public localhost mib-2
```

4.2.2 Productieswitches

Naast een viertal virtuele machines kunnen we ook gebruik maken van tien productieswitches die deel uitmaken van het netwerk van iMinds. Deze switches kunnen we gebruiken om na te gaan of de tests met de virtuele machines de realiteit goed kunnen benaderen. Er worden drie verschillende modellen gebruikt van verschillende grootte en functionaliteit. De grootste bevat bijna 300 poorten en functioneert ook als router.

[1] Dit is software die niet volledig vrij is maar op een of andere manier beperkt wordt door zijn licentie. De eisen die gesteld worden aan vrije software voor Debian zijn te vinden in de Debian Free Software Guidelines (DFSG)[6][7].

4.2.3 Netwerkvertraging

Netwerkvertraging of *latency* definiëren we als de zogenaamde *round-trip time*, of de totale tijd die nodig is om een bericht te sturen naar een ander toestel en voor die computer om het antwoord terug naar jou te sturen[2]. De latency tussen twee toestellen kan je gemakkelijk meten vanop een van de twee toestellen met behulp van het *ping*-commando.

Op een LAN-netwerk bedraagt de latency gewoonlijk minder dan 1 ms en valt dus te verwaarlozen. Zodra netwerkverkeer over het internet moet gaan, speelt latency wel een grote rol. Zolang de afstand niet te groot is (als we binnen West-Europa blijven), blijft de latency beperkt tot 10-50 ms. Als we daarentegen communiceren met bv. Amerika, dan spreken we al over 100-200 ms (respectievelijk voor de Oost- en Westkust). Voor sommige toepassingen (zoals bijvoorbeeld real-time spellen) is dit te hoog om nog bruikbaar te zijn.

Het belang voor ons van de latency is voornamelijk bij het opvragen van veel gegevens met GET- en GETNEXT-requests. Zoals uitgelegd in paragraaf 2.6, wordt er slechts één gegeven in een request gestopt en moet je steeds wachten op het antwoord ervan, alvorens je de volgende request stuurt. Als je te maken hebt met een latency van zeg maar 25 ms, dan wil dat zeggen dat je steeds 25 ms moet wachten alvorens je de volgende request kunt sturen. Stel dat je 200 gegevens wenst op te halen, dan doe je hier 5 seconden over (200 gegevens $*$ 25 ms wachten per gegeven).

Was er een netwerkvertraging van slechts 1 ms, dan zou het ophalen van dezelfde 200 gegevens nog maar 200 ms duren. Natuurlijk houden we hier nog geen rekening met de tijd die het toestel zelf nodig heeft om het bericht te verwerken en te beantwoorden.

De originele versie van de SNMP Data Retriever maakte gebruik van GET- en GETNEXT-requests en ondervond dus een grote invloed op de uitvoeringstijd door de netwerkvertraging. Om na te gaan hoe groot de impact juist is, hebben we tests gedaan met en zonder netwerkvertraging.

Meetresultaten

Deze tests hebben we gedaan op de virtuele machines omdat je in Linux gemakkelijk een artificiële netwerkvertraging kunt creëren. Daarvoor gebruik je het volgende commando:

```
$ sudo tc qdisc add dev eth0 root netem delay 25 ms
```

Codefragment 4.1: Artificiële netwerkvertraging creëren in Linux

Dit commando zorgt ervoor dat er een latency van 25 ms gecreëerd wordt op de *eth0* interface. We hebben de test gedaan met de originele versie van de SNMP Data Retriever bij het ondervragen van een of meerdere machines, met en zonder netwerkvertraging.

De resultaten zie je in tabel 4.1. De eerste rij toont de gemiddelde uitvoeringstijd en de tweede het CPU-gebruik van het (volledige) systeem.

	25 ms vertraging		geen vertraging	
	1 toestel	4 toestellen	1 toestel	4 toestellen
Uitvoeringstijd (s):	15,650	15,987	2,693	6,105
CPU-gebruik:	10%	35%	80-90%	100%

Tabel 4.1: De tijd nodig om 1-4 toestellen te ondervragen, met en zonder vertraging

[2] Latency kan ook gedefiniëerd worden als enkel de tijd dat een bericht nodig heeft om in één richting reizen. Dat is echter moeilijker om te meten. Meestal hanteert men de round-trip time omdat dat gemeten kan worden vanaf slechts een punt. [13]

Jammer genoeg wordt er per seconde maar één meting van het CPU-gebruik gedaan, waardoor de kortste test van minder dan drie seconden, bij een toestel zonder vertraging, niet zo nauwkeurig is. Desondanks is het toch voldoende om te zien dat het CPU-gebruik een bottleneck vormt bij het bevragen van vier toestellen zonder vertraging. Het moet wel gezegd worden dat de retriever in een virtuele machine draaide waaraan slechts een CPU-core werd toegewezen.[3]. Ook bij slechts een toestel zonder vertraging zie je dat het CPU-gebruik vrij hoog ligt. Wanneer we een vertraging van 25 ms introduceren is het CPU-gebruik een stuk lager. Hadden we 50 toestellen bevraagd — het maximum aantal dat de retriever tegelijkertijd doet — met een vertraging van 25 ms, dan hadden we evenwel weer in de problemen gezeten.

Op het CPU-gebruik komen we later nog terug, maar de belangrijkste reden dat we deze test gedaan hebben, is om de impact op de uitvoeringstijd in te kunnen schatten. We vragen ongeveer 217 gegevens op per toestel. Voor één enkel toestel zonder vertraging duurde dit ongeveer 2.700 ms. Ter vergelijking, volgens onze eerdere rekenformule had dit ongeveer 200 ms moeten duren. We zaten er dus een factor 10 naast. Er is natuurlijk flink wat meer aan de hand dan enkel het vervoeren van het pakket over de netwerkverbinding. De retriever doet namelijk ook een aantal databankinteracties zoals het aanmaken van tabellen en het wegschrijven van de resultaten in die databank (zie paragraaf 3.1.2).

Als we een vertraging van 25 ms invoeren, neemt de uitvoeringstijd toe met bijna een factor 6 tot 15.650 ms. Onze duidelijk oververeenvoudigde rekensom van daarnet zou uitkomen op een kleine 6 seconden.

Dezelfde vergelijking maken met vier toestellen is jammer genoeg nutteloos vanwege de CPU-bottleneck. Het is wel interessant om te zien dat het ondervragen van extra toestellen slechts een kleine impact heeft op de uitvoeringstijd. Logisch, want de vier toestellen worden tegelijkertijd ondervraagd.

Conclusie

Het is duidelijk dat de netwerkvertraging een belangrijke rol speelt in de uitvoeringstijd van de retriever. We raden daarom dan ook sterk aan om, indien mogelijk, de bevragingen *on-site* te doen (dit wil zeggen op hetzelfde netwerk in plaats van op afstand) om de netwerkvertraging te minimaliseren.

Het effect van de netwerkvertraging kan ook beperkt worden door het gebruik van GETBULK-requests, waarbij meerdere gegevens in een pakket worden gestopt. Als je 10 gegevens in een pakket stopt, betekent dat theoretisch al een snelheidwinst van factor 10. We zullen dit dan ook verder onderzoeken in paragraaf 4.2.5.

4.2.4 Filteren van de op te vragen gegevens

Een van de vragen die we ons gesteld hebben, is of het wel degelijk de moeite waard is om de gegevens die we opvragen te filteren? Is het met andere woorden nuttig dat we ons bezighouden met een lijst op te stellen van enkel de OID's die ons interesseren, of kunnen we evengoed meteen alle gegevens opvragen die een SNMP-agent ons aanbiedt?

[3] Dit was omdat er een VirtualBox-*image* werd voorzien waarop alles reeds voorgeconfigureerd was om snel te kunnen beginnen met het testen van de retriever. Bij latere tests draait de SNMP Data Retriever niet meer in een virtuele machine en beschikt ze over meerdere cores.

Meetresultaten

Voor deze test stellen we een lijst op van de typisch meest interessante OID's bij netwerkcomponenten. We bekijken hoeveel objecten die OID's ons opleveren en vergelijken dit met het totaal aantal objecten dat een SNMP-agent aanbiedt. Omdat de virtuele machines, die als switch geconfigureerd zijn, niet verbonden zijn met echte computers, zullen ze minder gegevens bevatten dan een echte switch. Anderzijds bevatten de virtuele machines heel wat extra informatie die een gewone switch niet heeft, omdat ze een volledig besturingssysteem draaien. Denk hierbij aan bijvoorbeeld een lijst van alle geïnstalleerde softwarepakketten. Omdat het belangrijk is dat deze test de realiteit zo goed mogelijk weergeeft, werd deze test dan ook uitgevoerd op productieswitches.

Dit is de lijst van OID's die we als interessant beschouwen:

```
1.3.6.1.2.1.1           (system)
1.3.6.1.2.1.2           (interfaces)
1.3.6.1.2.1.4           (ip)
1.3.6.1.2.1.17.1        (dot1dBase)
1.3.6.1.2.1.17.2        (dot1dStp)
1.0.8802.1.1.2.1.4.1    (lldpRemTable)
```

De twee OID's die beginnen met *dot1d* hebben betrekking op de switching functionaliteit. *dot1dBase* bevat onder andere de tabel met alle switchpoorten terwijl *dot1dStp* informatie bevat over het Spanning Tree Protocol (STP).

Om te bepalen hoeveel objecten er in totaal aangeboden worden, kun je een SNMP walk doen van OID '.' of '.1'.

Een voorbeeld van de uitvoer die onze test oplevert is het volgende:

```
Host: atlas1a1.intec.ugent.be
Date: Thu Apr 17 17:20:48 CEST 2014

Retrieved 267 objects for dot1dStp (1.3.6.1.2.1.17.2).
Retrieved 1318 objects for dot1dBase (1.3.6.1.2.1.17.1).
Retrieved 7943 objects for interfaces (1.3.6.1.2.1.2).
Retrieved 7 objects for system (1.3.6.1.2.1.1).
Retrieved 1716 objects for ip (1.3.6.1.2.1.4).
Retrieved 207 objects for lldpRemTable (1.0.8802.1.1.2.1.4.1).
Retrieved 11458 objects in total.
The agent has 369388 objects in total.
Percentage of retrieved objects from total: 3.10188744626247%
```

We zien dus hoeveel objecten we terugkrijgen bij het overlopen van elke OID, het totaal aantal objecten dat we teruggekregen hebben bij alle voor ons interessante OID's en hoeveel objecten het toestel in totaal aanbiedt (voor alle ondersteunde OID's). Tenslotte zien we ook nog het percentage van interessante objecten t.o.v. het totaal. Een samenvatting van de resultaten voor de verschillende productieswitches zien we in tabel 4.2. Gemiddeld komen we uit op een percentage van 4,48% interessante gegevens over alle switches heen.

Conclusie

Gezien het lage percentage van gegevens die voor ons echt interessant is, loont het zeker de moeite om een lijst op te stellen van de OID's die interessant zijn. Dit zal sowieso al een groot verschil uitmaken in de uitvoeringstijd omdat er 20 keer minder gegevens moeten opgevraagd worden. Bovendien moeten er dan ook een pak minder gegevens opgeslagen worden, zeker als er meerdere kopieën van dezelfde gegevens bijgehouden worden over een bepaalde tijdspanne.

Host	Interessante objecten	Totaal aantal objecten	Percentage
atlas1a1.intec.ugent.be	11.458	369.388	3,10%
atlas1a2.intec.ugent.be	5.636	123.617	4,56%
atlas2a1.intec.ugent.be	3.247	69.022	4,70%
atlas2a2.intec.ugent.be	3.238	69.201	4,68%
atlas2b1.intec.ugent.be	3.247	69.418	4,68%
atlas2b2.intec.ugent.be	3.238	69.449	4,66%
atlas3a1.intec.ugent.be	3.256	69.952	4,65%
atlas3a2.intec.ugent.be	3.251	69.509	4,68%
atlas3b1.intec.ugent.be	3.247	70.212	4,62%

Tabel 4.2: Aantal interessante objecten t.o.v. het totaal aantal objecten

4.2.5 Bulkrequests

In paragrafen 2.6 en 4.2.3 bespraken we dat het samenbundelen van meerdere objecten in een request, althans theoretisch, een zeer goed idee is. Alvorens we de code vervangen in de SNMP Data Retriever die gebruik maakt van GETNEXT-requests om een SNMP walk te doen, kunnen we beter eerst testen welke snelheidswinsten we mogen verwachten bij het gebruik van bulkrequests. Daarna kunnen we beslissen of het de moeite waard is om deze manier van werken te implementeren.

Om dit te testen, maken we gebruik van reeds bestaande implementaties: de *snmpwalk* en *snmpbulkwalk* tools van Net-SNMP maken gebruik van respectievelijk GETNEXT- en GETBULK-requests om een SNMP walk te doen van een gegeven OID. Uitleg en voorbeelden over hoe je de Net-SNMP tools kunt gebruiken vind je terug in paragraaf 2.7. Herinner je dat we het aantal objecten per pakket kunnen configureren door het aantal max-repetitions te wijzigen. Standaard stond die op 10.

Meetresultaten

Om te beginnen bevragen we een virtuele machine vanaf een andere. In principe kunnen we ook een virtuele machine zichzelf laten bevragen, maar dat is geen realistische situatie want dan komt er geen netwerkverkeer aan te pas. De gemiddelde pingtijd tussen twee virtuele machines is 0,468 ms. We doen achtereenvolgens een SNMP walk van de volledige SNMP-agent (goed voor ongeveer 4200 objecten), gevolgd door meerdere SNMP walk's met bulk-requests met een verschillend aantal objecten per request.

We herhalen elke test meermaals en — omdat het aantal objecten variabel kan zijn over tijd — houden we van elke test het aantal ontvangen objecten bij alsook de tijd die nodig was. Als we op het einde alle objecten en alle tijden sommeren, zijn er twee metrieken waarvan we gebruik kunnen maken. Als we het totaal aantal objecten delen door de totale uitvoeringstijd bekomen we het gemiddeld aantal objecten per tijdseenheid. Het inverse daarvan is de gemiddelde tijd die nodig is om een object op te vragen. Wij zullen gebruik maken van de eerste metriek omdat die wat intuïtiever is.

De resultaten zie je in tabel 4.3. De eerste kolom bevat de operatie, met tussen haken het aantal objecten per request indien het om een SNMP bulkwalk gaat. De tweede kolom bevat het gemiddeld aantal objecten per ms. De derde kolom bevat weer het gemiddeld aantal objecten per ms, maar zonder uitschieters. Hier spreken we van een uitschieter als een operatie er anderhalf keer zo lang over doet als de meeste anderen. De laatste kolom ten slotte bevat het aantal uitschieters.

Bij de resultaten van SNMP bulkwalk's met een groot aantal objecten per request zagen we soms een klein aantal uitschieters (telkens op 100 iteraties). Bijvoorbeeld bij de SNMP bulkwalk's met

Operatie (objecten per request)	Objecten/ms	Objecten/ms*	Uitschieters
SNMP Walk	1,843	1,843	0
SNMP Bulkwalk (10)	6,937	6,937	0
SNMP Bulkwalk (50)	8,752	9,655	5
SNMP Bulkwalk (100)	8,054	9,892	10
SNMP Bulkwalk (200)	7,590	9,845	14
SNMP Bulkwalk (500)	7,854	9,887	13
SNMP Bulkwalk (1000)	7,225	9,857	19

* Zonder uitschieters

Tabel 4.3: Gemiddeld aantal objecten per ms voor SNMP walk en SNMP bulkwalk

1000 objecten per request deden de meeste er ongeveer 0,44 ms over. Een klein aantal deed er echter 1,5 of zelfs 2 ms over. Omdat de andere tijdsmetingen zeer dicht bij elkaar liggen (minder dan 0,10 ms afwijking) vermoeden we dat de virtuele machine tijdelijk minder rekenkracht toegewezen krijgt door het gastheerbesturingssysteem. Wellicht zien we dit verschijnsel vooral bij hoge aantallen objecten per request omdat die meer rekenwerk vereisen. Latere tests, waarbij de SNMP-agent niet op een virtuele machine draait, vertonen dit gedrag niet.

Uit de resultaten kunnen we alvast afleiden dat het samenbundelen van meerdere objecten per request zeer grote snelheidswinsten oplevert, zelfs al voor 10 objecten per request. Daar gaat het al 3,76 keer sneller dan een gewone SNMP walk. Bij de andere gaat het 3,92 (1000 objecten/request) tot 4,75 (50 objecten/request) keer zo snel als we de resultaten met uitschieters bekijken. Als we geen rekening houden met de uitschieters loopt het snelheidsverschil weinig uiteen en gaat het gemiddeld 5,33 keer zo snel voor 50 tot 1000 objecten per request.

We zien dat het weinig zin heeft om meer dan 100 objecten per request te bundelen. Om beter het ideaal aantal objecten per request te schatten, testen we het bereik tussen 10 en 100 objecten per request en voegen we er telkens 10 objecten bij. De resultaten daarvan staan in tabel 4.4.

Operatie (objecten per request)	Objecten/ms	Objecten/ms*	Uitschieters
SNMP Bulkwalk (10)	6,856	6,856	1
SNMP Bulkwalk (20)	8,455	8,455	0
SNMP Bulkwalk (30)	8,873	8,873	0
SNMP Bulkwalk (40)	8,943	8,943	0
SNMP Bulkwalk (50)	8,672	8,913	2
SNMP Bulkwalk (60)	8,709	9,108	2
SNMP Bulkwalk (70)	8,514	9,519	6
SNMP Bulkwalk (80)	9,103	9,560	3
SNMP Bulkwalk (90)	8,945	9,627	3
SNMP Bulkwalk (100)	8,336	9,725	9

* Zonder uitschieters

Tabel 4.4: Gemiddeld aantal objecten per ms voor SNMP bulkwalk met 10-100 objecten per request

Opnieuw zien we een klein aantal uitschieters (nog steeds op 100 iteraties). Verder zien we dat het gemiddeld aantal objecten per ms, althans als we de uitschieters buiten beschouwing laten, stelselmatig toeneemt. Houden we wel rekening met de uitschieters, dan beschouwen we een maximum bij 80 objecten per request.

Alvorens we onze conclusies trekken, herhalen we de test nogmaals maar op een productieswitch. De gemiddelde pingtijd naar deze switch bedraagt 1,831 ms. Dat is nog steeds zeer weinig, maar is toch drie keer meer dan bij de tests hierboven op de virtuele machines. Omdat we hier problemen

ondervonden bij het opvragen van grote aantallen gegevens (zie paragraaf 5.3), hebben we dit aantal beperkt door enkel de interfacetabel te overlopen, wat nog steeds goed is voor ongeveer 2750 objecten. De resultaten van deze test zie je in tabel 4.5.

Operatie (objecten per request)	Objecten/ms	Snelheid t.o.v. walk	Snelheidsverschil t.o.v. vorige operatie
SNMP Walk	0,448	100,00%	
SNMP Bulkwalk (10)	1,484	331,51%	231,51%
SNMP Bulkwalk (20)	1,745	389,88%	17,61%
SNMP Bulkwalk (40)	1,833	409,56%	5,05%
SNMP Bulkwalk (50)	1,848	412,86%	0,81%
SNMP Bulkwalk (100)	1,843	411,67%	-0,29%
SNMP Bulkwalk (200)	1,867	417,04%	1,30%
SNMP Bulkwalk (500)	1,846	412,39%	-1,11%
SNMP Bulkwalk (1000)	1,876	419,03%	1,61%

Tabel 4.5: Gemiddeld aantal objecten per ms voor SNMP walk en SNMP bulkwalk op een productieswitch

De test werd uitgevoerd vanaf dezelfde virtuele machine als de vorige tests, maar er waren deze keer geen uitschieters te zien in de resultaten. Hier zien we dat het op een productieswitch nog steeds sterk de moeite loont om over te stappen op bulk-requests. Met een verviervoudiging van de snelheid komen we goed in de buurt van het snelste resultaat uit de vorige test, waar het vijf keer zo snel ging. De grootste winst zit bij 10 objecten per request, maar er worden nog kleine winsten geboekt bij verdere verhoging. Vanaf 50 objecten per request begint de winst af te vlakken.

Conclusie

De testen met GETBULK-requests zien er veelbelovend uit, met SNMP walk's die tot vier keer sneller gaan dan hun variant met GETNEXT-requests. Het is overduidelijk dat het dan ook sterk de moeite loont om SNMP walk's te implementeren met behulp van GETBULK-requests.

Voor wat betreft het ideaal aantal objecten dat we per request opvragen, valt er te discussiëren. Gezien de resultaten bij de productieswitch uitwijzen dat de snelheidswinsten afvlakken vanaf 50 objecten per request, lijkt ons dat een goed aantal om te gebruiken bij de implementatie voor de SNMP Data Retriever. Alhoewel Net-SNMP een veilige 10 objecten per request aanhoudt, vinden we het toch de moeite om dit verder te verhogen, gezien het toch nog een 25-tal procent sneller kan gaan ten opzichte van GETNEXT-requests.

Als we kijken naar het aantal op te vragen gegevens, dan zien we dat de interfacetabel van een van de kleinere productieswitches alleen al bijna 3000 objecten bevat. In paragraaf 4.2.4 zagen we dat het aantal interessante objecten bij een switch 3 à 5000 bedraagt, tot zelfs ruim 11000 voor een grotere switch. Daarom neigen we des te meer naar hogere aantallen objecten per request als het over netwerkapparatuur gaat.

Natuurlijk kunnen we het aantal objecten per request laten aanpassen door de eindgebruiker, maar 50 objecten per request is een goede standaardwaarde.

4.2.6 SNMP Data Retriever versus Net-SNMP tools

Bij deze test vergelijken we de performantie tussen de SNMP Data Retriever en de Net-SNMP commandlinetools. Alvorens we beginnen, moeten we opmerken dat het niet echt om een eerlijke vergelijking gaat, want de geschiedenis van Net-SNMP dateert al terug van 1992[18]. De SNMP

Data Retriever is daarentegen een jong project dat nog volop ontwikkeld wordt. Gedurende de loop van de masterproef werden nog verschillende updates uitgebracht voor de SNMP Data Retriever. Om een vast referentiepunt doorheen de masterproef aan te houden werd echter geen rekening gehouden met deze updates. Wel zullen we de updates vermelden als die relevant zijn voor een bepaalde test.

Behalve het verschil in maturiteit van de software, doet de SNMP Data Retriever nog iets meer dan de Net-SNMP tools. Met de Net-SNMP tools kun je maar één opdracht uitvoeren (een GET-request of een SNMP walk bijvoorbeeld) op een enkel toestel. Om verschillende opdrachten uit te voeren op verschillende toestellen moet je de verschillende tools meerdere malen uitvoeren. De resultaten worden bij Net-SNMP uiteindelijk simpelweg uitgeschreven op het scherm.

Bij de SNMP Data Retriever daarentegen wordt er gewerkt met een XML-bestand (zie paragraaf 3.1) waarin je alle verschillende toestellen kunt opsommen, en voor elk soort toestel kun je een lijst van opdrachten configureren die moeten uitgevoerd worden voor dat type toestel. Op het einde som je dan alle toestellen op en geef je mee om wat voor soort toestel het gaat zodat de software weet welke opdrachten moeten uitgevoerd worden. Wanneer het programma wordt uitgevoerd, worden alle nodige opdrachten voor alle opgegeven toestellen in parallel uitgevoerd. Bij deze test houden we het wel maar bij één toestel, waardoor dat voordeel wegvalt bij deze test. Na het ophalen van de gegevens worden de resultaten uiteindelijk weggeschreven in een databank (of een XML-bestand in een latere versie). Die gegevens worden dan later door andere tools van NetworkMining verwerkt voor verschillende doeleinden zoals bijvoorbeeld het visualiseren van inventarissen.

Meetresultaten

De test vindt opnieuw plaats tussen twee virtuele machines. Omdat er slechts één machine bevraagd wordt, verliest de SNMP Data Retriever zijn schaalvoordeel door meerdere toestellen tegelijkertijd te ondervragen.

Liefst zouden we, om zoveel mogelijk gegevens op te vragen, een walk doen van een volledige SNMP-agent. Omdat de SNMP Data Retriever de *endOfMibView*-exceptie niet ondersteunt (zie paragraaf 5.2) is dit echter niet mogelijk. Deze exceptie wordt opgegooid wanneer het einde van een MIB-bestand bereikt wordt en er geen verdere gegevens meer zijn. De exceptie is een speciaal geval omdat ze geen gebruik maakt van het error-veld in het antwoordbericht. Het error-veld staat dus onterecht op 0, wat aangeeft dat er zich geen fout heeft voorgedaan. Daarom, en omdat het vrij ongewoon is om een volledige SNMP-agent te overlopen, is het begrijpelijk dat de SNMP Data Retriever nog geen ondersteuning had voor deze exceptie. Dit probleem werd doorgegeven aan NetworkMining en zal in een latere versie opgelost worden.

Omdat we geen volledige SNMP-agent kunnen overlopen, kiezen we een OID die voldoende gegevens bevat. We gaan uiteindelijk voor de *mgmt*-tak (1.3.6.1.2), de ouder van de *mib-2*-tak. Die OID is goed voor ongeveer 1700 objecten. De resultaten staan in tabel 4.6.

Operatie (objecter per request)	Uitvoeringstijd (ms)	Objecten/ms
SNMP Data Retriever	4575	0,357
Net-SNMP Walk	606	2,928
Net-SNMP Bulkwalk (10)	145	12,218
Net-SNMP Bulkwalk (50)	100	17,740

Tabel 4.6: Metingen tussen de SNMP Data Retriever en de Net-SNMP tools

Opnieuw zie je in de eerste kolom over welke operatie het gaat, alsook hoeveel objecten per request opgevraagd werden. In de tweede kolom staat de gemiddelde uitvoeringstijd en in de laatste het gemiddeld aantal objecten dat er werden opgehaald per ms.

Eerst en vooral zien we dat er een groot verschil is tussen de resultaten van de SNMP Data Retriever en de Net-SNMP walkopdracht, alhoewel die beiden GETNEXT-operaties gebruiken. Ook zien we weer hetzelfde grote verschil die GETBULK-requests veroorzaken bij de walk- en bulkwalkopdracht van Net-SNMP.

Het verschil tussen de SNMP Data Retriever en de Net-SNMP walkopdracht valt deels te verklaren door de bijkomende databankinteracties van de eerste. De resultaten geven aan dat er, behalve door het gebruik van GETBULK-requests, ook nog andere dingen te optimaliseren vallen.

We zien ook dat het samenbundelen van 50 objecten per request inderdaad nog een stuk beter is dan slechts 10 objecten. Misschien is het je ook opgevallen dat de resultaten nog een stuk beter zijn hier dan bij de tests over bulkrequests (paragraaf 4.2.5). Nochthans is het enige verschil met die test de datum waarop de test werd uitgevoerd en het aantal objecten dat werd opgehaald. Bij die test overliepen we echter gans de SNMP-agent en kregen we op het einde dan ook een endOfMibView-exceptie, die mogelijks een impact heeft gehad op de resultaten. Ter vergelijking, de bulkwalks toen behaalden bijna 10 objecten per ms tegenover ruim 17 objecten per ms hier.

We hadden graag ook de test herhaald op een productieswitch, maar we stootten hierbij opnieuw op problemen bij het genereren van veel verkeer op het netwerk van iMinds (zie paragraaf 5.3).

Conclusie

Er is nog flink wat ruimte over voor verbetering bij de SNMP Data Retriever als we de uitvoeringstijden vergelijken met die van de Net-SNMP tools. Welke performantiewinsten, en in welke componenten we die kunnen boeken, zal blijken uit de profileranalyse in paragraaf 4.2.7. Ook het verhogen van het aantal objecten per request van 10 naar 50 heeft duidelijk een extra positieve invloed op de uitvoeringstijd.

4.2.7 Profiling van de SNMP Data Retriever

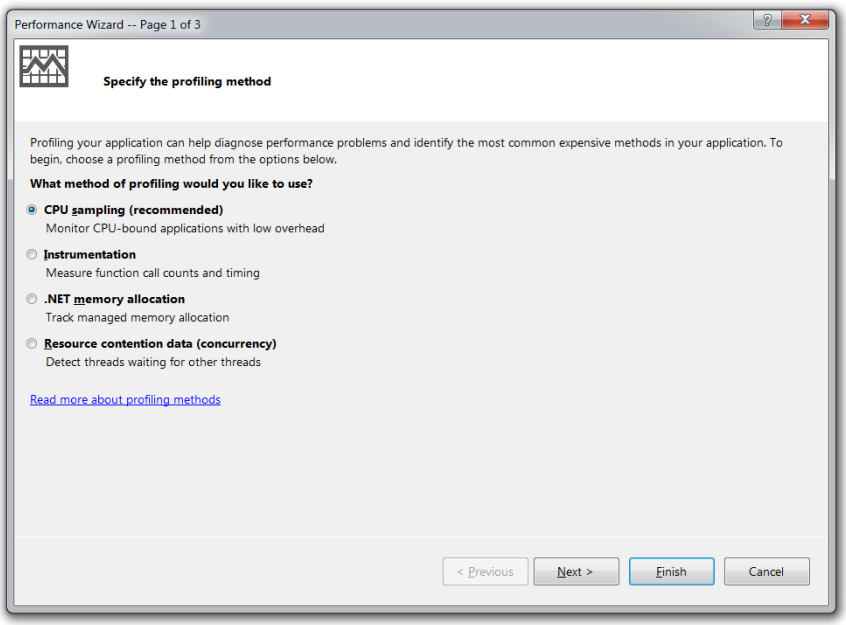
In deze paragraaf gaan we de SNMP Data Retriever van NetworkMining onder de loep nemen met de ingebouwde profiler van Visual Studio. Een profiler zal ons enkele belangrijke inzichten verschaffen over wat er achter de schermen gebeurt bij het uitvoeren van de retriever. Specifiek:

- hoe lang bepaalde stukken code er over doen,
- hoe vaak bepaalde stukken code uitgevoerd worden (zogenaamde hot code/paths),
- problematische stukken code detecteren die er veel langer over doen dan we verwachten,
- en bijgevolg welke stukken het meeste potentieel bieden om te optimaliseren.

Daar waar we problemen vinden of kansen zien om te optimaliseren zullen we de nodige aanpassingen doen en de resultaten daarvan testen. Een van de aanpassingen die we bekijken is het gebruik van GETBULK-requests, zoals onderzocht in paragraaf 4.2.5. Om te beginnen leggen we het gebruik van de profiler uit in Visual Studio 2013.

Gebruik van de profiler in Visual Studio 2013

De profiler in Visual Studio 2013 is erg makkelijk te gebruiken. Om te beginnen open je je project en klik je op *Analyze* in de werkbalk en kies je voor *Performance and Diagnostics*. Als alternatief kun je ook van de ALT+F2 sneltoets gebruik maken. Als *Target* staat standaard het huidige project geselecteerd. Onder *Available Tools* zou normaal ook de *Performance Wizard* moeten geselecteerd zijn. Zoals de beschrijving al verklapt, houdt dit onder andere het meten van CPU- en RAM-gebruik in. Figuur 4.1 toont de *Performance Wizard* die je te zien krijgt als je op *Start*



Figuur 4.1: De Performance Wizard

klikt. Hier moet je de profilingmethode kiezen die je wenst te gebruiken. Wij kiezen voor de eerste: *CPU Sampling*. De rest van de stappen staan standaard goed zodat je meteen op *Finish* mag klikken.

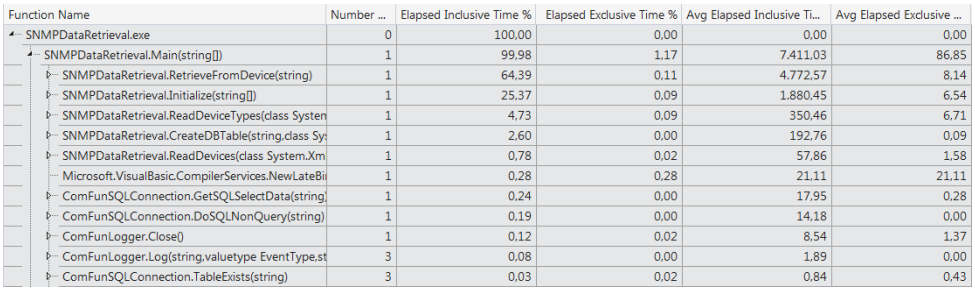
Wanner het programma klaar is met uitvoeren, worden de resultaten geanalyseerd. Als dat klaar is, krijg je een algemeen overzicht van de resultaten. Eerst en vooral zie je een grafiek met het CPU-gebruik doorheen de uitvoeringstijd van de applicatie. Daaronder zie je de *Hot Paths*, dat zijn de functies die verantwoordelijk zijn voor het grootste deel van de uitvoeringstijd. Waar wij in geïnteresseerd zijn is hieraan gerelateerd: de *Call Tree View*. Die geeft je een boomstructuur van functies die elkaar oproepen en enkele belangrijke statistieken: hoeveel keer een functie werd opgeroepen en hoeveel tijd de functie gemiddeld nodig had om uit te voeren, dit zowel procentueel (ten opzichte van de totale uitvoeringstijd) als in absolute tijd.

De call tree van de SNMP Data Retriever kun je zien in figuur 4.2. Hierbij werd de *Main* functie opengeklapt. Je kunt functies verder open klappen om te zien welke andere functies worden opgeroepen en hun aandeel in de uitvoeringstijd analyseren. Dit kan verder gaan tot je atomaire functies krijgt die geen andere functies meer oproepen.

In de call tree zie je naast de functienaam een aantal verschillende kolommen. Hieronder volgt de lijst van de kolommen en hun betekenis.

- **Number of calls:** dit spreekt voor zichzelf. Dit is het aantal keren dat een functie opgeroepen werd.
- **Elapsed Inclusive Time %:** dit is het percentage van de uitvoeringstijd dat werd gespendeerd in deze functie en haar kinderen.
- **Elapsed Exclusive Time %:** dit is het percentage van de uitvoeringstijd dat uitsluitend in deze functie werd gespendeerd, dus *exclusief* haar kinderen.
- **Avg Elapsed Inclusive Time:** dit is de gemiddelde uitvoeringstijd in milliseconden van deze functie en haar kinderen.
- **Avg Elapsed Exclusive Time:** dit is de gemiddelde uitvoeringstijd in milliseconden van

uitsluitend deze functie, dus weer *exclusief* haar kinderen.



Function Name	Number ...	Elapsed Inclusive Time %	Elapsed Exclusive Time %	Avg Elapsed Inclusive Ti...	Avg Elapsed Exclusive ...
SNMPDataRetrieval.exe	0	100,00	0,00	0,00	0,00
SNMPDataRetrieval.Main(string[])	1	99,98	1,17	7.411,03	86,85
SNMPDataRetrieval.RetrieveFromDevice(string)	1	64,39	0,11	4.772,57	8,14
SNMPDataRetrieval.Initialize(string[])	1	25,37	0,09	1.880,45	6,54
SNMPDataRetrieval.ReadDeviceTypes(class Systen	1	4,73	0,09	350,46	6,71
SNMPDataRetrieval.CreateDBTable(string,class Sy	1	2,60	0,00	192,76	0,09
SNMPDataRetrieval.ReadDevices(class System.Xm	1	0,78	0,02	57,86	1,58
Microsoft.VisualBasic.CompilerServices.NewLateBi	1	0,28	0,28	21,11	21,11
ComFunSQLConnection.GetSQLSelectData(string,	1	0,24	0,00	17,95	0,28
ComFunSQLConnection.DoSQLNonQuery(string)	1	0,19	0,00	14,18	0,00
ComFunLogger.Close()	1	0,12	0,02	8,54	1,37
ComFunLogger.Log(string,valuetype EventType.st	3	0,08	0,00	1,89	0,00
ComFunSQLConnection.TableExists(string)	3	0,03	0,02	0,84	0,43

Figuur 4.2: De Call Tree. De kolommen van links naar rechts: *Function Name*, *Elapsed Inclusive Time %*, *Elapsed Exclusive Time %*, *Avg Elapsed Inclusive Time*, *Avg Elapsed Exclusive Time*.

Meetresultaten

Bij dit experiment is het nodig dat we de SNMP Data Retriever lokaal uitvoeren en dus niet meer in een virtuele machine. Voor het wegschrijven van de resultaten installeren we een MariaDB-databank, maar MySQL is ook een optie.

We doen een walk van twee OID's op een enkele productieswitch. Dit zal ons 443 objecten opleveren, waarvan 441 die voor ons relevant zijn. Volgens de profiler doet de retriever daar ongeveer 7,4 seconden over. De call tree van de retriever hebben we voor de leesbaarheid in een tabel gegoten (zie tabel 4.7). De functies met een extreem kleine uitvoeringstijd (minder dan 0,01%) hebben we achterwege gelaten. De kolommen die je ziet zijn de functie, het aantal oproepen, de *inclusieve* tijd als percentage van de totale uitvoeringstijd en de gemiddelde *inclusieve* tijd in milliseconden.

Voor een globaal overzicht van de werking van de SNMP Data Retriever verwijzen we naar paragraaf 3.1.3.

Functie	Calls	Tijd (%)	Tijd (ms)
SNMPDataRetrieval.RetrieveFromDevice	1	64,39	4.772,57
SNMPDataRetrieval.Initialize	1	25,37	1.880,45
SNMPDataRetrieval.ReadDeviceTypes	1	4,73	350,46
SNMPDataRetrieval.CreateDBTable	1	2,60	192,76
SNMPDataRetrieval.ReadDevices	1	0,78	57,86
Microsoft.VisualBasic.CompilerServices.NewLateBinding.L...	1	0,28	21,11
ComFunSQLConnection.GetSQLSelectData	1	0,24	17,95
ComFunSQLConnection.DoSQLNonQuery	1	0,19	14,18
ComFunLogger.Close	1	0,12	8,54
ComFunLogger.Log	3	0,08	1,89
ComFunSQLConnection.TableExists	3	0,03	0,84

Tabel 4.7: De call tree van de Main methode

RetrieveFromDevice-methode

We beginnen met de functie die het meeste tijd in beslag neemt: de *RetrieveFromDevice*-functie. Zoals de naam al insinueert, gaat het hier om de methode die de requests stuurt om de gevraagde gegevens op te halen van de verschillende toestellen.

De methode werkt als volgt: voor elk toestel dat ondervraagd moet worden, wordt er een aparte thread gestart, met een maximum van 50 threads. Elk van die threads zal alle gegevens opvragen die opgevraagd moeten worden voor dat toesteltype (zie de configuratie van de SNMP Data Retriever in paragraaf 3.1.1). Wanneer een thread klaar is met gegevens opvragen, wordt de thread verwijderd. Indien er nog toestellen zijn die moeten ondervraagd worden, zal er een nieuwe thread opgestart worden voor het volgende toestel. Dit gaat zo door tot alle toestellen zijn ondervraagd.

In paragraaf 2.6 werden de verschillende SNMP operaties besproken waarmee men gegevens kan opvragen. De originele versie van de SNMP Data Retriever die we eerst testen, maakt gebruik van GET-requests voor enkelvoudige gegevens en GETNEXT-requests om een SNMP walk te doen van een volledige deelboom.

De call tree van de RetrieveFromDevice-functie zie je in tabel 4.8. De twee belangrijkste methoden hier zijn de *SyncRequest* en de *InsertResultRow*-methoden. SyncRequest maakt deel uit van de *SnmpSource* bibliotheek. Dit is de third party bibliotheek waarvan gebruik gemaakt wordt om alle SNMP interacties af te handelen. De SyncRequest methode wordt gebruikt om synchroon een request te versturen. Het feit dat de request synchroon gebeurt wil zeggen dat de code wacht op het antwoord alvorens verder te gaan. We zien dat de methode 443 keer is opgeroepen: er zijn dus 443 requests verstuurd geweest. Gemiddeld deed een request er een kleine 10 milliseconden over, allen samen goed voor bijna 60% (oftewel 4,28 seconden) van de totale uitvoeringstijd.

Functie	Calls	Tijd (%)	Tijd (ms)
SnmpSource.SnmpSession.SyncRequest	443	57,74	9,66
SNMPDataRetrieval.InsertResultRow	441	3,75	0,63
SnmpSource.SnmpSession..ctor	1	1,85	137,38
Microsoft.VisualBasic.CompilerServices.NewLateBinding.L...	445	0,42	0,07
ComFunSQLConnection.TableContainsColumn	2	0,12	4,43
ComFunLogger.Log	451	0,07	0,01
SnmpSource.SnmpPdu..ctor	1	0,05	3,84
Microsoft.VisualBasic.CompilerServices.Conversions.ToBo...	2	0,04	1,59
Microsoft.VisualBasic.CompilerServices.NewLateBinding.L...	6	0,04	0,52
SnmpSource.SnmpVariable.CreateSnmpVariable	443	0,02	0,00

Tabel 4.8: De call tree van de RetrieveFromDevice methode

Het versturen van de GETNEXT-requests en het wachten op de antwoorden voor de SNMP walkopdrachten zijn dus verantwoordelijk voor meer dan de helft van de totale uitvoeringstijd. Rekening houdend met de resultaten uit paragraaf 4.2.5 waarbij GETBULK-requests vier keer sneller gingen dan GETNEXT-requests, was de keuze dan ook snel gemaakt om de SNMP walkopdracht te implementeren met behulp van GETBULK-requests.

Na het implementeren van de SNMP walkopdracht met behulp van GETBULK-requests doen we een test op basis van dezelfde twee OID's als hierboven. Zoals we in paragraaf 4.2.5 beslist hebben, vragen we 50 objecten per request. We vergelijken de resultaten in tabel 4.9.

De eerste twee rijen geven de totale uitvoeringstijd weer van de SNMP Data Retriever, de laatste twee tonen de uitvoeringstijd van enkel de RetrieveFromDevice-methode gemeten met de profiler.

We zien dat de gemiddelde uitvoeringstijd met ruim vier seconden ingekort wordt, goed voor een ruime verdubbeling van de snelheid. Als we echter enkel kijken naar de RetrieveFromDevice-methode, zien we dat de methode zelf zeven keer sneller is geworden dankzij het gebruik van BULK-requests.

In tabel 4.10 zie je hoe de call tree van de RetrieveFromDevice methode er met de nieuwe implementatie uitziet. Nu doen de SyncRequest oproepen er drie keer zo lang over, maar er moeten maar 11 requests meer verstuurd worden in plaats van 443. Twee derde van de uitvoeringstijd van

	Uitvoeringstijd (s)	Objecten/ms
SNMP Walk met GETNEXT-requests	7,371	0,064
SNMP Walk met GETBULK-requests (50)	3,131	0,141
RetrieveFromDevice-methode met GETNEXT-requests	5,060	0,093
RetrieveFromDevice-methode met GETBULK-requests (50)	0,722	0,611

Tabel 4.9: SNMP Walk op basis van GETNEXT-requests versus GETBULK-requests in de SNMP Data Retriever

Functie	Calls	Tijd (%)	Tijd (ms)
SnmpSource.SnmpSession.SyncRequest	11	11,07	33,62
SNMPDataRetrieval.InsertResultRow	441	6,46	0,49
Microsoft.VisualBasic.CompilerServices.NewLateBinding.L...	1.768	1,79	0,03
SnmpSource.SnmpSession..ctor	1	0,58	19,50
Microsoft.VisualBasic.CompilerServices.NewLateBinding.L...	443	0,58	0,04
SNMPDataRetrieval.GetRequestsForDevice	1	0,34	11,22
SnmpSource.SnmpPdu..ctor	1	0,11	3,72
Microsoft.VisualBasic.CompilerServices.Conversions.ToBo...	441	0,10	0,01
ComFunLogger.Log	1	0,05	1,64
System.Array.Exists	6	0,05	0,27
SnmpSource.SnmpVariable.CreateSnmpVariable	11	0,04	0,13

Tabel 4.10: De call tree van de RetrieveFromDevice methode met GETBULK-requests

de RetrieveFromDevice-methode wordt besteed aan het ophalen van de gegevens en een derde aan het wegschrijven ervan. De requests zelf zijn nu nog maar voor 11% i.p.v. 58% verantwoordelijk voor de totale uitvoeringstijd.

De call tree van de Main-methode met de nieuwe implementatie zien we in tabel 4.11. De retrievalmethode die gebruik maakt van GETBULK-requests hebben we in een andere methode gestoken die de originele vervangt: RetrieveFromDevice2. De methode die zowel de gegevens ophaalt als wegschrijft is nu nog verantwoordelijk voor minder dan 22% i.p.v. ruim 64% van de totale uitvoeringstijd en is zoals eerder gezegd zeven keer sneller dan de vorige implementatie.

De InsertResultRow-methode in de RetrieveFromDevice-methode hebben we nog niet bekeken. Die is verantwoordelijk voor het wegschrijven van een gegeven in de databank. Omdat er 441 gegevens zijn die binnen de SNMP walk vallen, werd de methode 441 keer opgeroepen.

Eén gegeven per keer in de databank wegschrijven klinkt even inefficiënt als het ophalen van één gegeven per keer. Ook het invoegen van gegevens in een databank kan in bulk gebeuren met behulp van een zogeheten *bulk insert* in plaats van een gewone *insert*. Hierbij worden de gegevens als één geheel in de databank ingevoerd. Het voordeel daarbij is dat er zoveel mogelijk gegevens samengebundeld worden in pakketten, waardoor er minder tijd nodig is om alle gegevens te versturen. De databank hoeft ook maar één query te verwerken in plaats van één voor elk gegeven.

Als we kijken naar de tijd die nodig is om alle gegevens in de databank weg te schrijven, zien we dat dit slechts iets meer dan 200 ms of 6,5% van de totale uitvoeringstijd vereist. We beslissen dan ook dat het niet de moeite is om dit te implementeren, omdat er elders nog veel grotere winsten kunnen gehaald worden[4].

[4] Wat we echter over het hoofd hebben gezien is dat we de lokale databankinstallatie geïnstalleerd hebben op een

Functie	Calls	Tijd (%)	Tijd (ms)
SNMPDataRetrieval.Initialize	1	55,57	1.857,07
SNMPDataRetrieval.RetrieveFromDevice2	1	21,62	722,49
SNMPDataRetrieval.ReadDeviceTypes	1	12,03	401,84
SNMPDataRetrieval.CreateDBTable	1	5,31	177,32
SNMPDataRetrieval.ReadDevices	1	1,39	46,56
ComFunSQLConnection.DoSQLNonQuery	1	0,79	26,49
ComFunSQLConnection.GetSQLSelectData	1	0,52	17,33
Microsoft.VisualBasic.CompilerServices.NewLateBinding.L...	1	0,47	15,71
ComFunLogger.Close	1	0,22	7,40
ComFunLogger.Log	3	0,20	2,23
ComFunSQLConnection.TableExists	3	0,06	0,72

Tabel 4.11: De call tree van de Main-methode met GETBULK-requests

Initialize-methode

Als we de call tree van de Main-methode bekijken, dan valt het op dat er naast de RetrieveFromDevice-methode, die logischerwijs een belangrijke factor is in de uitvoeringstijd, nog een methode is die voor een groot stuk medeverantwoordelijk is voor de uitvoeringstijd: de *Initialize*-methode.

In de originele implementatie was die verantwoordelijk voor 25% van de uitvoeringstijd, in de nieuwe is dit opgelopen tot bijna 56%! De Initialize-methode doet er bijna 2 seconden over. Bijgevolg zijn we erg niewsgierig wat er juist in deze methode gebeurt waardoor deze zoveel tijd nodig heeft. We beginnen met het analyseren van de call tree van de functie, te zien in tabel 4.12. Wat we zien zijn twee oproepen naar een *Log*-methode die er gemiddeld bijna een seconde over doet *per oproep*!

Functie	Calls	Tijd (%)	Tijd (ms)
ComFunLogger.Log	2	45,61	762,05
ComFunSQLConnection..ctor	1	8,15	272,18
ComFunLogger.set_LogFile	1	0,42	14,17
Microsoft.VisualBasic.CompilerServices.Conversions.ToIn...	2	0,32	5,41
ComFunLogger.Log	4	0,29	2,44
System.Configuration.ConfigurationManager.get_AppSettin...	12	0,22	0,61
ComFunLogger.Log	3	0,13	1,46
ComFunLogger.Log	2	0,10	1,73
ComFun.NetworkMiningCopyRightStatement	1	0,06	2,14
ComFunLogger..ctor	1	0,03	1,07

Tabel 4.12: De call tree van de Initialize-methode

De *ComFunLogger* is een stuk code die gebruikt wordt om te loggen naar een tekstbestand. De naam komt van het feit dat ze een gemeenschappelijk stuk code is die over meerdere projecten kan gebruikt worden: *common functions*, of *ComFun* voor kort. Maar een functie die loggegevens wegschrijft naar een bestand zou niet zo lang mogen duren. I/O-operaties zijn kostbaar, maar niet in die mate. Als we de functie helemaal openklappen in figuur 4.3, vinden we de echte dader: een constructor van *System.Diagnostics.PerformanceCounter*. Een performance counter wordt gebruikt voor het monitoren van systeemcomponenten zoals processoren, geheugen en netwerk-I/O.

SSD. Een databank heeft enorm veel baat bij het gebruik van een SSD door de lage *random access* tijd en heeft weinig moeite om een groot aantal inserts bij te houden zoals onze test. Pas bij de grootschalige tests ontdekken we onze fout, en daar komen we er dan ook op terug.

Als je counters gebruikt in je applicatie kunnen ze je informatie geven over de performantie van je programma.[8] De ComFunLogger gebruikt ze om het geheugengebruik te meten en weg te schrijven in de logbestanden.

Function Name	Number of C...	Elapsed Inclusive Time %	Elapsed Excl...	Avg Elapsed Incl...	Avg Elapsed Excl...
SNMPDataRetrieval.Initialize(string[])	1	55,57	0,20	1.857,07	6,54
ComFunLogger.Log(string,valuetype EventType,valuetype ComFunLog	2	45,61	0,02	762,05	0,37
ComFunLogger.Log(string,valuetype EventType,valuetype ComFun	2	45,59	0,11	761,68	1,82
ComFunLogger.ComFunLogItem..ctor(class ComFunLogger)	2	44,19	0,00	738,32	0,00
System.Diagnostics.PerformanceCounter..ctor(string,string)	2	43,89	43,89	733,40	733,40
System.Diagnostics.PerformanceCounter.NextValue()	2	0,20	0,20	3,32	3,32
System.Diagnostics.Process.get_PrivateMemorySize64()	2	0,09	0,09	1,54	1,54
System.Diagnostics.Process.GetCurrentProcess()	2	0,00	0,00	0,06	0,06
System.DateTime.get_Now()	2	0,00	0,00	0,00	0,00
System.Math.Round(float64)	2	0,00	0,00	0,00	0,00
System.Object..ctor()	2	0,00	0,00	0,00	0,00
ComFunLogger.Log(class ComFunLogger/ComFunLogItem,boo	2	1,29	0,09	21,54	1,51

Figuur 4.3: De call tree van de Log-methode

Als oplossing werd ervoor gekozen om een alternatief *logging framework* te gebruiken. Dit lijkt een drastische maatregel (en dat is het ook), maar daar zijn goede redenen voor. De noodzaak aan een tool om informatie te loggen is een algemeen probleem. Er zijn dan ook een heleboel gratis en open-source logging frameworks die reeds hun nut en kunnen bewezen hebben. Ontwikkelaars doen er goed aan om deze bestaande tools te hergebruiken in hun projecten.

Financiëel is het een goede keuze want de software is al ontwikkeld en werd al grondig getest door vele anderen. Dit spaart tijd en geld uit voor de ontwikkeling van een eigen oplossing. De software is gratis in gebruik: er zijn geen licentiekosten aan verbonden. Er zijn ook geen of lage onderhoudskosten.

Ook vanuit het technisch perspectief is het een goede keuze. Omdat de software reeds ingezet wordt in verschillende projecten werd ze zo flexibel mogelijk gemaakt en bevat ze reeds een resem aan extra functionaliteit. Zo kan je loggen naar meerdere outputs en zijn er meerdere mogelijke outputs beschikbaar. Je kunt bijvoorbeeld loggen naar tekstbestanden, consolevensters, databanken, enzovoort. Mocht er toch iets toegevoegd dienen te worden, dan beschik je over de broncode om dit gemakkelijk zelf te kunnen doen.

De twee belangrijkste redenen echter zijn het feit dat ze ontwikkeld zijn om een zo klein mogelijke performantieimpact te hebben en dat de integratie ervan in een project zeer eenvoudig is. Zo was het veel sneller om een ander logging framework te gebruiken dan om de bestaande loggingcode te doorspitten op zoek naar problemen en mogelijke optimalisaties.

De keuze uit de verschillende logging frameworks is uiteindelijk gevallen op Apache log4net. Deze is gebaseerd op het waarschijnlijk bekendste logging framework voor Java: Apache log4j. Bij de keuze werd rekening gehouden met de performantieimpact en de features van de verschillende logging frameworks.[5]

Net als bij het implementeren van GETBULK-requests laten we opnieuw de SNMP Data Retriever dezelfde OID's opvragen van een productieswitch. We vergelijken de resultaten met de vorige testen in tabel 4.13. Om zicht te hebben op de overhead van de SNMP Data Retriever bekijken we ook de performantie van enkel de RetrieveFromDevice-methode (ophalen en wegschrijven in databank) en alle SyncRequest-methoden samen (enkel het ophalen van de gegevens). Ten slotte vergelijken we ook met de resultaten die we in het verleden behaalden met Net-SNMP bij het bevragen van dezelfde switch. Toen ging het wel om een pak meer gegevens: ongeveer 2750 tegenover ongeveer 441 hier.

Opnieuw zien we een grote verbetering: de uitvoeringstijd is nu 1,7 keer sneller ten opzichte van de vorige versie, en vier keer sneller dan de originele versie! Alhoewel we zeer grote performantiewinsten

[5] Een vergelijking tussen de bekendste logging frameworks voor .NET vind je terug in de bronnenlijst bij bron [3] en [4].

	Uitvoeringstijd (s)	Objecten/ms
SNMP Walk met GETNEXT-requests	7,371	0,064
SNMP Walk met GETBULK-requests (50)	3,131	0,141
SNMP Walk met GETBULK-requests en log4net (50)	1,839	0,240
RetrieveFromDevice-methode met GETBULK-requests (50)	0,722	0,611
SyncRequest-methode met GETBULK-requests (50)	0,387	1,138
Net-SNMP Bulkwalk (50)	1,488	1,848

Tabel 4.13: Vergelijking van de verschillende implementaties van SNMP Data Retriever en Net-SNMP

Functie	Calls	Tijd (%)	Tijd (ms)
SNMPDataRetrieval.RetrieveFromDevice2	1	38,80	725,19
SNMPDataRetrieval.ReadDeviceTypes	1	17,20	321,51
SNMPDataRetrieval.Initialize	1	17,02	318,13
log4net.Config.XmlConfigurator.Configure	1	10,25	191,52
SNMPDataRetrieval.CreateDBTable	1	8,28	154,77
SNMPDataRetrieval.ReadDevices	1	2,50	46,77
ComFunSQLConnection.GetSQLSelectData	1	0,95	17,75
Microsoft.VisualBasic.CompilerServices.NewLateBinding.L...	1	0,87	16,26
ComFunSQLConnection.DoSQLNonQuery	1	0,75	14,02
log4net.LogManager.GetLogger	1	0,44	8,14
ComFunSQLConnection.TableExists	3	0,12	0,76
log4net.ILog.Info	3	0,06	0,39

Tabel 4.14: De call tree van de Main methode met GETBULK-requests en log4net

behaald hebben ten opzichte van de originel versie van de SNMP Data Retriever, als we enkel kijken naar het ophalen van de gegevens door de SyncRequest-methode, dan is dit nog steeds 38% trager dan bij Net-SNMP. Maar omdat deze methode onderdeel is van de third-party bibliotheek SnmpSource kunnen we daar verder geen verbeteringen meer aanbrengen.

De call tree van de Main-methode van de nieuwste versie zie je in tabel 4.14. De totale uitvoeringstijd wordt nu voornamelijk bepaald door de volgende onderdelen:

- 38,80% (725 ms): het bevragen van het toestel en het wegschrijven van de gegevens in de databank.
- 17,20% (322 ms): het inlezen van de toesteltypes uit het XML-configuratiebestand. Het gebruik van reguliere expressies heeft hier de meeste impact op.
- 17,02% (318 ms): het initialiseren, de meeste tijd wordt hier besteed aan de constructie van een SQL *Connection*-object (250 ms).
- 10,25% (192 ms): het inlezen van het log4net-configuratiebestand.
- 8,28% (155 ms): het aanmaken van de databanktabellen.
- 2,50% (47 ms): het inlezen van de te bevragen toestellen uit het XML-configuratiebestand.

Conclusie

Het implementeren van de SNMP walkopdracht met behulp van GETBULK-requests en het vervangen van het logging framework door log4net heeft ervoor gezorgd dat de SNMP Data Retriever vier keer sneller werkt in vergelijking met de originele versie. Alhoewel we het hier niet bekeken hebben, is er nog een optimalisatiemogelijkheid bij het wegschrijven van de gegevens in de databank. Daar kunnen de gegevens best in bulk weggeschreven worden in plaats van gegeven per gegeven zoals nu.

4.2.8 Tabellen rij per rij opvragen i.p.v. kolom per kolom

Zoals uitgelegd in paragraaf 2.5 worden tabellen kolom per kolom overlopen bij een SNMP walkopdracht. Een meer natuurlijke manier van werken zou zijn om de tabel rij per rij op te vragen. En misschien valt die methode zelfs sneller uit.

Een gewone SNMP walk wordt geïmplementeerd door GETNEXT-requests (of GETBULK-requests) en vraagt zo steeds de volgende OID op. Om een tabel rij per rij op te vragen, moeten we de opeenvolgende kolommen van de eerste rij opvragen, gevolgd door de kolommen van de tweede enzovoort. Maar het kan nog sneller: we kunnen meteen alle kolommen van de eerste rij opvragen, gevolgd door alle kolommen van de tweede rij enzovoort. Om dat te doen kunnen we een GETNEXT-request opstellen met de OID's van alle kolommen van een tabel. Een GETNEXT-request geeft ons dan de kolommen terug van de eerste rij en zo weten we meteen ook de index waarmee we de volgende request kunnen samenstellen. Dat geeft ons dan de kolommen van de tweede rij en zo gaan we verder tot we de volledige tabel hebben opgehaald.

Jammer genoeg is het niet voldoende om met Net-SNMP een snmpbulkwalk te doen met de OID's van alle kolommen want die doet simpelweg sequentieel een snmpbulkwalk van elke opgegeven OID. Dus moeten we het wiel heruitvinden, en een snmpbulkwalk implementeren die wel alle OID's in parallel overloopt en op het juiste moment stopt.

Daarvoor maken we gebruik van GETBULK-requests met de OID's van alle kolommen. Door het aantal herhalingen in te stellen kunnen we meteen ook meerdere rijen tegelijkertijd opvragen. Het aantal max-repetitions is dan gelijk aan het aantal rijen dat in een request wordt opgevraagd. Om het aantal objecten in een request te weten te komen, vermenigvuldig je simpelweg het aantal max-repetitions met het aantal kolommen.

Om op het juiste moment te stoppen moeten we ofwel op voorhand het aantal rijen kennen, ofwel moeten we de teruggegeven OID's bekijken om te zien of ze nog steeds deel uitmaken van de tabel, zoals een gewone SNMP walk doet.

De voorwaarde om dit alles te kunnen doen, is dat we vooraf de OID's van de kolommen van de tabel kennen. Om de snelheid te vergelijken tussen beiden manieren om een tabel te overlopen, bepalen we het maximum aantal rijen dat we tegelijkertijd kunnen opvragen, zodat het aantal objecten in een request onder 50 blijft. Dan overlopen we de tabel met een gewone SNMP walk op basis van GETBULK-requests met hetzelfde aantal objecten in een request.

Een voorbeeld: de interfacetabel ifTabel heeft 22 kolommen, dus dan vragen we 2 rijen per request op (goed voor 44 objecten per request). Bij de gewone snmpbulkwalk geven we dan als aantal max-repetitions 44 op zodat beide methoden evenveel objecten per request opvragen.

Meetresultaten

Om te beginnen vragen we de dot1dBasePortTable (1.3.6.1.2.1.17.1.4) op van de productieswitches bij iMinds. Vanwege de problemen bij het genereren van grote hoeveelheden verkeer (paragraaf 5.3) konden we slechts één iteratie per keer uitvoeren. Alhoewel we de test meermaals hebben herhaald

om zeker te zijn dat er geen grote verschillen zijn in de metingen, is het toch belangrijk om in het achterhoofd te houden dat de resultaten niet erg nauwkeurig zijn.

Toestel	Objecten	Uitvoeringstijd rij per rij (s)	Uitvoeringstijd kolom per kolom (s)
atlas1a1	1315	1,09	0,59
atlas2a1	615	0,36	0,25
atlas2a2	250	0,37	0,28
atlas3a1	250	0,32	0,21

Tabel 4.15: Tabel rij per rij opvragen versus kolom per kolom

Ondanks de minder nauwkeurige resultaten in tabel 4.15 kunnen we wel een algemene trend waarnemen: de tabel kolom per kolom ophalen is gemiddeld ongeveer de helft sneller dan ze rij per rij op te halen. Deze trend zien we ook bij de andere productieswitches. Voor alle duidelijkheid: deze testen zijn gebaseerd op de Net-SNMP tools.

Naar aanleiding van de grootschalige testen hebben we al een kleine opstelling klaar, die gelijkaardig is aan de virtuele machines qua softwareconfiguratie, maar waarbij elke machine op een *dedicated server*[6] draait. Omdat deze servers in een apart netwerk zitten hebben we gelukkig geen problemen met het genereren van veel verkeer. Daar kunnen we zoveel verkeer versturen als de fysieke verbinding aankan. We kunnen daar dezelfde test doen maar met voldoende herhalingen om een nauwkeuriger resultaat te bekomen.

De ping tijd tussen de twee toestellen is gemiddeld 0,204 ms. Dat is nog lager dan tussen de virtuele toestellen, die 0,468 ms bedroeg, en ongeveer negen keer lager dan bij de productieswitches van hierboven. Bij deze test vragen we de interfacetabel ifTable (1.3.6.1.2.1.2.2) op. De tabel heeft 22 kolommen en de server heeft zes interfaces, samen goed voor 154 objecten. Omdat 154 objecten wat aan de lage kant is, kunnen we een aantal VLAN's toewijzen aan interfaces om het aantal objecten te verhogen, want VLAN's komen ook in de interfacetabel terecht. Om een VLAN aan te maken gebruik je het volgend commando:

```
$ sudo ip link add link eth3 name eth3.100 type vlan id 100
```

Codefragment 4.2: Aanmaken van een VLAN

Met dit commando wordt het VLAN met ID 100 toegewezen aan interface *eth3* en krijgt die de naam *eth3.100*. We doen dit voor een 100-tal VLAN's, goed voor 2200 extra objecten of 2354 objecten in totaal in de tabel.

	Objecten	Uitvoeringstijd (ms)	Objecten/ms
Rij per rij	154	109	1,413
Kolom per kolom	154	30	5,133
Rij per rij	2354	1523	1,546
Kolom per kolom	2354	80	29,425

Tabel 4.16: Tabel rij per rij opvragen versus kolom per kolom op dedicated hardware

In tabel 4.16 zien we de resultaten. Opnieuw is het sneller om een tabel kolom per kolom op te vragen in plaats van rij per rij. Deze keer is het zelfs ruim drie keer sneller bij 154 objecten. 2354 objecten zijn ongeveer 15 keer meer objecten, en de tabel rij per rij overlopen doet er ook 15 keer langer over. Bij het kolom per kolom overlopen echter is de uitvoeringstijd amper met een factor

[6] Bij een dedicated server worden de resources van een machine volledig toegewijd tot een of meer taken, in tegenstelling tot een virtuele machine die slechts een deel van de resources van een toestel krijgt.

drie toegenomen! We hebben de test (van 100 iteraties) meermaals uitgevoerd om te bevestigen dat de resultaten kloppen, maar de metingen bleken zeer consistent.

Het is duidelijk dat het sneller is om tabellen kolom per kolom op te vragen. Maar behalve het snelheidsverschil zijn er nog andere redenen om tabellen op die manier te overlopen:

- De implementatie bestaat al en is uitvoerig getest.
- Voorkennis van de tabel is vereist om een tabel rij per rij te overlopen: er moet dus een goede integratie zijn van de MIB-bestanden om op de kolommen van elke tabel te bepalen en om te weten welke OID's overeenstemmen met een tabel.
- Als de tabel deel uitmaakt van een SNMP walk van een hoger gelegen OID, dan moet er tijdens de uitvoering van implementatie gewisseld worden bij de OID van de tabel, wat extra complexiteit veroorzaakt.
- Er moet rekening gehouden worden met randsituaties. Zo zou het mogelijk zijn dat er celwaarden in een tabel ontbreken volgens [10]. Wij hebben dit echter nooit kunnen vaststellen. Bij het rij per rij overlopen zou een antwoordpakket dus al een kolomwaarde kunnen bevatten van de volgende rij!
- Het configureren van het aantal objecten dat per request opgehaald moet worden, is niet zo flexibel: omdat je enkel het aantal rijen kunt opgeven, is het aantal objecten per request automatisch een veelvoud van het aantal kolommen.

Conclusie

Het rij per rij overlopen van een tabel is niet alleen trager dan ze kolom per kolom te overlopen, ze brengt ook een groot aantal problemen mee bij de implementatie. Indien je toch een tabel rij per rij wil *lezen*, dan kan je beter gebruik maken van bijvoorbeeld de Net-SNMP *snmptable* tool. Deze vraagt een tabel kolom per kolom op zoals gewoonlijk, maar beeldt ze rij per rij af en houdt ook rekening met speciale gevallen zoals gaten in de tabel.

4.2.9 Impact van de fragmentatie van pakketten

In dit experiment onderzoeken we de impact van de fragmentatie van pakketten. Dit doet zich voor als we proberen teveel objecten in een request te steken waardoor ze groter wordt dan de MTU en bijgevolg moet opgesplitst worden in meerdere pakketten alvorens ze verstuurd kan worden.

We trachten om de gemiddelde grootte te bepalen van objecten om een schatting te doen van het ideaal aantal objecten per request zodat we zo veel mogelijk objecten per request kunnen opvragen maar er toch zo weinig mogelijk fragmentatie voorkomt. Dit experiment wordt bemoeilijkt omdat de objecten en zelfs de headers van een SNMP-bericht geen vaste grootte hebben. Om de gemiddelde objectgrootte te bepalen, moeten we een SNMP-bericht volledig ontleden en de grootte van alle headers bepalen. Eenmaal we de gemiddelde objectgrootte kennen, berekenen we hoeveel objecten er in een niet-gefragmenteerd bericht passen. Vervolgens meten we hoelang een gefragmenteerd bericht er over doet tegenover een niet-gefragmenteerd bericht, hoeveel gefragmenteerde berichten er werkelijk zijn in een SNMP walk, en tenslotte vergelijken we de uitvoeringstijd met een SNMP walk met een "optimaal" aantal objecten per request.

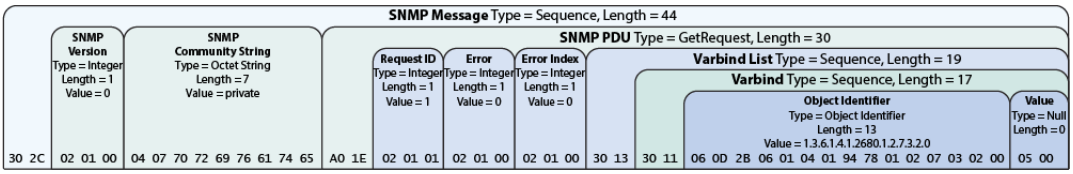
Gemiddelde objectgrootte en optimaal aantal objecten per request

Om de gemiddelde objectgrootte te berekenen, doen we een SNMP walk van de mib-2-tak (1.3.6.1.2.1). We gebruiken de variant op basis van GETNEXT-requests, zodat elk antwoordbericht juist één object bevat. Dat levert ons 3560 *antwoord*berichten op. De gemiddelde lengte van een

antwoordbericht bedraagt 102,26 bytes. Dus nemen we willekeurig een antwoordbericht van 102 bytes om te ontleden.

Om te beginnen hebben we de ethernetheader, goed voor 14 bytes[14]. Daarna volgt de IP-header, typisch van 20 bytes[11]. Na de IP-header komt de SNMP-PDU die de overige 68 bytes (102−14−20) voor zijn rekening neemt.

De berichtstructuur van een SNMP-bericht hebben we besproken in paragraaf 2.9. Zoals gezegd hebben SNMP-berichten een variable headerlengte. Maar veel headervelden hebben in de praktijk een vaste lengte. We overlopen alle velden van een SNMP-header en bepalen hun typische lengte. Als referentie kun je nogmaals de structuur van een SNMP-bericht bekijken in figuur 4.4.



Figuur 4.4: Berichtstructuur van een SNMP-bericht in detail[2]

Het SNMP-bericht is een sequentie van:

- De SNMP versie: steeds 1 byte.
- De community string: variabel, maar uitgaande van de community ”public” gaat het om 6 bytes.
- De SNMP-PDU: bestaat uit:
  - De request ID: de unieke identificatie van een request. Bij de SNMP walk hadden alle requests de ID 1, dus: 1 byte.
  - Het errorveld: een byte.
  - Error Index: staat op nul indien er zich geen fout heeft voorgedaan, dus ook 1 byte.
  - Varbind List: een sequentie van varbinds, maar bij de SNMP walk op basis van GETNEXT-requests gaat het om slechts een object.
    - Varbind: stelt een object voor en is hetgeen we trachten te bepalen
      - OID: variabel aantal bytes
      - Value: variabel aantal bytes

In totaal heeft een SNMP-bericht dus 8 vaste TLV-tripletten (tot en met de Varbind List) met elk een overhead van 2 bytes voor de tag en length, wat goed is voor 16 bytes. De values die een vaste lengte hebben zijn samen goed voor 10 bytes. Elk object wordt voorgesteld door een Varbind bestaande uit een OID en een value, samen goed voor een overhead van 6 bytes per object. Uiteindelijk komen we uit op een headerlengte van 26 bytes voor het SNMP-bericht zonder objecten.

Als we de headerlengte van 26 bytes aftrekken van de 68 bytes van het SNMP-gedeelte van ons GETNEXT-response, houden we nog 42 bytes over voor de varbind die het object en zijn OID en value bevatten. Een gemiddeld object heeft dus een lengte van 42 bytes, inclusief zijn headers.

We gaan uit van een MTU van 1500 bytes voor een ethernetpakket. Als we daarvan de lengtes aftrekken van de ethernet-, IP- en SNMP-header houden we nog 1440 bytes ($1500 - 14 - 20 - 26$) over voor de objecten en hun headers. Uitgaande van een gemiddelde objectlengte van 42 bytes is dat goed voor iets meer dan 34 objecten.

Meetresultaten

We laten de SNMP Data Retriever op basis van de nieuwe implementatie een SNMP walk doen met 34 en 50 objecten. Met behulp van Wireshark analyseren we de pakketten van beide tests.

We bepalen voor beide tests de impact van de fragmentatie op antwoordberichten door de gemiddelde tijd te meten die de pakketten nodig hebben. De tijd die een pakket nodig heeft is dan het tijdsverschil tussen het versturen van de request tot het ontvangen van het responsbericht. We bepalen tenslotte voor beide tests ook het percentage pakketten die gefragmenteerd werden.

	Objecten/request	Aantal	Percentage	Tijd (ms)	Tijdsverschil (ms)
Niet-gefragmenteerde pakketten	50	22		6,996	
Gefragmenteerd pakketten	50	50	69,44%	9,822	2,826 (+40,39%)
Niet-gefragmenteerde pakketten	34	83		5,843	
Gefragmenteerd pakketten	34	22	20,95%	7,882	2,039 (+34,9%)

Tabel 4.17: Impact en percentage van gefragmenteerde pakketten bij bulkrequests

In tabel 4.17 zien we het aantal gefragmenteerde- en niet-gefragmenteerde pakketten bij bulkrequests van 34 en 50 objecten per request. Ook getoond is de gemiddelde tijd die nodig was om een antwoordbericht te ontvangen en het tijdsverschil tussen de gefragmenteerde- en niet-gefragmenteerde pakketten. We zien dat bij 50 objecten per request het percentage van pakketten dat gefragmenteerd werd vrij hoog ligt: net geen 70% en dat ze er ook 40% langer over doen. Bij 34 objecten per request is dat percentage sterk gedaald en we zien ook dat de tijd die de pakketten erover doen gedaald is als gevolg van het kleiner aantal objecten per request.

Maar de vraag is nu of het loont om 32% minder objecten in een pakket te steken zodat ze 35 à 40% sneller gaan? Daarvoor laten we de SNMP Data Retriever een SNMP walk doen van de *enterprises*-tak (1.3.6.1.4.1) en de gekende mib-2-tak, samen goed voor 3558 objecten.

Objecten/request	Objecten	Uitvoeringstijd (s)	Tijdsverschil (s)
50	3558	17,735	
34	3558	18,626	0,891 (+5,02%)

Tabel 4.18: Gemiddelde uitvoeringstijd van een SNMP walk met 50 en 34 objecten per request

In tabel 4.18 zien we dat de SNMP walk met 34 objecten per request er gemiddeld bijna een seconde (of 5%) langer over doet dan de SNMP walk met 50 objecten per request. Het is duidelijk dat de tijdswinst van niet-gefragmenteerde pakketten niet opweegt tegen de tijdswinst bij het opvragen van meer objecten per request.

Conclusie

Ondanks het feit dat we antwoordberichten een stuk sneller krijgen als ze niet gefragmenteerd hoeven te worden, lijkt die tijdswinst toch niet op te wegen tegen de tijdswinst van het opvragen van meer requests per object. We kunnen dus beter meer requests per object opvragen, zelfs als dat een hoge mate van fragmentatie veroorzaakt.

4.3 Grootschalige benchmarks en experimenten

4.3.1 Testopstelling

Virtual Wall

De Virtual Wall is een testomgeving bij iMinds voor geavanceerde netwerken, gedistribueerde software en dienstenevaluatie. Ze ondersteunt het onderzoek naar de schaalbaarheid van toepassingen. De Virtual Wall bestaat uit een kleine 200 zware servers met processoren gaande van vier tot twaalf cores, tot 24 GB geheugen en hebben tot zes gigabit netwerkinterfaces. De servers zijn verbonden met grote switches die als schakelbord functioneren. Zo kunnen de servers vanop afstand op elke mogelijke wijze met elkaar verbonden worden. De servers zelf kunnen ook op afstand voorzien worden van voorgeconfigureerde software images. Zo kunnen servers geconfigureerd worden als server, klant of zelfs als een netwerkelement. Op het netwerk kan men ook testen doen met verbindingen die pakketten verliezen, beperkte bandbreedte hebben of een grote netwerkvertraging ondervinden[24, 25].

Figuur 4.5: De Virtual Wall bij iMinds[25]

Overzicht opstelling

De bedoeling van de opstelling op de Virtual Wall is om een situatie na te bootsen waarbij een groot aantal toestellen ondervraagd moet worden. Omdat de Virtual Wall gedeeld wordt door meerdere onderzoekers kunnen we natuurlijk niet zomaar de helft van de servers gebruiken voor ons experiment. Bovendien hadden onze virtuele machines in VirtualBox genoeg met een core en 256 MB geheugen voor een SNMP-agent. Dan zou een machine met twaalf cores en 24 GB geheugen natuurlijk zware *overkill* zijn. Wat we dus doen is een groot aantal virtuele machines opzetten op een paar van die servers. Er zijn verschillende virtualisatietechnologieën die we kunnen aanwenden waaronder OpenVZ en Xen. Vanwege problemen die verder besproken worden in paragraaf 5.4, werd er geopteerd voor Linux Containers (LXC). Met LXC kunnen we een 50-tal virtuele machines laten draaien op een enkele server. We hebben dan uiteindelijk maar drie servers nodig: twee waarop de virtuele machines draaien en een waarop de SNMP Data Retriever draait. Elk van die 100 virtuele machines zal dan een eigen SNMP-agent draaien.

Linux Containers

LXC is een lichte virtualisatietechnologie voor Linux die werkt op het niveau van het besturingssysteem. Daarom gaat het niet om een volwaardige virtuele machine, maar voorziet het een virtuele omgeving met een eigen procesruimte, bestandssysteem en netwerkinterfaces. LXC maakt hiervoor

gebruik van de resourcemanagement- en resource-isolatiefeatures in de kernel zoals *control groups (cgroups)*[7] en *namespace isolation*[8][16, 26].

Netwerkconfiguratie

De fysieke netwerktopologie is zeer eenvoudig: de drie servers worden simpelweg met elkaar verbonden via een switch. De Linux containers (vanaf nu nodes genoemd) zijn daarmee echter nog niet bereikbaar. Daarvoor moet er op elk van de twee hostmachines een bridge[9] geconfigureerd worden met enerzijds zijn eigen netwerkinterface en anderzijds alle nodes die op die machine draaien. Daarvoor moet je voor elke node twee virtuele interfaces aanmaken die met elkaar verbonden zijn. Eén interface wijs je dan toe aan de bridge en de ander aan de node[15]. De commando's daarvoor zien er als volgt uit:

```
# Bepaal de PID van de container vnode0:
$ pid=$(sudo lxc-info -pHn vnode0)

# Maak een paar virtuele interfaces aan die met elkaar verbonden zijn:
$ sudo ip link add name veth0 type veth peer name veth0_container

# Wijs een interface toe aan de bridge br0:
$ sudo brctl addif br0 veth0

# Wijs de andere interface toe aan de container:
$ sudo ip link set dev veth0_container netns $pid name veth0
```

Codefragment 4.3: Linux containers verbinden met een bridge[15]

Schematisch ziet de combinatie van de fysieke netwerkopstelling en de netwerkconfiguratie op de hosts er uit als in figuur 4.6.

Softwareconfiguratie containers

Alle 100 nodes hebben een gelijkaardige softwareconfiguratie als van onze virtuele machines in VirtualBox van paragraaf 4.2.1. De nodes zijn voorzien van hun eigen SNMP-agent en draaien het LLDP-protocol. Het verschil met de kleinschalige opstelling is dat ze niet als bridge geconfigureerd zijn. Het belangrijkste is dat elke node over voldoende gegevens beschikt die we kunnen opvragen, en dat aantal kunnen we indien nodig makkelijk artificieel verhogen, door het creëren van VLAN's (zoals in paragraaf 4.2.8).

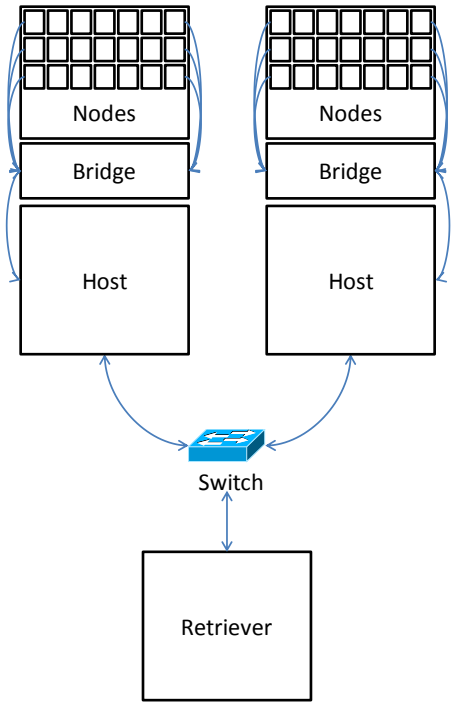
4.3.2 Impact databankinteracties

Als je de voetnoten gelezen hebt in onze tests met de profiler in paragraaf 4.2.7, dan weet je dat we daar een fout hebben gemaakt door te vergelijken met resultaten waarbij we de databank geïnstalleerd hadden op een SSD. Daardoor werd het probleem van het wegschrijven van de opgehaalde gegevens in de databank sterk geminimaliseerd. Omdat de SNMP Data Retriever en de databank in de Virtual Wallopstelling geïnstalleerd werden op een traditionele harde schijf, merkten we dan ook snel op dat we bijlange niet dezelfde performantiewinsten behaalden als bij onze kleinschalige tests. Ook NetworkMining ondervond dezelfde performantieproblemen bij het wegschrijven naar de databank en hadden daarop een nieuwe versie ontwikkeld die als alternatief de opgehaalde gegevens naar een XML-bestand kan wegschrijven.

[7] cgroups zorgen voor de resource-isolatie van de CPU, het geheugen, I/O, netwerk enzovoort[16].

[8] Namespace isolation zorgt ervoor dat applicaties een volledig geïsoleerd zicht hebben op het besturingssysteem, waaronder processen, het netwerk, gebruikers en bestandssystemen[16].

[9] Ondanks het feit dat deze functie in Linux een bridge genoemd wordt, komt de functionaliteit ervan overeen met een switch. De naam komt van het feit dat ze een brug vormt tussen twee of meer netwerkverbindingen.



Figuur 4.6: Netwerkopstelling Virtual Wall

Wegens tijdsgebrek hebben we het gebruik van bulk inserts niet meer kunnen implementeren. Om toch de impact van het wegschrijven van de gegevens te kunnen meten, hebben we een optie ingebouwd die deze databankinteracties simpelweg achterwege laat.

Meetresultaten

We doen de test met 1, 10 en 100 nodes en dat zowel voor de originele versie als de nieuwe versie die gebruik maakt van bulkrequests en log4net als loggingframework. Om de tabelgrootte te beperken, vermelden we echter niet elke combinatie. We doen opnieuw een SNMP walk van de mib-2- en enterprises-OID's, nog steeds goed voor 3558 objecten.

Nodes	Versie	Insert	Uitvoeringstijd (s)	Factor	Factor nieuw vs oud
1	Nieuw	Ja	6,634		
		Nee	2,669	2,49	
10	Oud	Ja	77,333		
		Nee	29,762	2,60	
100	Oud	Ja	1100,418		
		Nee	125,820	8,75	
	Nieuw	Ja	1051,373		1,05
		Nee	17,229	61,02	7,30

Tabel 4.19: Impact van de databankinteracties

De resultaten zie je in tabel 4.19. De eerste kolom duidt het aantal nodes aan, de tweede de versie, waarbij "oud" de originele versie aanduidt en "nieuw" de versie die gebruik maakt van bulkrequests en log4net. De derde kolom geeft aan of de resultaten worden weggeschreven in de databank of

niet, de vierde de gemiddelde uitvoeringstijd en de vijfde geeft aan hoeveel keer sneller de versie zonder inserts is. De laatste kolom tenslotte geeft aan hoeveel keer sneller de nieuwe versie is tegenover de oude versie, met en zonder inserts.

De resultaten laten een duidelijk beeld zien: er is een gigantisch verschil in uitvoeringstijd indien men wel of niet de gegevens wegschrijft in de databank. Gaande van 2,5 keer bij een tot tien nodes, tot ruim 60 keer sneller bij de nieuwe versie! Het is duidelijk dat het aanpakken van de inserts een zeer hoge prioriteit moet krijgen en dat het gebruik van bulk inserts sterk aangeraden is.

We zien ook dat het verschil tussen de oude versie en de nieuwe versie verwaarloosbaar is bij een groot aantal nodes als men de databankinserts niet aanpakt: de nieuwe versie is daar slechts 5% sneller. Maar als de retriever niet tegengehouden wordt door de databaseinserts, is de nieuwe versie ruim zeven keer sneller als er veel toestellen bevraagd moeten worden! Het is dus belangrijk om de databankinteracties aan te pakken om de nieuwe versie van de SNMP Data Retriever volledig tot zijn recht te laten komen.

Conclusie

Zowel bij kleine, maar vooral bij grote aantallen toestellen die bevraagd moet worden is er een grote impact op de uitvoeringstijd door de databankinteracties. Het gebruik van bulk inserts moet dan ook een topprioriteit gemaakt worden. Een mogelijke manier van werken zou zijn om het wegschrijven van gegevens in een aparte thread te laten verlopen. Zo zijn er geen 50 threads die tegelijkertijd proberen hun data weg te schrijven naar de databank. Men kan een tabel in het geheugen bijhouden om initieel de resultaten in bij te houden. In een aparte thread kan men dan de data uit die tabel wegschrijven naar de databank in zo groot mogelijke blokken met bulk inserts.

Omdat het implementeren en testen van die functionaliteit redelijk wat tijd in beslag kan nemen, kan men overwegen om op korte termijn de databank op te slaan op een SSD om zo het effect van de databankinteracties te minimaliseren.

4.3.3 Benchmarks uitvoeringstijd

Om de schaalbaarheid van de SNMP Data Retriever te verbeteren hebben we al twee grote aanpassingen gedaan aan de SNMP Data Retriever, namelijk het gebruik van bulkrequests en een alternatief loggingframework. In deze paragraaf beoordelen we de uitvoeringstijd van de nieuwe versie van de SNMP Data Retriever met onze aanpassingen ten opzichte van de oude. In paragraaf 4.3.2 hebben we ook gezien dat de databankinteracties een grote negatieve impact hebben op de uitvoeringstijd. Omdat we daarvoor geen oplossing hebben geïmplementeerd, moeten we het stellen bij het vergelijken van de uitvoeringstijd met en zonder databankinteracties. Ten slotte bekijken we ook de invloed van het aantal te bevragen toestellen op de uitvoeringstijd en op basis daarvan doen we een schatting voor het opvragen van tot 10.000 toestellen.

Uitvoeringstijd op grote schaal

De uitvoeringstijd van de SNMP Data Retriever op grote schaal hebben we al gemeten in de paragraaf over de impact van de databankinteracties. We nemen die resultaten er bijgevolg weer bij. In die paragraaf hebben we al gezien dat het verschil tussen de oude versie en de nieuwe slechts 5% bedraagt als de gegevens in de databank weggeschreven moeten worden. Wanneer dat niet het geval is, is de nieuwe versie ruim zever keer sneller dan de oude! Het is dus van absoluut belang dat dit probleem aangepakt wordt, of dat er gebruik gemaakt wordt van een SSD om de databank op te slaan. Natuurlijk zal een alternatieve implementatie met bulk inserts niet exact 7,3 keer

sneller zijn, maar het geeft wel een goed idee van hoeveel sneller de SNMP Data Retriever zou kunnen gaan.

Nodes	Versie	Insert	Uitvoeringstijd (s)	Factor nieuw vs oud
100	Oud	Ja	1100,418	
		Nee	125,820	
	Nieuw	Ja	1051,373	1,05
		Nee	17,229	7,30

Tabel 4.20: Uitvoeringstijd van de SNMP Data Retriever op grote schaal

Invloed aantal bevraagde toestellen

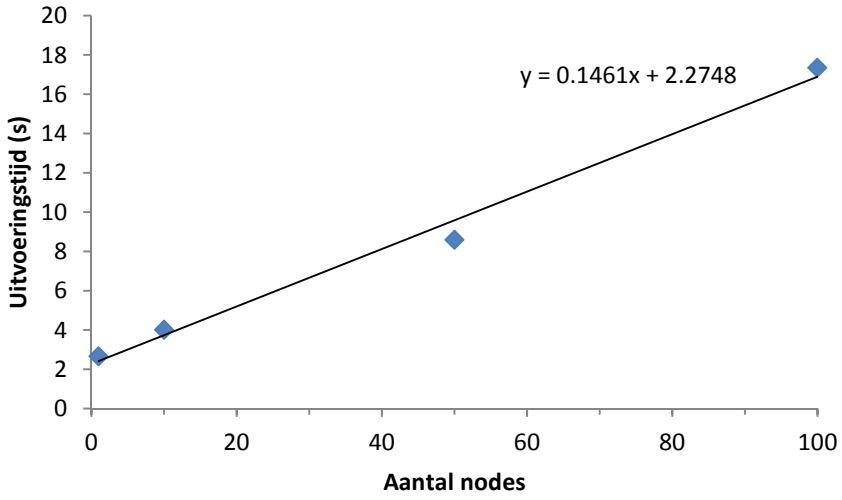
In tabel 4.21 zien we de gemiddelde uitvoeringstijd van de SNMP Data Retriever bij 1, 10, 50 en 100 nodes, zonder databankinteracties. Alhoewel er tot 50 toestellen tegelijkertijd opgevraagd kunnen worden, zien we toch dat de uitvoeringstijd ook bij minder dan 50 toestellen toeneemt.

	1 node	10 nodes	50 nodes	100 nodes
Uitvoeringstijd (s)	2,669	4,013	8,590	17,351

Tabel 4.21: Uitvoeringstijd bij een verschillend aantal toestellen

Om te zien hoe sterk de uitvoeringstijd juist toeneemt met het aantal te bevragen toestellen, hebben we de resultaten in een grafiek voorgesteld en de trendlijn getekend.

Het is duidelijk dat er een lineair verband is tussen het aantal toestellen dat moet bevraagd worden en de uitvoeringstijd. De trendlijn leert ons dat elk *extra* toestel dat bevraagd moet worden ongeveer 0,15 seconden extra vraagt.



Figuur 4.7: Uitvoeringstijd bij een verschillend aantal toestellen

In tabel 4.22 hebben we de theoretische uitvoeringstijd berekend aan de hand van de bovenstaande formule voor 1 tot 10.000 toestellen. We zien dat de resultaten weinig afwijken van de reële resultaten. De theoretische uitvoeringstijd voor 1000 toestellen bedraagt amper 2,5 minuten en voor 10.000 toestellen gaat het om minder dan 25 minuten.

Toestellen	Uitvoeringstijd (s)
1	2,669
2	2,815
3	2,961
4	3,107
5	3,253
10	3,984
20	5,445
30	6,906
40	8,367
50	9,828
100	17,133
1000	148,623
10000	1463,523

Tabel 4.22: Theoretische uitvoeringstijd bij een verschillend aantal toestellen

Conclusie

Op voorwaarde dat het probleem met de databankinteracties aangepakt wordt, kunnen we besluiten dat de gemaakte verbeteringen al een zeer positieve invloed hebben op de uitvoeringstijd van de SNMP Data Retriever op grote schaal.

Ook de impact van het aantal te bevragen toestellen op de uitvoeringstijd blijft zeer beperkt met ongeveer 0,15 seconden extra per bijkomend toestel.

Een lage uitvoeringstijd is echter niet het enige criterium om te beslissen of de retriever goed schaalbaar is: ook met andere resources moet goed omgegaan worden, zoals CPU, geheugen en bandbreedte. Die zaken onderzoeken we in de volgende paragrafen.

4.3.4 Benchmarks CPU-gebruik

In deze paragraaf bekijken we de invloed van het aantal op te vragen toestellen op het CPU-gebruik van de SNMP Data Retriever. We bekijken ook het CPU-gebruik gedurende de uitvoering van het programma en bespreken het gedrag ervan. Ook het aantal threads gedurende de uitvoering die gebruikt worden om de toestellen te bevragen nemen we onder de loep.

Meetresultaten

In figuur 4.8 zien we het CPU-gebruik van de SNMP Data Retriever bij een verschillend aantal nodes. Misschien vraag je je af waarom het CPU-gebruik over de 100% gaat. De reden daarvoor is omdat de server waarop de retriever draait in totaal 16 cores heeft. De belasting van een volledige core komt dan overeen met 100%, twee cores met 200% enzovoort. Als je dus een CPU-gebruik van 900% ziet, dan komt dat ongeveer overeen met een volledige belasting van 9 cores. In de praktijk wordt de werklast echter verdeeld over meer dan 9 cores zodat ze niet allemaal volledig belast worden.

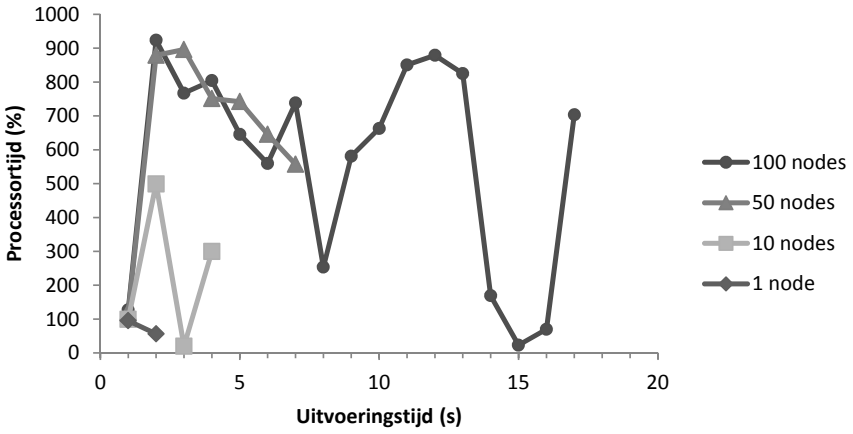
We zien dat het CPU-gebruik meestijgt met het aantal nodes. Vanaf 50 nodes echter stijgt het CPU-gebruik niet verder voorbij iets meer dan 900%. Dat is logisch, want er worden maar maximaal 50 toestellen tegelijkertijd opgevraagd.

Naar het einde toe zien we iets vreemds. We verwachten dat, eenmaal het resterend aantal te bevragen toestellen onder de 50 zakt, het CPU-gebruik geleidelijk aan zou moeten zakken tot 0%

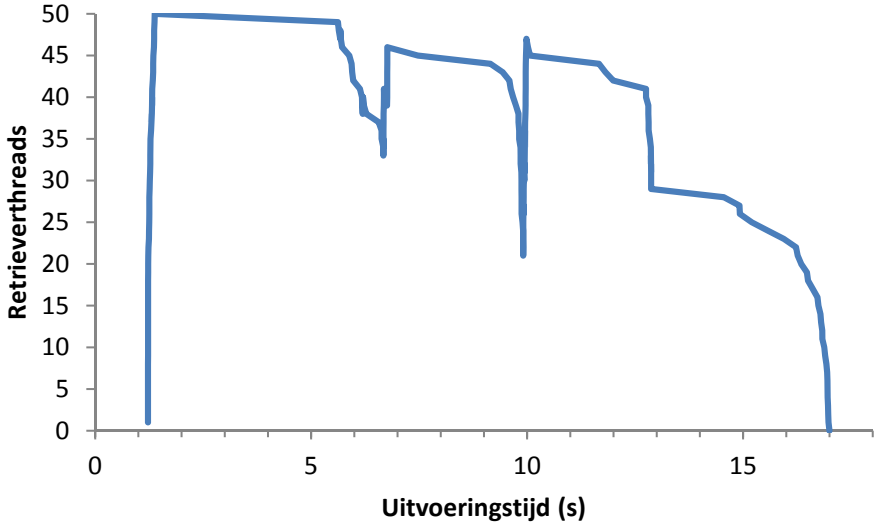
wanneer alle toestellen bevraagd zijn en het programma afsluit. Wat we echter zien is dat het CPU-gebruik bij 100 nodes vanaf 13 seconden sterk zakt om kort daarna weer sterk te stijgen.

Met behulp van het logbestand en Wireshark kunnen we de oorzaak vinden. Met het logbestand kunnen we zien hoeveel threads er actief zijn om toestellen te bevragen, en met Wireshark kunnen we zien hoeveel requests onderweg zijn. Een grafiek van het aantal retrieverthreads zie je in figuur 4.9.

Het probleem dat zich voordoet is het volgende: er zijn op het einde minder dan 50 toestellen die nog moeten bevraagd worden, waardoor er minder dan 50 threads actief zijn. In de grafiek met het aantal threads zie je rond 13 seconden dat het aantal threads daalt, omdat de laatste antwoorden binnenkomen van een aantal toestellen die dus afgewerkt zijn. Omdat er verder geen toestellen meer zijn worden er ook geen nieuwe threads meer gestart. De threads die wel nog actief zijn blijken uit Wireshark allemaal nog te wachten op een antwoord van de toestellen en hebben dus tijdelijk niks te doen. Wanneer de antwoorden kort daarna aankomen, krijg je opnieuw een piek in het CPU-gebruik door de gegevens die moeten verwerkt worden. Tegelijkertijd zie je dat het aantal threads verder afneemt naarmate de laatste toestellen afgewerkt worden.



Figuur 4.8: CPU-gebruik bij verschillend aantal nodes



Figuur 4.9: Aantal retrieverthreads tijdens de uitvoer van de SNMP Data Retriever

In de grafiek met het aantal retrieverthreads zien we trouwens ook een probleem met hoe de threads beheerd worden. Zoals uitgelegd in paragraaf 3.1.3 worden in de oude versie die wij gebruiken de threads in een lijst bijgehouden, en wordt er gewacht op de eerste thread om te voltooien eer er nieuwe threads gestart worden. Het probleem daarbij is dat het niet noodzakelijk de eerst gestarte thread is die eerst klaar zal zijn. Terwijl er gewacht wordt op de eerste thread kunnen er gerust al andere threads klaar zijn en dat zien we ook in de grafiek rond 5 en 10 seconden. Het aantal threads daalt, maar doordat er gewacht wordt op de eerste thread, duurt het even eer er nieuwe threads aangemaakt worden. Dit probleem werd doorgegeven aan NetworkMining en werd vrij snel opgelost in een nieuwe versie.

Conclusie

Doordat er maximaal 50 toestellen tegelijkertijd bevraagd worden blijft het CPU-gebruik ook constant bij het bevragen van meer 50 toestellen. Door het constant houden van het aantal threads zien we echter dat niet alle rekenkracht benut wordt. Alhoewel er 16 cores beschikbaar zijn, er is maar een CPU-belasting die overeenkomt met een volledige belasting van iets meer dan 9 cores.

Het beheren van threads zou dus beter aangepakt kunnen worden: enerzijds bij het bepalen van het aantal threads dat moet gebruikt worden, afhankelijk van hoeveel rekenkracht er nog over is (en het RAM- en bandbreedtegebruik, maar die spelen een veel kleinere rol zoals we verder zullen zien). Anderzijds moeten ook nieuwe threads gestart worden van zodra een toestel afgewerkt is, maar dat is een probleem dat reeds opgelost werd door NetworkMining.

4.3.5 Benchmarks geheugenverbruik

Over het geheugenverbruik valt er weinig te vertellen. We bekijken opnieuw de invloed van het aantal te bevragen toestellen op het geheugengebruik maar we zullen zien dat de conclusies die we gemaakt hebben in de vorige paragraaf hier ook van toepassing zijn.

Meetresultaten

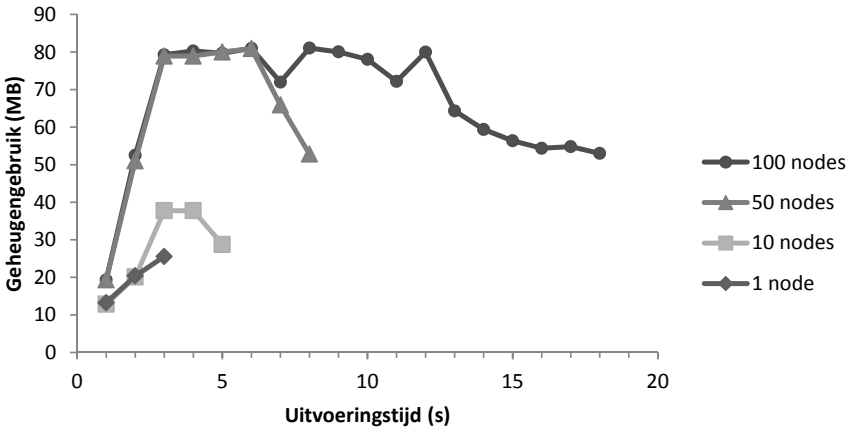
In figuur 4.10 zien we het geheugengebruik van de SNMP Data Retriever bij een verschillend aantal nodes. Net als bij het CPU-gebruik zien we dat het geheugenverbruik meestijgt met het aantal op te vragen toestellen. En ook hier stijgt het geheugengebruik niet verder als er meer dan 50 toestellen moeten bevraagd worden.

Het geheugenverbruik blijft relatief constant gedurende de uitvoer van het programma. Er is een initiële piek wanneer de threads aangemaakt worden en het verbruik zakt pas wanneer het aantal threads daalt. De hoeveelheid geheugen dat gebruikt wordt valt op zich ook zeer goed mee: bij 50 of meer toestellen is er een piekverbruik van ongeveer 80 MB.

Conclusie

Net als het CPU-gebruik stijgt het geheugengebruik niet verder als er meer dan 50 toestellen bevraagd worden. Het geheugengebruik is gedurende de uitvoer vrij constant en met een maximaal verbruik van ongeveer 80 MB kunnen we zeggen dat de SNMP Data Retriever absoluut niet veel geheugen nodig heeft.

Mocht het voorstel om de impact van de databankinteracties te beperken, geïmplementeerd worden (paragraaf 4.3.2), is er duidelijk nog genoeg geheugen vrij om een tabel in het geheugen te houden alvorens ze in de databank weg te schrijven.



Figuur 4.10: Geheugenverbruik bij een verschillend aantal nodes

4.3.6 Benchmarks bandbreedte

In deze paragraaf bekijken we het bandbreedteverbruik van de SNMP Data Retriever. We vergelijken het bandbreedteverbruik van een SNMP walk op basis van GETNEXT- en GETBULK-requests. Ook gaan we na wat de invloed is van het aantal te bevragen toestellen op het bandbreedtegebruik.

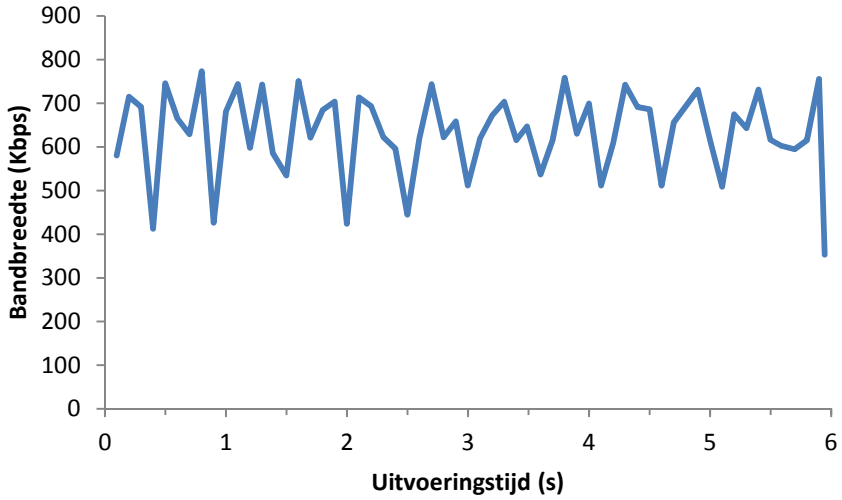
SNMP Walk met GETNEXT- versus GETBULK-requests

In tabel 4.23 staan de netwerkstatistieken voor een SNMP walk op basis van GETNEXT- en GETBULK-requests. Weinig verrassend is dat er met bulkrequests een stuk minder pakketten verstuurd worden dan ontvangen. Vermits we 50 objecten per request opvragen, zijn er ook ongeveer 50 keer minder pakketten die moeten verstuurd worden. Omdat er veel minder pakketten worden verstuurd en ontvangen, worden er natuurlijk ook minder bytes verstuurd en ontvangen. Minder pakketten wil ook zeggen minder overhead door de headers van de pakketten. En omdat er in totaal minder data is verstuurd en ontvangen, was er ook gemiddeld minder bandbreedte nodig.

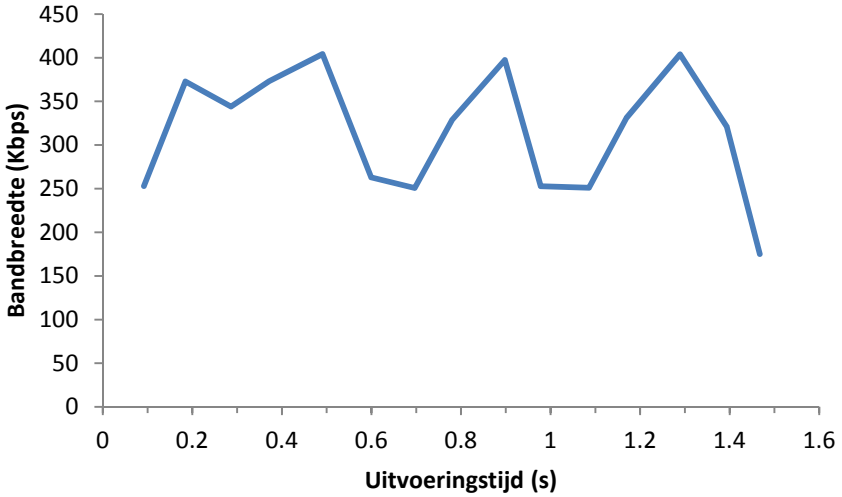
	Walk	Bulkwalk
Pakketten verstuurd	2752	56
Pakketten ontvangen	2752	56
KiloBytes verstuurd	234,86	4,78
KiloBytes ontvangen	239,83	54,25
Gem. KiloBytes per seconde	79,82	40,23
Gem. Megabits per seconde	0,65	0,33

Tabel 4.23: Netwerkstatistieken SNMP walk met GETNEXT- versus GETBULK-requests

We zien dit ook terug in figuren 4.11 en 4.12 die de bandbreedte tonen gedurende de SNMP walk met respectievelijk GETNEXT- en GETBULK-requests. De fluctuatie die je ziet in beide grafieken is trouwens enkel te wijten aan de fluctuatie in responstijd van de SNMP-agent. De verwerkingstijd aan de kant van de retriever, of de tijd die nodig is om een nieuwe request te versturen, is daarentegen zeer consistent. De resultaten werden hierbij echter niet weggeschreven in de databank.



Figuur 4.11: Bandbreedte SNMP walk met GETNEXT-requests



Figuur 4.12: Bandbreedte SNMP walk met GETBULK-requests

Invloed aantal ondervraagde nodes

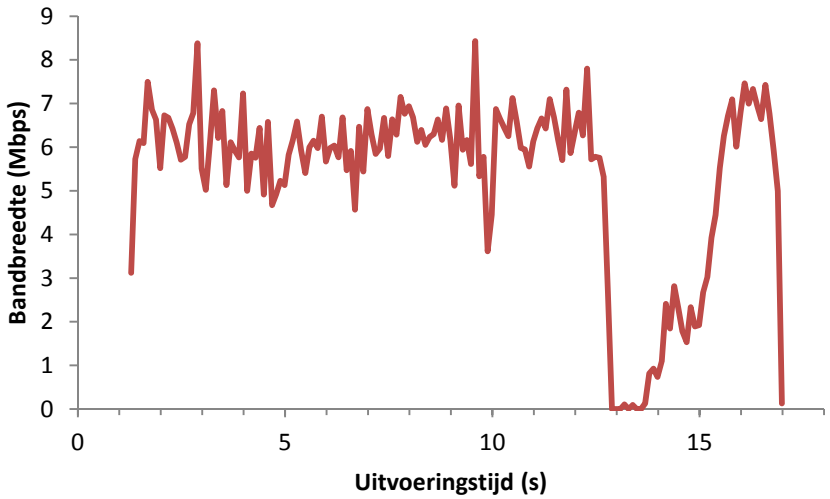
In tabel 4.24 zien we de netwerkstatistieken voor 1, 10, 50 en 100 nodes. Omdat de toestellen quasi gelijk zijn aan elkaar, zien we dat het aantal pakketten en de hoeveelheid data lineair stijgt met het aantal nodes. Bij het gemiddelde bandbreedteverbruik is dat echter niet het geval doordat meer nodes opvragen langer duurt. Ondanks het feit dat de toestellen tegelijkertijd opgevraagd worden, duurt het toch iets langer om meerdere toestellen op te vragen. Laat ons als voorbeeld het verschil tussen een en tien nodes nemen. Er zijn wel tien keer meer toestellen op te vragen, maar er wordt bijna twee keer langer over gedaan. Daardoor stijgt het bandbreedteverbruik dus ongeveer maar met een factor vijf in plaats van met een factor tien.

Het bandbreedteverbruik zou ook niet verder mogen stijgen als er meer dan 50 toestellen bevraagd worden. We zien zelfs dat ze daalt. Als we naar de grafiek van het bandbreedteverbruik bij 100 nodes kijken in figuur 4.13 zien we snel waarom. Ook in het bandbreedteverbruik zien we hetzelfde fenomeen terug als bij het CPU-gebruik in paragraaf 4.3.4: een dip gevolgd door een piek in het verbruik naar het einde toe. Als er gelijktijdig gewacht wordt op antwoorden van de toestellen zal het bandbreedteverbruik op dat moment natuurlijk ook zakken.

	1 node	10 nodes	50 nodes	100 nodes
Pakketten verstuurd	72	648	3507	7093
Pakketten ontvangen	72	648	3507	7093
KiloBytes verstuurd	6,75	60,75	328,10	663,70
KiloBytes ontvangen	97,82	880,44	4.762,68	9.633,34
Gem. KiloBytes per seconde	84,64	398,51	730,84	655,82
Gem. Megabits per seconde	0,69	3,27	5,99	5,37
Tijd* (s)	1,235	2,362	6,966	15,701

* Benodigde tijd enkel voor het SNMP-verkeer

Tabel 4.24: Netwerkstatistieken voor een verschillend aantal nodes



Figuur 4.13: Bandbreedte bij het bevragen van 100 nodes

Vandaar dus dat het gemiddelde bandbreedteverbruik lager uitvalt. De mediaan van het bandbreedteverbruik valt bij 100 nodes op bijna 6 Mbps, wat wel overeenkomt met het verbruik van 50 nodes.

Het bandbreedteverbruik valt op zich ook zeer goed mee, als je rekening houdt met het feit dat netwerkverbindingen meestal 100 Mbps of zelfs al 1 Gbps zijn. In geval van een verbinding van 100 Mbps gaat het om een belasting van 6%, bij een verbinding van 1 Gbps zelfs maar om een belasting van 0,6%.

Conclusie

Naast de uitvoeringstijd is er nog een voordeel bij het gebruik van bulkrequests: zowel het totale dataverkeer als de benodigde bandbreedte vallen een stuk lager uit dan bij GETNEXT-requests.

Net als bij het CPU- en geheugengebruik kunnen we opnieuw vaststellen dat het bandbreedteverbruik niet verder stijgt als er meer dan 50 toestellen bevraagd worden. Ook over het bandbreedteverbruik hoeven we ons dus weinig zorgen te maken. De bevragingen kunnen zelfs 's nachts ingepland worden, zodat er helemaal geen impact is op het netwerkverkeer tijdens de werkuren.

Hoofdstuk 5

Problemen

In dit hoofdstuk bespreken we kort de belangrijkste problemen die we ondervonden hebben tijdens de masterproef. We zullen het hebben over randgevallen die nog niet voorzien waren door de SNMP Data Retriever, de problemen op het iMinds netwerk en problemen bij het toepassen van virtualisatie op de Virtual Wall.

5.1 Ondersteuning voor lange OID's

Een probleem waar we al redelijk vroeg op stootten, was het optreden van SQL-fouten bij het wegschrijven van gegevens met een lange OID in de databank. De grote boosdoeners hier waren de zeer lange indexen van de tabel *inetCidrRouteTable* (1.3.6.1.2.1.4.24.7).

```
inetCidrRouteEntry OBJECT-TYPE
     SYNTAX InetCidrRouteEntry
     MAX-ACCESS not-accessible
     STATUS current
     DESCRIPTION "A particular route to a particular destination, under a
            particular policy (as reflected in the
            inetCidrRoutePolicy object).

            Dynamically created rows will survive an agent reboot.

            Implementers need to be aware that if the total number
            of elements (octets or sub-identifiers) in
            inetCidrRouteDest, inetCidrRoutePolicy, and
            inetCidrRouteNextHop exceeds 111, then OIDs of column
            instances in this table will have more than 128 sub-
            identifiers and cannot be accessed using SNMPv1,
            SNMPv2c, or SNMPv3."
     INDEX { inetCidrRouteDestType, inetCidrRouteDest, inetCidrRoutePfxLen, inetCidrRoutePolicy,
          inetCidrRouteNextHopType, inetCidrRouteNextHop }
    ::= { inetCidrRouteTable 1  }
```

Codefragment 5.1: Definitie van een inetCidrRouteEntry

Codefragment 5.1 laat de definitie zien van een index uit die tabel en toont dat ze bestaat uit een samenstelling van maar liefst zes velden. Een van die velden is een IP-adres, dat een IPv4- of een IPv6-adres kan zijn. Zoals je weet zijn IPv6-adressen echter vrij lang, 128 bits om precies te zijn. Een voorbeeld van een OID uit de inetCidrRouteTable die een IPv6-adres bevat zie je in codefragment 5.2.

```
IP-FORWARD-MIB::inetCidrRouteIfIndex.ipv6."fe:80:00:00:00:00:00:0a:00:27:ff:fe:6d:bd:c5"
    .128.1.7.ipv6."00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00" = INTEGER: 1

1.3.6.1.2.1.4.24.7.1.7.2.16.254.128.0.0.0.0.0.10.0.39.255.254.109.189.197
    .128.1.7.2.16.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0 = INTEGER: 1
```

Codefragment 5.2: Tekstuele en numerieke notatie van een OID uit inetCidrRouteTable

Omdat de resultaattabel die de SNMP Data Retriever aanmaakte slechts 55 karakters voorzag voor een OID, resulteerde dit in SQL-fouten bij OID's die te lang waren, zoals ook te zien is in codefragment 5.3.

```
2014/02/24 14:28:54.92  Error       [21/311MB]  Failed to insert result row:
ComfunSQLConnectionException: Error executing SQL statement: 'replace into results
     values(@devname,'1.3.6.1.2.1.4.34.1.5.2.16.254.128.0.0.0.0.0.10.0.39.255.254.109.189.197',
     @attrname,@value,'2014-02-24 14:28:54','OK','2014-02-24 14:28:54')' with parameters
     '@attrname RFC1213-MIB.ip System.String @devname debian-vm-01 System.String @value
     1.3.6.1.2.1.4.32.1.5.2.2.16.254.128.0.0.0.0.0.0.0.0.0.0.0.0.64 System.String' using
     connectionstring 'database=snmpdb;data source=localhost;user
     id=xxxxx;password=xxxxx;port=3306;old syntax=yes'. --->
     MySql.Data.MySqlClient.MySqlException: #22001Data too long for column 'OID' at row 1
```

Codefragment 5.3: SQL-fout bij te lange OID's

De oplossing is gelukkig eenvoudig: met het vergroten van de toegelaten lengte van de OID-kolom in de resultaattabel is het probleem opgelost. Om dat te doen passen we simpelweg de SQL-query aan in de SNMP Data Retriever die de tabel aanmaakt. We nemen een grote marge voor de kolombreedte en vergroten ze naar 4096 karakters om toekomstige problemen te vermijden, en omdat brede kolommen toch geen grote impact hebben.

Reeds bestaande tabellen moeten echter ofwel manueel aangepast worden om de kolom te verbreden, ofwel kan men de tabel verwijderen en de SNMP Data Retriever een nieuwe laten aanmaken indien de reeds opgeslagen resultaten verwijderd mogen worden.

5.2 Ondersteuning voor de endOfMibView-exceptie

Zoals we gezien hebben in de berichtstructuur van een SNMP-bericht in paragraaf 2.9, zijn er twee velden voorzien in een SNMP-bericht om aan te geven dat er zich een fout heeft voorgedaan: *Error* en *Error Index*. Het eerste veld geeft met een statuscode aan of er zich een fout heeft voorgedaan, en zoja, dewelke. Het tweede veld bevat in geval van een fout een wijzer naar het object dat de fout veroorzaakt heeft

Een van de nieuwe dingen die SNMPv2c introduceerde zijn SNMP-excepties. Er bestaan drie zulke excepties: *EndOfMibView*, *NoSuchInstance* en *NoSuchObject*. Het bijzondere aan deze SNMP-excepties is dat ze toegevoegd worden aan de objectlijst (Varbind List) van een bericht en het errorveld niet wijzigen als ze zich voordoen[9]. Een SNMP-bericht waarin een fout is opgetreden dat aangeduid wordt door een exceptie, heeft dus nog steeds nul als waarde voor het errorveld, wat aangeeft dat er geen fout zou opgetreden zijn.

De reden daarvoor is om het mogelijk te maken dat een exceptie zich kan voordoen samen met andere objecten die wel correct opgehaald zijn. Herinner je dat je met GET- en GETNEXT-requests meerdere gegevens kunt ophalen, maar dat als er zich een fout voordoet je helemaal geen gegevens terug zal krijgen met SNMPv1 (paragraaf 2.7.2). Met SNMPv2c werd dit probleem dus met SNMP-excepties opgelost.

De endOfMibView-exceptie doet zich voor bij een SNMP walk wanneer er geen verdere data meer op te halen is[1][21]. Wij kwamen deze exceptie tegen bij het overlopen van een ganse SNMP-agent met de SNMP Data Retriever, waarbij er op het einde natuurlijk geen gegevens meer zijn en de exceptie opgegooid wordt.

Omdat de originele versie van de SNMP Data Retriever enkel gebruik maakte van GET- en GETNEXT-requests was SNMPv1 voldoende. Bovendien is het overlopen van een ganse SNMP-agent vrij uitzonderlijk. Men kwam dus nooit in aanraking met deze (en de andere) excepties. Bijgevolg werd er voor het detecteren van fouten enkel gekeken naar het errorveld en werd er geen rekening gehouden met excepties.

In de originele versie had dit geen gevolgen omdat excepties zich toch niet voordoen in SNMPv1. Bij de nieuwe versie zal hier echter wel op moeten gecontroleerd worden vermits er voor bulkrequests gebruik gemaakt moet worden van SNMPv2c.

De manier om dit aan te pakken is vrij eenvoudig. Men controleert net zoals voorheen nog steeds het errorveld, maar bij het overlopen van de objecten in het antwoordbericht zal men ook moeten controleren of er een exceptie aanwezig is en de nodige acties ondernemen. Bij een endOfMibView-exceptie zal men de SNMP walk moeten staken.

5.3 "DoS-bescherming" op intern netwerk iMinds

Een van de problemen waar we veel hinder van ondervonden hebben was een schijnbare Denial of Service (DoS) bescherming op het netwerk van iMinds. Een DoS-aanval probeert een systeem buiten werking te stellen door bijvoorbeeld een server te overrompelen met een zo groot mogelijke hoeveelheid data. Het gaat echter slechts om een vermoeden want de precieze oorzaak hebben we nooit kunnen vaststellen.

Het probleem doet zich voor als we een toestel bevragen in of via het iMinds netwerk. Dit gebeurde zowel bij het bevragen van de iMinds productieswitches als bij toestellen op de Virtual Wall. Wanneer we op korte tijd "te veel" verkeer genereerden — de juiste hoeveelheid of de tijdspanne hebben we niet kunnen vaststellen —, werd alle verkeer geblokkeerd gedurende exact 20 seconden. Het feit dat er iedere keer exact 20 seconden geen verkeer doorkon is een sterke aanwijzing dat een of andere netwerkbescherming in actie treedt. In figuur 5.1 zie je een bandbreedtegrafiek die het fenomeen toont.

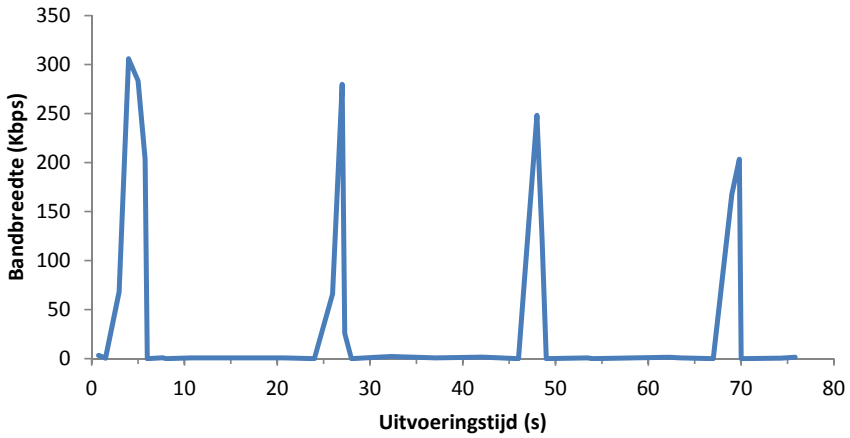
De omstandigheden waarbij de bescherming in werking trad konden we niet bepalen. Er was geen bandbreedtedrempel of een limiet op het aantal pakketten per seconde die overschreden werd. Het probleem deed zich voor bij de SNMP Data Retriever, maar niet bij een Net-SNMP bulkwalk, en dan weer wel bij onze eigen SNMP walkimplementatie, die tabellen rij per rij overloopt en daarvoor Net-SNMP GETBULK-requests gebruikt.

We zijn ook bij de netwerkbeheerders van iMinds ten rade gegaan, maar zij hadden geen weet van wat er verantwoordelijk zou kunnen zijn hiervoor.

Bij de kleinschalige tests die gebruik maakten van de productieswitches konden we wel nog resultaten bekomen als we zeer weinig iteraties uitvoerden. Op die manier konden we toch nog een idee krijgen of de tests op de virtuele machines een realistisch beeld gaven van de werkelijkheid.

Bij de grootschalige tests was het probleem eenvoudig opgelost door het toestel waarop de SNMP Data Retriever draaide ook in de Virtual Wall te plaatsen. Door een directe verbinding te leggen tussen de retriever en de toestellen ging het verkeer niet meer over het iMinds netwerk maar bleef het geïsoleerd in de Virtual Wall waarop geen netwerkbeperkingen van toepassing zijn.

[1] Zowel bij het gebruik van GETNEXT- en GETBULK-requests, indien gebruik gemaakt wordt van SNMPv2c of hoger. Bij SNMPv1 met GETNEXT-requests wordt de foutcode *noSuchName* gebruikt om aan te geven dat er geen data meer is[5].



Figuur 5.1: Bandbreedte tijdens het uitvoeren van tests op het iMindsnetwerk

5.4 Virtualisatie op de Virtual Wall

Een netwerkopstelling op de Virtual Wall wordt geconfigureerd aan de hand van een script dat de *NS-file* genoemd wordt en waarin men de machines en de netwerkverbindingen definieert.

Virtualisatie wordt ondersteund op de Virtual Wall, maar is volgens de documentatie nog volop in ontwikkeling. Men kan hiervoor gebruik maken van OpenVZ of Xen[27, 29]. Om hiervan gebruik te maken volstaat het om andere parameters mee te geven met de gedefinieerde machines in de NS-file, zodat aangeven wordt dat het om virtuele machines gaat. De Virtual Wall zorgt er dan voor dat de nodige hostmachines voorzien worden en de gevraagde toestellen gevirtualiseerd worden.

De problemen die wij ondervonden, hadden te maken met compatibiliteitsproblemen met de netwerkencapsulatiemethoden. Er zijn twee zulke encapsulatiemethoden: *veth-ne* en *vlan*. Standaard wordt de eerste gebruikt, maar men kan er ook voor kiezen om de tweede te gebruiken. Documentatie hierover is echter zo goed als niet te vinden.

Om meerdere toestellen allemaal met elkaar te verbinden definieert men in de NS-file een LAN-netwerk en voegt men daar alle toestellen aan toe. In de praktijk wordt dat dan vertaald door een switch waarmee alle toestellen verbonden zijn. Het probleem is dat, als men de standaardencapsulatie gebruikt, de virtuele machines daar niet mee overweg kunnen. De foutmelding die gegeven wordt bij het opzetten van de netwerkopstelling ziet er zo uit:

```
Encapsulation not supported on big-lan since at least one of the nodes in big-lan does not
    support 'veth-ne' link emulation
```

Hierbij werd het LAN-netwerk *'big-lan'* genoemd. Als men de vlan-encapsulatiemethode gebruikt, kunnen de niet-gevirtualiseerde machines daar op hun beurt niet mee overweg. Het komt er dus op neer dat we alle virtuele machines wel met elkaar kunnen verbinden maar niet met fysieke machines, ondanks dat dit volgens de documentatie wel mogelijk zou moeten zijn, zelfs zonder de encapsulatiemethode te wijzigen[27].

De oplossing die wij dan uiteindelijk hebben gebruikt, was om, in plaats van de virtualisatie over te laten aan de Virtual Wall, dit zelf te configureren. Sahel Sahhaf heeft ons hierbij op weg geholpen door ons te voorzien van een werkende opstelling die gebruik maakte van LXC (zie paragraaf 4.3.1).

Hoofdstuk 6

Voorstellen voor verdere optimalisaties

In dit hoofdstuk bespreken we de verdere optimalisaties die nog kunnen (en in één geval moeten) gebeuren om de SNMP Data Retriever op grote schaal te kunnen inzetten. De belangrijkste optimalisatie wellicht die nog moet gebeuren is het gebruik van bulk inserts bij het wegschrijven van de opgehaalde resultaten in de databank. Ook bij het beheer van de retrieverthreads is er nog ruimte voor verbetering, en op lange termijn kan men ook denken aan het inbouwen van intelligentie die bij het opvragen rekening houdt met de veranderlijkheid van gegevens.

6.1 Bulk inserts bij databankinteracties

De databankinteracties werden uitvoerig besproken in de paragrafen 3.1.3, 4.2.7 en 4.3.2. Een verdere optimalisatie die zeker nog gedaan moet worden, is het gebruik van bulk inserts bij het wegschrijven van de opgehaalde gegevens in de databank.

In paragraaf 4.3.2 hebben we gezien dat het wegschrijven van de gegevens in de databank een gigantische impact heeft op de uitvoeringstijd, die alleen maar toeneemt met het aantal te bevragen toestellen. Het bevragen van slechts 100 toestellen ging zonder het wegschrijven van de resultaten maar liefst 61 keer sneller. Sterker nog, de impact van de databankinteracties is zodanig groot dat de performantiewinst van de andere optimalisaties bijna teniet gedaan wordt. De nieuwe versie was bij 100 toestellen slechts 5% sneller dan de oude, maar ruim zeven keer sneller als de gegevens niet weggeschreven moesten worden. Het implementeren van bulk inserts is dus absoluut cruciaal om de SNMP Data Retriever succesvol te kunnen inzetten op grote schaal.

Het implementeren van bulkrequests hoeft zelfs niet eens zoveel tijd en werk te vereisen. Een eenvoudige implementatie houdt een tabel bij in het geheugen die de resultaten tijdelijk opslaat. De InsertResultRow-methode moet dan aangepast worden om de gegevens in de tabel in het geheugen weg te schrijven in plaats van rechtstreeks naar de databank. Eenmaal een bepaalde hoeveelheid gegevens in de tabel in het geheugen zitten, kan de InsertResultRow-methode beslissen om al die data in een keer weg te schrijven naar de databank met een bulk insert.

Het nadeel van deze manier van werken is dat het wegschrijven van de resultaten in dezelfde thread gebeurt als het bevragen van een toestel. Als de gegevens weggeschreven worden zal die thread en het bevragen van dat ene toestel een stuk langer duren dan anders.

Een andere manier van werken die wat meer tijd en werk vraagt om te implementeren werd ook al voorgesteld in paragraaf 4.3.2. Daarbij wordt de verantwoordelijkheid van het wegschrijven

van de resultaten in de databank overgeheveld naar een aparte thread. Deze zal dan instaan om, wanneer het aantal gegevens in de tabel in het geheugen een bepaalde drempelwaarde overschrijdt, de gegevens weg te schrijven naar de databank, zonder de andere retrieverthreads daarbij te storen. De InsertResultRow-methode is dan enkel nog verantwoordelijk voor het wegschrijven van de gegevens in de tabel in het geheugen en heeft een minimale impact op de bevraging.

Men kan vervolgens nagaan of het beter is om periodiek de gegevens weg te schrijven tijdens het bevragen van de toestellen, of om de gegevens ineens weg te schrijven eens alle toestellen bevraagd zijn, of iets vroeger wanneer het aantal retrieverthreads begint te dalen.

De gegevens hoeven zelfs niet weggeschreven te worden in een databank. Men zou verschillende implementaties kunnen voorzien, die de gegevens in de tabel in het geheugen op verschillende manieren verwerken. Men kan er bijvoorbeeld voor kiezen om de gegevens naar een bestand weg te schrijven of zelfs naar beide. Het grote voordeel is dat het opslaan van de gegevens los staat van het opvragen ervan, zodat de snelheidsimpact ervan beperkt blijft.

6.2 Dynamisch threadbeheer

Het beheer van threads en het CPU-gebruik werden besproken in paragrafen 3.1.3 en 4.3.4.

Er was al gebleken dat het beheer van de threads voor het bevragen van de toestellen in de originele versie van de SNMP Data Retriever beter kon. Het probleem was dat er gewacht werd tot de eerst gestarte thread klaar was alvorens er nieuwe threads werden aangemaakt voor de volgende toestellen. Dat leidde tot korte periodes waarbij het aantal retrieverthreads zakte en er geen nieuwe threads werden aangemaakt. Maar zoals gezegd werd dit probleem reeds aangepakt door NetwerkMining tijdens de masterproef en wordt er nu gebruik gemaakt van een andere implementatie in nieuwere versies.

We hebben ook gezien dat het aantal threads vaststond op 50 threads. Een betere manier van werken zou het aantal threads dynamisch bepalen aan de hand van de beschikbare resources, zowel van CPU-, RAM- als bandbreedtegebruik. Uit de tests in paragrafen 4.3.5 en 4.3.6 bleek echter dat het geheugen- en bandbreedtegebruik beperkt blijft en er dus meer dan genoeg reserve over is wat betreft geheugen en bandbreedte. Het aantal threads zou dus voornamelijk afhangen van het CPU-gebruik.

Op de server van de Virtual Wall met de retriever was er nog een overschot aan CPU-rekenkracht, maar het omgekeerde is ook mogelijk. Bij een tekort aan rekenkracht heeft het geen zin om meer threads aan te maken en kan het mogelijk zelfs een averechts effect hebben.

Het dynamisch beheren van threads is een algemeen probleem waar veel softwareprojecten mee geconfronteerd worden. Gelukkig betekent dat ook dat er reeds vele oplossingen voor zijn ontwikkeld die vrij te gebruiken zijn. Het .NET-platform biedt hiervoor sinds versie 4 een eigen oplossing aan onder de vorm van de Task Parallel Library (TPL).

Het doel van TPL is om het ontwikkelaars gemakkelijker te maken om van parallelisme en multithreading in applicaties gebruik te maken. Om zo efficiënt mogelijk gebruik te maken van alle processorcores die beschikbaar zijn, schaalt TPL dynamisch de mate van parallelisme. TPL verzorgt ook het inplannen van threads in een *thread pool*, de ondersteuning voor het annuleren van threads, toestandsbeheer en andere low-level details[23].

6.3 Intelligentie

Op langere termijn kan de SNMP Data Retriever voorzien worden van enige intelligentie bij het opvragen van netwerkinformatie. Dit is vooral van toepassing als er historische data bijgehouden

wordt. Sommige gegevens zijn namelijk zeer dynamisch, terwijl andere zo goed als nooit veranderen. Denk bijvoorbeeld aan temperatuurmetingen en een overzicht van alle netwerkinterfaces die aanwezig zijn in een toestel. Het eerste gegeven verandert constant, het laatste haast nooit. Het is dan ook logisch dat het laatste niet zo vaak opgevraagd moet worden als het eerste. Zo zou een algoritme of heuristiek kunnen ontwikkeld worden die de veranderlijkheid van gegevens bepaalt en afhankelijk daarvan beslist hoe vaak die gegevens opgehaald moeten worden. Gegevens kunnen ook meer of minder belangrijk zijn dan andere. Afhankelijk van het belang dat een netwerkbeheerder aan gegevens hecht, kan hij dan de frequentie waarmee bepaalde gegevens moeten opgevraagd worden verhogen of verlagen.

Conclusie

Bij de aanvang van de masterproef werd de SNMP Data Retriever enkel ingezet op kleine- en middelgrote netwerken. We hebben de belangrijkste aspecten van het retrievalproces doorgelicht en hebben daarbij de belangrijkste problemen blootgelegd die het gebruik van de retriever op grootschalige netwerken belemmerden. We hebben de nodige aanpasingen aangebracht aan de retriever om die problemen op te lossen, waaronder het gebruik van bulkrequests en de implementatie van een alternatief loggingframework. Met die aanpassingen konden we besluiten dat de schaalbaarheid van de applicatie er sterk op vooruit is gegaan.

Er is echter nog een cruciaal probleem dat opgelost moet worden alvorens we de SNMP Data Retriever kunnen aanbevelen voor het gebruik ervan op grootschalige netwerken: de databankinteracties. We hebben verschillende manieren besproken waarop dit probleem aangepakt kan worden, en de eenvoudigste zijn gelukkig niet al te ingrijpend en hoeven niet al te veel tijd in beslag te nemen.

Zijn we geslaagd in onze opzet om de schaalbaarheid te onderzoeken en te verbeteren? Zeker, alhoewel er wel nog wat werk te doen is op het gebied van databankinteracties. Als we niet vertraagd zouden worden door de databankinteracties, dan hebben we de SNMP Data Retriever ruim zeven keer sneller gemaakt bij het bevragen van 100 toestellen. Een theoretische oefening op de uitvoeringstijd bij nog grotere aantallen toestellen leerde ons dat de impact van extra te bevragen toestellen beperkt genoeg blijft om de retriever te kunnen inzetten op zeer grote netwerken.

Ook het geheugen- en bandbreedtegebruik blijft zeer minimaal bij het bevragen van een groot aantal toestellen. Bij het CPU-gebruik konden we wel nog opmerken dat er bij het beheren van het aantal threads nog verbetering mogelijk is.

Bibliografie

[1] *Address Resolution Protocol - Wikipedia, the free encyclopedia*. URL: https://nl.wikipedia.org/wiki/Address_Resolution_Protocol.

[2] Douglas Bruey. *SNMP: Simple? Network Management Protocol*. Rane Corporation. Dec 2005. URL: http://www.rane.com/note161.html.

[3] *Comparison of logging frameworks - Essential Diagnostics*. URL: https://essentialdiagnostics.codeplex.com/wikipage?title=Comparison.

[4] *Comparison of .NET Logging Frameworks and Libraries*. URL: http://www.dotnetlogging.com/comparison/.

[5] B. Wijnen en D. Levi. *RFC 2089 - V2ToV1 Mapping SNMPv2 onto SNMPv1 within a bi-lingual SNMP agent*. Jan 1997. URL: https://tools.ietf.org/html/rfc2089.

[6] *De Debian Richtlijnen voor Vrije Software (DFSG)*. URL: https://www.debian.org/social_contract.nl.html#guidelines.

[7] *Debian Free Software Guidelines - Wikipedia, the free encyclopedia*. URL: https://nl.wikipedia.org/wiki/Debian_Free_Software_Guidelines.

[8] Michael Groeger. *An Introduction To Performance Counters - CodeProject*. 2005. URL: http://www.codeproject.com/Articles/8590/An-Introduction-To-Performance-Counters.

[9] Wes Hardaker. *SNMPv3 endOfMibView - Stack Overflow*. 20 dec 2011. URL: https://stackoverflow.com/questions/8569554/snmpv3-endofmibview/8570559#8570559.

[10] *Holes in Tables - snmptable tutorial - Net-SNMP Wiki*. URL: http://www.net-snmp.org/wiki/index.php/TUT:snmptable#Holes_in_Tables.

[11] *IPv4 Header - Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/IPv4#Header.

[12] Douglas R. Mauro en Kevin J. Schmidt. *Essential SNMP*. 2de ed. O'Reilly Media, 2005. ISBN: 978-0596008406.

[13] *Latency (engineering) - Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Latency_(engineering)#Communication_latency.

[14] Naman Latif. *How many bytes are in an ethernet header - The Cisco Learning Network*. 19 aug 2009. URL: https://learningnetwork.cisco.com/message/30662#30663.

[15] *LXC - Linux Containers - Add new network interface without restarting - Stack Overflow*. 21 apr 2014. URL: https://stackoverflow.com/questions/22780927/lxc-linux-containers-add-new-network-interface-without-restarting/23204991#23204991.

[16] *LXC - Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/LXC.

[17] Joris Moreau. *Computernetwerken III: netwerkbeheer*. Hfdstk. SNMP.

[18] *Net-SNMP History*. URL: http://www.net-snmp.org/about/history.html.

[19] David T. Perkins. *SNMPinfo: SNMP Versions*. URL: http://www.snmpinfo.com/versions.htm.

[20] *Simple Network Management Protocol - Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Snmp.

[21] *SNMP v2c and v3 EndOfMibView, NoSuchInstance and NoSuchObject types | SnmpSharpNet*. URL: http://www.snmpsharpnet.com/?p=139.

[22] *SNMP Version 3 (SNMPv3)*. URL: http://www.ibr.cs.tu-bs.de/projects/snmpv3/.

[23] *Task Parallel Library (TPL)*. URL: http://msdn.microsoft.com/en-us/library/dd460717.aspx.

[24] *Technische tests - iMinds*. URL: http://www.iminds.be/nl/digitaal-onderzoek/technische-testing.

[25] *Virtual Wall - iLab.t testbeds 1.0.0 documentation*. URL: http://doc.ilabt.iminds.be/ilabt-documentation/virtualwallfacility.html.

[26] *Virtualization - What is LXC and how to get started? - Ask Ubuntu*. URL: https://askubuntu.com/questions/293275/what-is-lxc-and-how-to-get-started.

[27] *Vnodes - Emulab*. URL: https://wiki.emulab.net/Emulab/wiki/vnodes.

[28] *What's New in VirtualBox 4.3?* 15 okt 2013. URL: https://blogs.oracle.com/fatbloke/entry/what_s_new_in_virtualbox.

[29] *Xen - Emulab*. URL: https://wiki.emulab.net/wiki/xen.