Studienarbeit

SNMP in VNUML Simulationen

Frank Bohdanowicz
Juli 2006

Universität Koblenz-Landau
Abteilung Koblenz

Fachbereich Informatik
AG Rechnernetze und Rechnerarchitektur

Betreuer:
Prof. Dr. Christoph Steigner
Dipl. Inf. Harald Dickel

Hiermit versichere ich, dass ich die vorliegende Studienarbeit selbständig verfasst und keine anderen als die von mir angegebenen Quellen und Hilfsmittel verwendet habe.

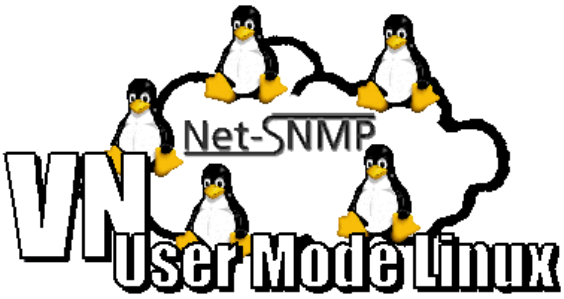
Koblenz, den 30.7.2006

_____

Frank Bohdanowicz

Zusammenfassung der Studienarbeit

Das Simple Network Management Protocol (SNMP) gilt als die Universalsprache des Netzwerkmanagements. Bereits Anfang der 90er Jahre wurde die erste Version von SNMP durch die Internet Engineering Task Force (IETF) zum Standard-Internet Management Protocol erklärt und Teil der TCP/IP Protocol-Suite. Für die meisten Betriebssystemplattformen sind SNMP Implementierungen verfügbar und viele netzwerkfähige Geräte und Netzwerkmanagement-Programme unterstützen SNMP, das sich vor allem für die Verwaltung plattformübergreifender und herstellerunabhängiger Netzwerke bewährt.
Virtual Network User Mode Linux (VNUML) ist ein mächtiges Netzwerk-Simulationsprogramm für Linux mit dem virtuelle Linux-Rechnernetze aufgebaut werden können, um darin Programmabläufe zu simulieren. Die VNUML Netzwerk-Simulationen sind in erster Linie für das Entwickeln, Analysieren und Testen von Linux Netzwerk-Software, wie zum Beispiel Netzwerk-Protokollen, geeignet. Das Simulationsprogramm entstand im Rahmen des Euro6IX-Projektes, zur Einführung des IPv6 Standards in Europa, am Telematics Engineering Department der Technischen Universität Madrid. Die Rechner der virtuellen Netze, die VNUML aufbaut, basieren auf User Mode Linux (UML), einer in breitem Spektrum eingesetzten Virtualisierung des Linux-Kernels.
Diese Studienarbeit beschäftigt sich mit den Möglichkeiten und der Funktionsweise des Netzwerkmanagements mit SNMP. Dafür wird die SNMP-Software Net-SNMP in VNUML Simulationen eingesetzt, um die Möglichkeiten der Konfiguration und des Umgangs mit SNMP in einer praxisnahen Umgebung zu untersuchen.
Der Einsatz von Net-SNMP in VNUML Simulationen kann dazu dienen, die Integration von Netzwerk-management mit SNMP für relevante Rechnernetze vorzubereiten und Möglichkeiten der Konfiguration und der Verwendung auszuloten oder VNUML Simulationen im Allgemeinen mit diesem bewährten Netzwerkmanagement-System zur Unterstützung auszustatten.



Inhaltsverzeichnis

1 **Einleitung** 6
2 **Aufbau der Studienarbeit** 7

3 **Grundlagen des Netzwerkmanagements** 8
3.1 Einleitung 8
3.2 Die Funktionsbereiche des Netzwerkmanagements 9
3.2.1 Fehlermanagement 9
3.2.2 Konfigurationsmanagement 10
3.2.3 Abrechnungsmanagement 10
3.2.4 Leistungsmanagement 10
3.2.5 Sicherheitsmanagement 11
3.3 Zusammenfassung 11

4 **SNMP-Grundlagen** 12
4.1 Einleitung 12
4.2 SNMP in den RFCs 14
4.3 Das SNMP Konzept 17
4.3.1 Der Manager 18
4.3.2 Der Agent 19
4.3.3 Die Managementinformationen 20
4.3.3.1 ASN.1 und BER 21
4.3.3.2 Structure of Management Information 26
4.3.3.3 Management Information Base 32
4.3.4 Das Netzwerkmanagement-Protokoll SNMP 33
4.4 Sicherheit in SNMP 39
4.4.1 SNMPv1 und SNMPv2c 39
4.4.2 SNMPv3 39
4.4.2.1 Das User-Based Security Model 39
4.4.2.2 Das View-Based Access Control Model 40
4.5 Erweiterungsmodelle für den SNMP-Agenten 41
4.5.1 SMUX 41
4.5.2 AgentX 41
4.6 Zusammenfassung 41

5 **Netzwerk-Simulationen mit Virtual Network User Mode Linux** 42
5.1 Einleitung 42
5.2 VNUML-Grundlagen 42
5.3 Durchführung einer VNUML Simulation 45
5.3.1 Entwurfs Phase 46
5.3.2 Implementierungs Phase 47
5.3.3 Ausführungs Phase 50
5.4 Das VNUML Management-Netzwerk 51
5.5 User Mode Linux Rechner 52
5.6 Hinweise zur Installation von Software in den UML-Rechner 60
5.7 Die Routing-Software Quagga 63
5.7.1 Installation 64
5.7.2 Beispiele für Konfiguration und Umgang mit Quagga 66
5.8 Zusammenfassung 69

6 **Net-SNMP in VNUML Simulationen** 70
6.1 Einleitung 70
6.2 Entwicklung und geschichtlicher Hintergrund zu Net-SNMP 70
6.3 Installation 71
6.4 Der Net-SNMP Manager 74
6.4.1 Konfiguration des Net-SNMP Managers 74
6.4.2 Anwendungsprogramme des Net-SNMP Managers 75
6.4.3 Der Net-SNMP Trap-Server 80
6.5 Der Net-SNMP Agent 82
6.5.1 Konfiguration des Net-SNMP Agenten 82
6.5.1.1 Konfiguration von Zugriffsrechten 82
6.5.1.2 Konfiguration von Managementinformationen 85
6.5.1.3 Konfiguration der Objekt-Überwachung 86
6.5.1.4 Konfiguration von Erweiterungen 88
6.5.2 Die Anwendungsprogramme des Net-SNMP Agenten 91
6.5.3 Erstellen und Einbinden eigener MIB-Module 92
6.6 Anwendungsbeispiele 100
6.7 Zusammenfassung 105

Anhang

Anleitung:

7 **Erstellen eines VNUML-Rechners auf Basis von Gentoo-Linux** 106
7.1 Einleitung 106
7.2 Erstellen eines UML-Root-Dateisystems auf Basis von Gentoo-Linux 106
7.2.1 Vorbereiten des Gentoo-Systems 107
7.2.2 Konfiguration des Gentoo-Systems für VNUML 107
7.2.3 Installation benötigter Software 114
7.2.4 Erstellen der Imagedatei 117
7.3 Erstellen eines UML-Kernels 119
7.4 Starten des UML-Rechners 120
7.5 Zusammenfassung 120

Abbildungsverzeichnis 121
Tabellenverzeichnis 121
Listingverzeichnis 122
Befehlsverzeichnis 123

Literaturverzeichnis 126

1 Einleitung

Virtual Network User Mode Linux (VNUML) bietet die Möglichkeit sehr umfangreiche und detaillierte Simulationen mit virtuellen Linux-Rechnernetzen auf einem Host-Computer durchzuführen. Die Rechner der virtuellen Netzwerke basieren auf User Mode Linux (UML), einer bewährten und in breitem Spektrum eingesetzten Variante des Linux-Kernels, die das Emulieren kompletter Linux-Rechner ermöglicht. Diese virtuellen Linux-Rechner sind in ihrer Funktionalität von reellen Linux-Rechnern kaum zu unterscheiden.
VNUML ermöglicht mit den Simulationen von Rechnernetzen hauptsächlich das Durchspielen netzwerkbelastender Programmabläufe sowie die Entwicklung und Analyse netzwerk-basierter Software. Nicht berücksichtigt wird eine Simulation der physikalischen Beanspruchung der Netzwerkverbindungen. Diese stellen unter VNUML lediglich idealisierte Verbindungen dar, ohne Latenz oder anderer Merkmale echter Netzwerkleitungen, und sind vielmehr von der Leistungsfähigkeit des Host-Computers abhängig. Über eine VNUML eigene XML-Sprache werden die Topologien der zu simulierenden Rechnernetze in Szenarios beschrieben, die VNUML dann für den Aufbau und die Verwaltung einer Simulation verwendet. Dabei generiert VNUML die virtuellen Linux-Rechner des Netzwerk-Szenarios mittels UML, teilt ihnen die definierten IP-Adressen zu und stellt die festgelegten Netzwerkverbindungen her. Durch das Starten verschiedener Anwendungen innerhalb der verschiedenen UML-Rechner können diese unterschiedliche Funktionen im simulierten Netzwerk einnehmen.
Für das dynamische Routing in den virtuellen Netzwerken empfehlen die VNUML-Autoren die freie Routing-Software Quagga. Mittels Quagga können Netzwerke aufgebaut werden in denen Routing-Protokolle wie BGP, RIP oder OSPF eingesetzt werden.
VNUML lässt derzeit nur eine bedingte Verwaltung der Simulation zu. So können in den Szenarios Kommandos definiert werden, die das Kopieren von Dateien auf einzelne UML-Rechner oder das Starten von Anwendungen innerhalb einzelner UML-Rechner einer Simulation veranlassen. Ein detailreiches Management der einzelnen virtuellen Rechner einer Simulation, zum Beispiel um Ereignisse zu protokollieren oder Zustände zu messen, bietet VNUML selbst derzeit nicht.
Seit längerem wird in vielen Computer-Netzwerken das Simple Network Management Protocol (SNMP) zum Verwalten der verschiedensten Netzwerk-Komponenten verwendet. SNMP ist das Standard-Internet Management Protocol der Internet Engineering Task Force (IETF) zur Verwaltung von Netzwerken und gehört zur TCP/IP-Protocol-Suite. Bereits in den frühen 90er Jahren wurde die erste Version von SNMP durch die IETF zum Standard erklärt, woraufhin sich seine Unterstützung durch netzwerkfähige Geräte und Netzwerkmanagement-Produkte recht schnell verbreitete. Mittlerweile ist SNMP in der Version drei Standard bei der vor allem die Sicherheit und Modularität gegenüber den beiden Vorgängern verbessert wurde. Die Vorteile von SNMP sind, neben den generellen Vorzügen eines offenen Standards, die Plattform- und Herstellerunabhängigkeit, die sehr weitverbreitete Unterstützung durch zahlreiche Netzwerk-Produkte und die Möglichkeit, recht schnell und unkompliziert neue Ressourcen zur Verwaltung in den SNMP-Managementprozess einzubinden.
Auch für das Betriebssystem Linux liegen einige Implementierungen der SNMP Spezifikationen vor. Zu den bekanntesten und umfangreichsten gehört die SNMP-Software Net-SNMP. Net-SNMP ist Open-Source und frei verfügbar.
Die Möglichkeiten des Netzwerkmanagements, die die SNMP-Spezifikationen definieren und die Net-SNMP implementiert und verfügbar macht, können in VNUML Simulationen komfortabel getestet und analysiert werden. Da die virtuellen Rechner der VNUML Netzwerk-Simulationen UML-Rechner sind, die sich aus Sicht der Software nur geringfügig von reellen Rechnern unterscheiden, ist Net-SNMP auch in diesen virtuellen Netzen voll einsatzfähig. Neben der Vorgehensweise bei der Installation und Konfiguration von Net-SNMP ist auch die bereits vorkonfigurierte Unterstützung für die Verwaltung der Linux Betriebssystem-Ressourcen von Interesse sowie die Möglichkeiten des Einbindens eigener Ressourcen. Net-SNMP könnte auch zur Unterstützung der Durchführung von VNUML Simulationen herangezogen werden und das VNUML-Management um Funktionen erweitern, wie zum Beispiel das Melden von Ereignissen oder das Sammeln von Messdaten.

2 Aufbau der Studienarbeit

Im ersten Teil der schriftlichen Ausarbeitung dieser Studienarbeit werden allgemeine Grundlagen des Netzwerkmanagements angesprochen. Danach folgt eine umfassende Erläuterung der Theorie des Simple Network Management Protocols (SNMP). Aufbau und Funktionsweise von SNMP werden in diesem Teil der Arbeit beschrieben. Anschließend wird auf die Netzwerk-Simulationssoftware Virtual Network User Mode Linux (VNUML) eingegangen. Beispielhaft wird ein kleines Netzwerk-Szenario für die Simulation mit VNUML aufgebaut und durchgespielt. Auf den Zusammenhang zwischen User Mode Linux und VNUML wird näher eingegangen und eine allgemeine Vorgehensweise bei der Installation neuer Software, zur Verwendung in VNUML Simulationen, wird erläutert. Die Routing-Software Quagga wird vorgestellt und die Möglichkeit Quagga über SNMP via SMUX zu verwalten. Abschließend wird die SNMP-Software Net-SNMP umfassend vorgestellt. Die Vorgehensweise bei der Installation und Konfiguration von Net-SNMP wird näher erläutert. In einigen aufgeführten Beispielen werden Möglichkeiten der Anwendung und des praktischen Einsatzes von Net-SNMP dargestellt. Auch das Einbinden individueller Ressourcen-Informationen für die Verwaltung über den Net-SNMP Agenten und weitere Möglichkeiten der Erweiterung werden vorgestellt.
Abschließend wird im Anhang der schriftlichen Ausarbeitung die Vorgehensweise bei der Erstellung eines UML-Rechners auf Basis von Gentoo-Linux für den Einsatz in VNUML erläutert, da die Installation der SNMP-Software Net-SNMP in den von den VNUML-Autoren bereitgestellten UML-Rechner mit einigen Schwierigkeiten verbunden ist.

In dieser Studienarbeit kommt die folgende Software zum Einsatz :

- Net-SNMP Version 5.2.2
  Net-SNMP Projekt-Homepage: http://www.net-snmp.org

- Quagga Version 0.99.4
  Quagga Projekt-Homepage: http://www.quagga.net

- VNUML Version 1.5
  VNUML Projekt-Homepage: http://jungla.dit.upm.es/~vnuml

3 Grundlagen des Netzwerkmanagements

In diesem Kapitel wird auf die allgemeinen Grundlagen des Netzwerkmanagements eingegangen. Hierzu wird die Definition von Netzwerkmanagement der Internationalen Organisation für Normung (ISO) erläutert, die die Aufgaben des Netzwerkmanagements in fünf Funktionsbereiche untergliedert hat.

3.1 Einleitung

Netzwerkmanagement ist die Bezeichnung der Gesamtheit aller Funktionen und Komponenten zur Überwachung und Steuerung von Netzwerken. Es bildet die Voraussetzung für den reibungslosen Betrieb eines Netzes. Netzwerkmanagement steht für den Einsatz verschiedener Mehrwertdienste in Netzwerken, die zur Verbesserung und Sicherstellung der Kommunikations- und Informationsflüsse beitragen sollen.

Zu den Aufgaben des Netzwerkmanagements gehört

- das Sammeln von Informationen über die Nutzung des Netzes,
- die Erstellung von Berichten und Statistiken für Planung, Betrieb, Ausfall und Wartung,
- die Konfiguration des Netzes
- sowie die Leistungs-, Ereignis- und Fehlerüberwachung.

Die Erwartungen die in Netzwerkmanagement gesteckt werden, wuchsen entsprechend mit den Einsatzmöglichkeiten von Netzwerken.
Bevor das Internet Realität wurde gab es das ARPANET, das Netz der „Advanced Research Project Agency“, ein Projekt des Amerikanischen Verteidigungsministeriums und eines der ersten Rechnernetze, das auf „packet-switching“, auf TCP/IP, basierte. Das ARPANET verfügte nur über eine geringe Anzahl von angeschlossenen Geräten, Routern und Terminals, die eine überschaubare Funktionalität besaßen. So genügte hier bereits das einfache PING (Packet Internet Groper) Programm, um effizientes Netzwerkmanagement zu betreiben. Mit dem Zeitstempel im Header des zurückgekommenen PING Paketes (PONG) konnte der Gesundheitszustand einer Netzwerkverbindung abgeschätzt und ein fehlerhaftes Gerät im Netz meist schnell eingegrenzt und ausfindig gemacht werden.
Nachdem 1983 das TCP/IP-Protokoll vom amerikanischen Verteidigungsministerium zum Standard-Internet Protokoll erklärt wurde, wuchsen das Internet und firmeninterne Netzwerke Mitte der Achtziger Jahre sprunghaft an. Die Anzahl der an die Netze angebundenen Geräte und deren Funktionen waren nun weit weniger überschaubar, aber zur Verwaltung, Überwachung und Steuerung der Netze fehlte es an einheitlichen Standards. Es existierten zwar unterschiedliche Tools und Ansätze für Netzwerkmanagement-Standards, doch hatte jeder Hersteller von netzwerkfähigen Geräten und Programmen auch seine eigenen Vorstellungen von Netzwerkmanagement, was gerade die Verwaltung heterogener Netze, also Netze mit unterschiedlichen Geräten verschiedener Hersteller, sehr schwierig machte. So begannen Ende der Achtziger Jahre mehrere unabhängige Gruppen und Standardisierungsorganisationen mit der Entwicklung von Standard-Netzwerkmanagementmodellen.
Die beiden hervorzuhebenden Standardisierungsorganisationen hierbei sind die Internet Engineering Task Force (IETF) und die Internationale Organisation für Normung (ISO), die unterschiedliche Ansätze bei der Entwicklung von Netzwerkmanagementmodellen verfolgten.

Für die IETF sollte Netzwerkmanagement die folgenden Charakteristiken besitzen:

- möglichst einfach Gehalten
- variablenorientierter Ansatz
- der Austausch von Managementinformationen darf auch unzuverlässig sein (z.B. zugunsten der Stabilität)

Im Gegensatz dazu verfolgte die ISO die folgenden Ansätze:

- möglichst leistungsfähiges Management
- objektorientierter Ansatz
- der Austausch von Managementinformationen muss auf zuverlässige Art und Weise erfolgen

Unter Leitung der IETF wurde schließlich das Internet-Standard Management Framework entwickelt und Anfang der 90er Jahre standardisiert. Für das Kommunikationsmodell dieser Managementarchitektur wurde das Simple Network Management Protocol (SNMP) entwickelt, das dann auch Namensgeber des kompletten Managementmodells wurde.

Die ISO realisierte ihre Ansätze der Netzwerkmanagement-Standardisierung im Common Management Information Service (CMIS), was die Dienstprimitiven und die Informationsstruktur definiert, sowie dem Common Management Information Protocol (CMIP) für den Nachrichtenaustausch. Die ISO entwickelte CMIP/CMIS jedoch als Grundlage für ihr OSI-Management, für die Verwaltung von Netzwerken die auf dem Open Interconnection System Reference Model (OSI) basieren, das die bestehenden TCP/IP Netze auch beerben sollte. Da jedoch die breite Einführung OSI basierter Netze auf sich warten ließ, wurde schließlich mit CMOT (CMIP over TCP/IP) das OSI-Protokoll für das Netzwerkmanagement auf einem TCP/IP-Protokollstack entwickelt, das sich auf der Anwendungsschicht wie CMIP verhält.

Ursprünglich war SNMP nur als Übergangslösung für das wesentlich leistungsfähigere aber auch komplexere CMOT gedacht. So baut SNMP auch auf einigen ISO/OSI-Standards wie zum Beispiel ASN.1 (Abstract Syntax Notation One) und dem darauf basierenden Objekt-Registrierungsbaum der ISO auf und hätte wohl ohne größere Schwierigkeiten durch CMOT ersetzt werden können. Doch wie bei allen OSI-Protokollen gab es auch bei CMOT eine Reihe von Startschwierigkeiten, die vor allem in der aufwändigen Implementierung begründet sind. Die Weiterentwicklung von CMOT wurde mittlerweile eingestellt.

3.2 Die Funktionsbereiche des Netzwerkmanagements

Im Zusammenhang mit der Entwicklung des OSI-Managements führte die ISO das Konzept der spezifischen Management-Funktionsbereiche unter der Bezeichnung SMFA's (System Management Functional Area) ein und definierte fünf funktionelle Bereiche, die die Verwaltung eines Netzwerks beschreiben und die auch unter dem Begriff FCAPS bekannt sind. FCAPS ist das Akronym für Faults, Configuration, Accounting, Performance und Security, in deutscher Übersetzung also, Fehler, Konfiguration, Abrechnung, Leistung und Sicherheit.
Im folgenden werden diese fünf Funktionsbereiche des Netzwerkmanagements näher erläutert.

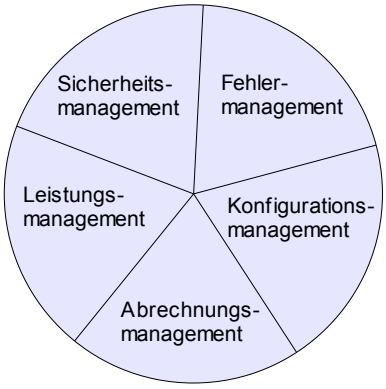


Abb. 3.1: Die Funktionsbereiche des Netzwerkmanagements

3.2.1 Fehlermanagement

Unter dem Begriff Fehlermanagement werden alle Funktionen des Netzwerkmanagements zusammengefasst, die zur Fehlerprophylaxe, Fehlererkennung und Fehlerbehebung im Netzwerk benötigt werden. Der Begriff Fehler steht dabei für unerwünscht auftretende Ereignisse, die sich negativ auf die Funktion und den Betrieb des Netzwerks auswirken. Das vermeiden, identifizieren, isolieren, protokollieren und korrigieren dieser Ereignisse ist die Hauptaufgabe des Fehlermanagements.
Fehlermanagement ist ein sehr wichtiger Bereich des Netzwerkmanagements, denn es dient dazu das Netzwerk in einem verfügbaren Zustand zu halten, bzw. diesen wiederherzustellen. Die Fähigkeit einen Fehler zu erkennen und zu beheben hat Vorrang vor allen anderen Bereichen des Netzwerkmanagements, denn ein Netzwerk muss generell fehlerfrei funktionieren um seine Aufgaben überhaupt erfüllen zu können.

Die Fehlererkennung in Netzwerken kann in zwei Bereiche untergliedert werden.

- **Reaktive Fehlererkennung**
  Hierbei ist der Fehler bereits eingetreten, die beteiligten Netzwerk-Komponenten sind aber dazu fähig auf den Fehler zu reagieren und darauf aufmerksam zu machen.

Eine Reaktive Fehlererkennung wäre zum Beispiel, wenn ein Router feststellt, dass ein an ihn angeschlossenes Gerät nicht mehr antwortet und er darauf reagiert, in dem er eine Fehlermeldung erzeugt, in der der Fehler beschrieben ist.

- **Proaktive Fehlererkennung**
  Die proaktive Fehlerkennung befasst sich mit der frühzeitigen Erkennung von Problemen, die Fehler zur Folge haben können.
  Eine solche Fehlererkennung hätte zum Beispiel ein Datei-Server der feststellt, dass seine Festplatte zu 90% mit Daten gefüllt ist und der daraufhin eine Fehlermeldung erzeugt, in der er vor der zu vollen Festplatte warnt.

Fehlermanagement beschäftigt sich hauptsächlich mit dem bestimmen von ausgewogenen Grenzwerten und überwacht deren Einhaltung. Dabei werden gegenwärtige Werte eines Zustands in regelmäßigen Abständen mit festgelegten Grenzwerten verglichen. Verletzen die gegenwärtigen Zustandswerte die Grenzwerte reagiert das System mit einem vorher definierten Verhalten und erzeugt zum Beispiel eine Alarmmeldung. Von der Definition dieser Grenzwerte hängt oftmals die einfache Fehlererkennung von einer Fehlerfrüherkennung ab je nach dem, ob bei Überschreitung des Grenzwertes der Fehler bereits eingetreten ist oder sein zukünftiges Eintreten wahrscheinlich ist. Zum Bestimmen und Einhalten der Grenzwerte greift Fehlermanagement auf Funktionen des Leistungs- und Konfigurationsmanagements zurück.

3.2.2 Konfigurationsmanagement

Das Konfigurationsmanagement dient im weitesten Sinne dazu, das gesamte vernetzte und verteilte System verfügbar zu machen und den Normalbetrieb zu sichern. Es befasst sich mit der physikalischen und logischen Verbindung der Netzwerk-Geräte und der korrekten Konfiguration darin laufender Anwendungen. Konfigurationsmanagement stellt Funktionen und Hilfsmittel zur Planung, Erweiterung und Änderung der Konfiguration sowie zur Pflege und Dokumentation der Konfigurationsinformationen eines Netzwerks bereit. Zu den Aufgaben gehört auch die Definition von Betriebsmitteln und das Initialisieren und Terminieren derselben. Ebenso das Einstellen und Ändern von Parametern und das Sammeln von Zustandsdaten, so dass auch das wiederherstellen stabiler Zustände möglich ist.
Je nach Netzgröße und Systemvielfalt werden unterschiedliche Anforderungen an das Konfigurations-management gestellt. Dieser Managementbereich bietet heute wohl die meisten Tools, auch weil jeder Hersteller netzwerkfähiger Komponenten herstellereigene Standards aufbaut, um seinen Produkten eine komfortable Zugangsschnittstelle für die Konfiguration zu bieten und auch um eher homogene Netzwerke mit Produkten aus dem eigenen Angebot zu fördern.

3.2.3 Abrechnungsmanagement

Das Abrechnungsmanagement befasst sich mit Funktionen zur ordnungsgemäßen Abwicklung der Benutzung des Netzwerks, wie zum Beispiel Zugangsverwaltung, Verbraucherkontrolle, Abrechnungshilfe und angebotene Mehrwertdienste.
Bei nicht nach Benutzung abgerechneten Netzwerken wird hier auch oftmals der Begriff „Administration" statt „Accounting" oder auch „Benutzermanagement" verwendet. Abrechnungsmanagement stellt Statistiken über die Verwendung von Netzwerkressourcen bereit, so dass Kostenpläne erstellt und Quoten kontrolliert werden können. Zugriffe und Einschränkungen zur Systembenutzung werden geregelt und es wird der Zeitraum festgehalten über den ein Benutzer eine Netzwerkressource verwendet hat. Anhand dieser Daten können die Kosten für die Benutzung des Netzwerks anwenderbezogen zugeordnet und abgerechnet werden. Anwendungs-beispiele für das Abrechnungsmanagement sind Billing-Funktionen und Tariffing-Funktionen.

3.2.4 Leistungsmanagement

Mit Leistungsmanagement ist das Sammeln und Analysieren von Leistungsdaten des Netzwerks gemeint. Leistungsmanagement umfasst Funktionen zur Messung und Auswertung des Leistungsverhaltens des Netzwerkes, in erster Linie um Überlastsituationen rechtzeitig erkennen und vermeiden zu können und vorhandene Leistungsreserven optimal zu verteilen. Mittels des Leistungsmanagements kann festgestellt werden, ob das Netzwerk noch genug Reserven für höhere Belastungen besitzt oder ob es an seiner

Leistungsgrenze angelangt ist und ausgebaut werden muss. Insbesondere die Auswertung und Aufbereitung geschwindigkeitsrelevanter Informationen sind beim Leistungsmanagement von Interesse, zum Beispiel die durchschnittliche Auslastung des Netzwerks, der Durchsatz, die Verfügbarkeit bzw. die Antwortzeiten von einzelnen Endgeräten. Die ermittelten Daten werden meist grafisch als Leistungskurve angezeigt. So können sehr einfach Hoch- und Niederbelastungszeiten des Netzwerks festgestellt und netzwerkbelastende Arbeitsprozesse entsprechend gleichmäßiger verteilt werden.
Mittels Anwendungen des Leistungsmanagements kann auch der durchschnittliche Zustand bzw. der Normalzustand des Netzwerks berechnet werden, so dass Fehlerzustände durch eventuell auftretende ungewöhnliche Leistungsschwankungen frühzeitig erkannt werden können. Die Durchschnitts-Leistungsdaten dienen dann der Berechnung geeigneter Grenzwerte für die Fehlererkennung. Dafür werden in festen Intervallen Leistungsdaten des Netzwerks abgefragt und in einer Datenbank gespeichert um sie statistisch auszuwerten.

3.2.5 Sicherheitsmanagement

Das Sicherheitsmanagement beinhaltet alles was in einem Netzwerk mit Sicherheit und dem Schutz von Informationen zu tun hat. Es umfasst aber auch den Schutz von Objekten, von Diensten und Ressourcen. Sicherheitsmanagement dient der Identifizierung von Risiken aller Art, die den reibungslosen Betrieb des Netzwerks beeinträchtigen können. Das Sicherheitsmanagement hat dafür Sorge zu tragen, dass das Netzwerk gegenüber diesen Risiken abgesichert ist.

Sicherheitsmanagement kann in zwei Ebenen untergliedert werden.

- **Physikalische Sicherheit**
  Diese Ebene der Sicherheit befasst sich mit der physikalischen und lokalen Isolation von Netzwerkkomponenten zum Schutz vor äußeren, physikalischen Einflüssen.

- **Logische Sicherheit**
  Diese Ebene befasst sich in erster Linie mit der Verwaltung von Passwörtern, der Zugriffsberechtigung auf Netze und Komponenten, der Authentifizierung, sowie dem Ver- und Entschlüsseln von Daten.

Sicherheitsmanagement umfasst auch sicherheitspolitische Aspekte und muss ethische und gesetzliche Komponenten, wie zum Beispiel gesetzliche Datenschutzbestimmungen, ebenso berücksichtigen wie wirtschaftliche Voraussetzungen.
Das Sicherheitsmanagement muss unter der Prämisse geplant werden, dass Informationen einen Wert darstellen, der quantifiziert und qualifiziert werden kann und mit entsprechendem Aufwand geschützt werden muss.

3.3 Zusammenfassung

In der Praxis überschneiden sich die Aufgaben der fünf Bereiche, weswegen sie nicht isoliert betrachtet werden dürfen. Je nach Art der Aufgaben, für die ein Netzwerk eingesetzt wird, müssen auch die Funktionsbereiche des Netzwerkmanagements unterschiedlich stark gewichtet werden. Reale Netzwerkmanagement-Systeme unterstützen in der Regel nur einen Teil der aufgezählten Funktionsbereiche.
Auch SNMP unterstützt nicht alle Funktionsbereiche des Netzwerkmanagements. Da SNMP hauptsächlich eine Rahmenstruktur bereitstellt, um Kontrolldaten zwischen einer verwaltenden Netzwerk-Managementstation und einer verwalteten Netzwerk-Komponente zu übertragen, ist es in erster Linie für den Einsatz im Leistungs-, Konfigurations- und Fehlermanagement geeignet. So können mit SNMP Ereignisse aufgezeichnet und Netzwerkmessdaten gesammelt werden. Außerdem können Informationen aufgezeichnet werden, die den Netzwerkaufbau beschreiben. Darüber hinaus unterstützt SNMP auch das Ändern von Geräteparametern und -variablen. Für den Einsatz im Sicherheitsmanagement ist SNMP nur bedingt geeignet, in dem zum Beispiel das Überschreiten eines sicherheitsrelevanten Grenzwertes als Fehler interpretiert wird, der via SNMP an zentraler Stelle gemeldet werde kann. Für den Einsatz im Abrechungsmanagement ist SNMP eher nicht geeignet.

4 SNMP-Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen von SNMP erläutert. Aufbau und Funktionsweise des Protokolls erlauben ein besseres Nachvollziehen der Einsatzmöglichkeiten im Netzwerkmanagement.

4.1 Einleitung

Der Begriff SNMP steht für „Simple Network Management Protocol“, was auf deutsch soviel heißt wie „Einfaches Netzwerk Verwaltungsprotokoll“. SNMP wurde für das Verwalten von verteilten, heterogenen, hersteller- und plattformübergreifenden Netzwerken entwickelt. Das Protokoll gehört zur Gruppe der TCP/IP-Anwendungsprotokolle, die von der Internet Engineering Task Force (IETF) standardisiert wurden. Mittlerweile ist es aber auch für andere Protokollumgebungen, wie zum Beispiel AppleTalk oder OSI, verfügbar. Mit SNMP können Kontrolldaten über eine Netzwerkverbindung transportiert werden. Es ermöglicht hauptsächlich den Austausch von Status- und Statistikdaten zwischen verwalteten Netzwerk-Komponenten und einem verwaltenden Netzwerkmanagement-System. Die primären Ziele von SNMP sind die Verringerung der Komplexität der Managementfunktionen, die Erweiterbarkeit und die Unabhängigkeit der Netzwerk-Komponenten.

Mit SNMP sind hauptsächlich die folgenden Managementaufgaben erfüllbar:

- Überwachung von Netzwerk-Komponenten
- Fernsteuerung und Fernkonfiguration von Netzwerk-Komponenten
- Fehlererkennung und Fehlerbenachrichtigung

SNMP definiert eine verteilte Managementarchitektur, mit Managern und Agenten, die auf dem Client/Server Modell basiert und ein zentrales Netzwerkmanagement für die unterschiedlichsten Netzwerk-Komponenten erlaubt. Der Manager interagiert via SNMP mit seinen Agenten und kontrolliert deren Zustände. Über den Manager lassen sich Managementinformationen über den Zustand der Ressourcen des Agenten abfragen und Parameter, die das Verhalten des Agenten beeinflussen können, konfigurieren. SNMP stellt den Management-Applikationen des Managers dafür eine überschaubare Anzahl an Lese- und Schreiboperationen zur Verfügung.
Darüber hinaus bietet SNMP dem Agenten die Möglichkeit von sich aus seinem Manager eine Ereignismeldung zu zusenden, die diesen über ein im Zuständigkeitsbereich des Agenten eingetretenen Ereignisses unterrichtet.

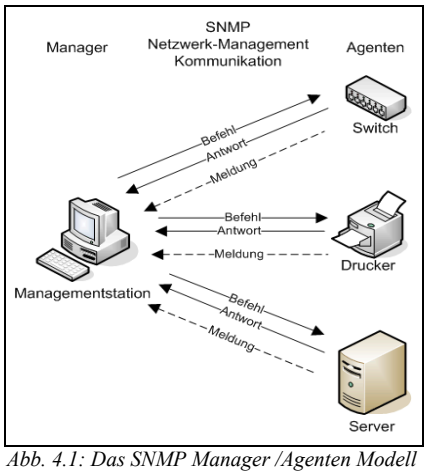


Abb. 4.1: Das SNMP Manager /Agenten Modell

Damit Agenten und Manager, auch verschiedener Hersteller, die gleiche Managementinformation auch auf gleiche Weise behandeln besitzt jeder Agent eine Beschreibung seiner Managementinformationen in einer Datendefinitionssprache, genannt „Structure of Management Information“. Mit Hilfe dieser Definitionssprache werden die Managementinformationen zur Identifizierung in eine logische Struktur abgebildet, die „Management Information Base“ genannt wird.

SNMP entstand in den späten Achtziger Jahren und war ursprünglich nur als kurzfristige Übergangslösung gedacht. Anfang der Neunziger Jahre wurde es von der IETF zum Standard-Internet Management Protocol für TCP/IP Netze erklärt und fand darauf hin sehr rasch Verbreitung. Mittlerweile ist SNMPv3, die dritte Version von SNMP, offizieller Standard. SNMPv3 baut auf den Techniken seiner beiden Vorgänger, SNMPv1 und SNMPv2, auf. Für SNMPv2 wurden verschiedene Spezifikationen veröffentlicht, die unterschiedliche Schwerpunkte bei der Verbesserung von SNMPv1 verfolgen. SNMPv2 wurde jedoch nie ein offizieller Internet-Standard. Letztlich hat sich hier die als SNMPv2c bezeichnete Variante durchgesetzt, die zwar keinerlei sicherheitstechnische Verbesserungen gegenüber SNMPv1 aufweist, aber aufgrund ihrer funktionellen Erweiterungen durch die Einführung neuer Protokolloperationen einen effizienteren Einsatz erlaubt. Darüber hinaus unterscheidet sich SNMPv3 durch wesentlich bessere Sicherheitsmechanismen und einer modularen Architektur, die es ermöglicht einzelne Komponenten der Spezifikation durch andere technische Lösungen zu ersetzen oder um diese zu erweitern. Moderne Netzwerk-Geräte und Managementsoftware unterstützen meist alle drei Versionen von SNMP.
In der Praxis hat sich SNMP in erster Linie als Frage/Antwortprotokoll durchgesetzt das hauptsächlich Aufgaben des Leistungsmanagements wahrnimmt und Zustands-Informationen sowie Leistungsdaten von verschiedenen, vor allem ressourcenarmen, Netzwerk-Geräten an einer zentralen Stelle im Netzwerk sammelt.

SNMP entstand ursprünglich aus dem Secure Gateway Management Protocol (SGMP), das für die Fern-Verwaltung von Internet Gateways entwickelt wurde. Schon kurz nach der Standardisierung von SNMPv1 wurden erste Versuche unternommen die mangelhafte Sicherheit in SNMP zu verbessern. Mit der Spezifikation zu SNMP security sollte das vorhandene community-based Administration Framework durch ein party-based Administration Framework ersetzt werden. SNMP security setzte sich allerdings nie durch. Mit dem Simple Management Protocol (SMP) wurde schließlich ein potentieller SNMP Nachfolger präsentiert, der mit SNMP auch koexistieren konnte. SMP baut auf dem Sicherheitsmodell von SNMP security auf, bietet aber eine bessere Leistung. Mit SMP wurde der Bulk-Retrieval Mechanismus eingeführt, für den effizienteren Umgang mit Managementinformationen in Tabellenstrukturen, sowie eine verbindungsorientierte Manager-zu-Manager Kommunikation für den Austausch von Informationen zwischen Managern. SMP war nicht wie SNMPv1 auf TCP/IP Protokolle beschränkt, sondern konnte eine ganze Reihe weiterer Transportprotokolle verwenden, darunter IPX-, AppleTalk- und OSI-Protokolle. Aus den Spezifikationen zu SMP und SNMP Security wurde schließlich SNMPv2p (party) entwickelt. Das Sicherheitsmodell von SNMPv2p galt aber als zu kompliziert in der Anwendung und konnte sich nicht durchsetzen. Einige Autoren der SNMPv2p Entwicklergruppe spalteten sich ab und förderten die Entwicklung konkurrierender SNMPv2 Varianten. Zu den nennenswerten gehört hier SNMPv2usec, mit benutzer-basierter Sicherheit (user-based Administration Framework), SNMPv2*, mit benutzer-basierter Sicherheit und weiteren Funktionen, und SNMPv2c mit der community-basierten Sicherheit von SNMPv1. Letztere setzte sich letztlich bei den Herstellern von SNMP-fähigen Produkten durch, wurde aber nicht von der IETF standardisiert. Da sich die Autoren der SNMPv2 Spezifikationen nicht einigen konnten berief die IETF eine neue Entwicklergruppe für die Entwicklung einer SNMPv3 Spezifikation. SNMPv3 hatte die Vorgabe die Sicherheit in SNMP zu verbessern. Außerdem sollte es soweit möglich auf bestehenden Spezifikationen aufbauen und bereits vorhandene wiederverwenden. SNMP sollte weiterhin so einfach wie möglich gehalten werden und nur eine minimale Implementierung benötigen. Die Architektur von SNMPv3 sollte modular Aufgebaut werden, so dass Erweiterungen einfach möglich sind. Der modulare Aufbau hatte auch den Zweck, dass die Möglichkeit besteht die einzelnen Komponenten der SNMPv3 Spezifikationen unabhängig voneinander zu standardisieren. Mit SNMPv3 wird schließlich ein Teil der SNMPv2 Spezifikationen übernommen und eine neue modulare Architektur spezifiziert.

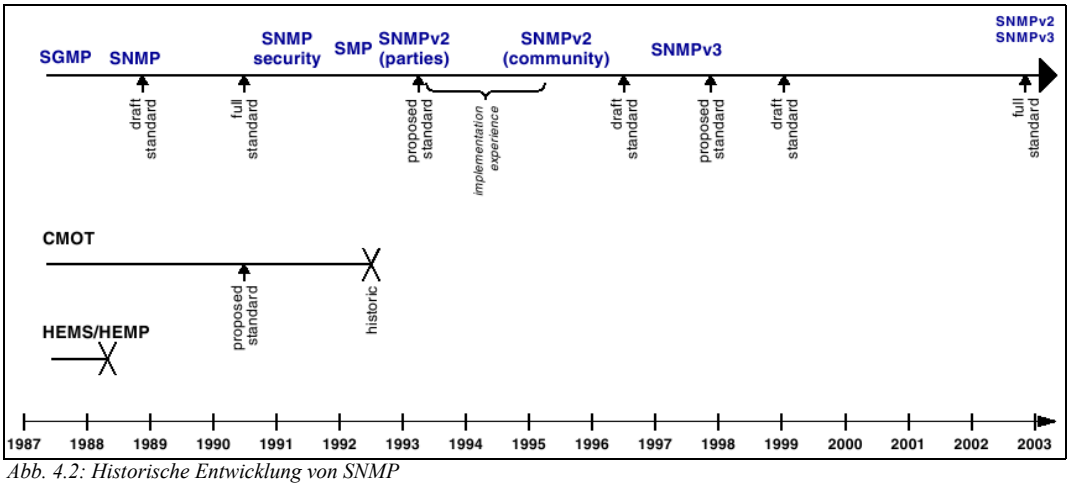


Abb. 4.2: Historische Entwicklung von SNMP

Neben SNMP wurden ursprünglich noch zwei weitere nennenswerte Modelle für den Internet-Management-Standard von der IETF spezifiziert. Das erste ist das High-Level Entity Management System (HEMS) mit seinem High-Level Entity Management Protocol für die Kommunikation. HEMS/HEMP ist eine Verallgemeinerung früherer Host-Monitor-Protokolle, die im Internet-Management zuvor bereits eingesetzt wurden. Das zweite ist CMOT, eine TCP/IP Variante des von der ISO für OSI standardisierten CMIP (Common Management Information Protocol). CMOT galt als leistungsfähiger als SNMP aber auch komplexer in der Anwendung. CMOT und HEMS/HEMP werden nicht mehr weiterentwickelt.
Derzeit gelten Web Services als potentielle Erben von SNMP. Die Kernfunktionen von Web Services sind vom W3C (World Wide Web Consortium) spezifiziert. Dazu gehört SOAP (Simple Object Access Protocol) für den Datenaustausch, WSDL (Web Services Description Language) für die Beschreibung und XML als Basis der Datendefinitionssprache. Ein Web Service unterstützt direkte Interaktionen mit anderen Software-Agenten unter Verwendung XML-basierter Nachrichten durch den Austausch über internetbasierte Protokolle.

4.2 SNMP in den RFCs

SNMP wird über RFC-Dokumente definiert und spezifiziert. RFC steht für „Requests for Comments“, zu deutsch in etwa „Bitten um Kommentare“. Es handelt sich dabei um eine fortlaufend nummerierte Reihe von technischen und organisatorischen Dokumenten zum Internet und dessen Anwendungen. Als RFC Herausgeber galt lange Zeit der Internet Pionier Jonathan Postel. Mittlerweile ist die Funktion des RFC-Herausgebers durch die Internet Society und im Namen der Internet Engineering Task Force (IETF) an eine kleine Gruppe der Netzwerk Abteilung des Information Sciences Institute (ISI) der University of Southern California übertragen worden.
RFCs spielen eine große Rolle beim Standardisierungsprozess der IETF. In RFCs werden Sachverhalte zu neuen Techniken die das Internet betreffen erläutert und es wird zur Diskussion über die Vorgehensweise aufgerufen. Wird der Inhalt eines RFCs von der IETF zur allgemeinen Anwendung empfohlen und zum Standard erhoben, so erhält dieses RFC den Status „Standard“, abgekürzt mit STD, und wird unter einer zusätzlichen STD Nummer, unter der mehrere RFCs der gleichen Thematik zusammengefasst sein können, geführt. Ist der Inhalt eines RFCs in der Praxis nicht mehr relevant, so wird es mit dem Status „Historic“ gekennzeichnet. Sobald ein RFC veröffentlicht wurde, wird es nicht mehr verändert oder gelöscht. Werden in einem RFC Fehler entdeckt, so wird ein Errata zu diesem RFC herausgegeben. Sind es grundsätzliche Fehler oder muss der Inhalt aktuellen Gegebenheiten angepasst werden, so wird ein neues RFC herausgegeben, welches das ältere für obsolet erklärt.

Einige Stadien, die RFCs innehaben können, sind:

- Informational - zur Information, Hinweis, Idee
- Proposed Standard - Vorschlag zum Standard
- Draft Standard - Entwurf zum Standard, Begutachtung von mindestens zwei voneinander unabhängigen Implementierungen ist nun Voraussetzung für Standard
- (Full) Standard - offizieller Standard und zur allgemeinen Anwendung empfohlen
- Experimantal - zum Experimentieren
- Historic - ausgemustert, in der Praxis nicht mehr zur Anwendung empfohlen

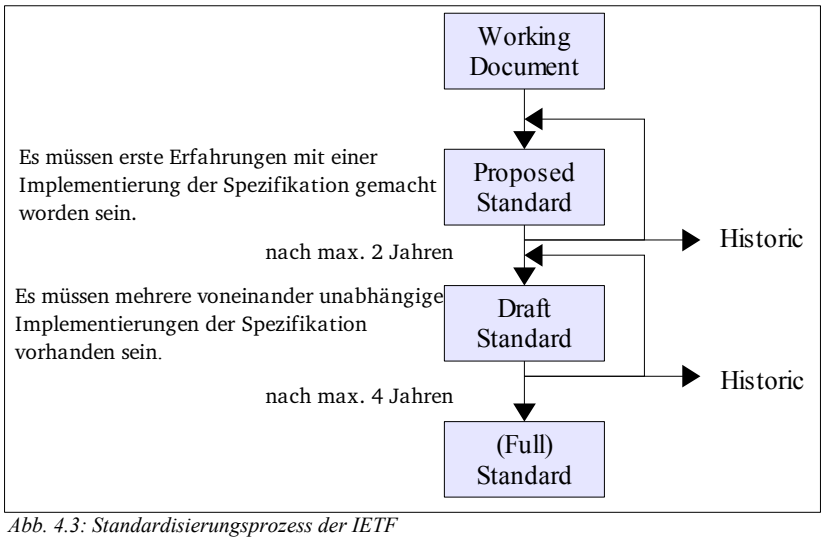


Abb. 4.3: Standardisierungsprozess der IETF

In Abbildung 4.3 ist der Standardisierungsprozess der IETF dargestellt. Damit eine neu entworfene Spezifikation von der IETF zum Internet-Standard erhoben wird, müssen erst über mehrere voneinander unabhängige Softwareimplementierungen der Spezifikation Erfahrungen in der Praxis gesammelt werden.
Beim Standardisierungsprozess der Internationalen Organisation für Normierung (ISO) müssen solche „Referenz“- Softwareimplementierungen nicht vorliegen. Einige Management-Entwickler sehen unter anderem auch darin den Grund dafür, dass das leistungsfähigere Management-System der ISO, das Common Management Protocol over TCP/IP (CMOT), sich letztlich nicht gegenüber SNMP in der Praxis durchsetzen konnte.

Die Suchmaschine des RFC-Editors ist unter der Adresse „http://www.rfc-editor.org“ im Internet zu finden. Zum Thema Simple Network Management Protocol, bzw. SNMP, findet der RFC-Editor gegenwärtig etwa 130 RFC-Dokumente. Stetig werden es mehr da auch die Definitionen neuer Managementinformationen, die Bestandteil des SNMP Standard-Managementkatalogs (MIB II) sind, über RFCs herausgegeben werden.

Bezüglich der von der IETF definierten Standards kann SNMP in drei Komponenten untergliedert werden:

- einer Datendefinitionssprache zum Definieren von Informationen und Ereignissen zum Zwecke des herstellerübergreifenden Managements. Diese Datendefinitionssprache ist ASN.1 (Abstract Syntax Notation One) und wurde von der ISO standardisiert. Für SNMP unterliegt sie einer speziellen Normierung die „Structure of Management Information“ (SMI) genannt wird, weswegen oft auch SMI als die Datendefinitionssprache von SNMP bezeichnet wird. Es gibt derzeit zwei standardisierte Versionen von SMI. SMIv1 blieb Standard obwohl der Nachfolger SMIv2 ebenfalls zum Standard erhoben wurde. Das ist ungewöhnlich liegt aber daran, dass einige Managementinformationen die Bestandteil des Standard-Managementkatalogs (MIB II) von SNMP sind, in SMIv1 definiert sind und dann ebenfalls ihren Standard Status verlieren würden. In der Praxis wird SMIv1 freilich nicht mehr zu Anwendung empfohlen.

- einer standardisierten „Sammelstelle“ für die in SMI definierten Managementinformationen. Auf Basis von SMI ist die „Management Information Base“ (MIB) definiert, eine Art Katalog oder auch textuelle Datenbank, die alle Definitionen der Managementinformationen eines Agenten beinhaltet und diese logisch strukturiert. Eine standardisierte Ausgabe der MIB, mit Definitionen von Management-informationen bzgl. allgemeiner und herrstellerübergreifender, technischer Bereiche des Internets, bietet eine Grundlage dessen, was eine Managementstation an Informationen verstehen sollte und was ein Agent an Managementinformationen, bzgl. der technischen Bereiche die er unterstützt, liefern sollte. Diese MIB liegt derzeit in der Version zwei vor (MIB-II). Es gibt aber zahlreiche Aktualisierungen mit neuen Definitionen von Managementinformationen die allgemeine technische Bereiche des Internets betreffen und als Update des Standards über weitere RFC-Dokumente veröffentlicht wurden.
  Private Organisationen können zusätzlich eigene MIBs für die Managementinformationen ihrer Produkte anbieten, die jedoch selten als RFC-Dokument vorliegen und auch nicht von der IETF standardisiert werden.

- einem Netzwerkmanagement-Protokoll zur Übertragung der Managementinformationen zwischen Managern und Agenten. Dieses Netzwerkmanagement-Protokoll ist das Simple Network Management Protocol (SNMP). Die Bezeichnung SNMP wird jedoch auch für das komplette Internet-Standard Management Framework verwendet. Es gibt drei Versionen des SNMP Netzwerkmanagement-Protokolls, die zusammen eine geringe Anzahl an Protokoll-Operationen für die Management-Kommunikationen zwischen Managern und Agenten definieren. SNMPv1 definiert bereits die Get-, GetNext-, Set- und Trap-Operation für die Lese-, Schreib- und Melde-Kommunikation. Mit SNMPv2 kommt die Inform-Operation, für die Manager-zu-Manager-Kommunikation, sowie die GetBulk-Operation, für eine effizientere Lese-Kommunikation, hinzu. SNMPv3 greift auf die Protokoll-Operationen von SNMPv2 und SNMPv1 zurück und definiert lediglich das Administrations- und das Sicherheitsmodell neu.

SNMP in der Version drei wurde im Dezember 2002 zum Standard erklärt und wird unter der Nummer STD 62 im RFC-Editor geführt. Die gegenwärtige Struktur der Datendefinition von SNMP, mit Namen SMIv2, wird unter der Nummer STD 58 geführt. Sie wurde bereits im April 1999 standardisiert.
Unter der Nummer STD 15 ist die Version 1 von SNMP gelistet, die im Mai 1990 zum Standard erklärt wurde mittlerweile aber den Status „Historic“ inne hat. SNMPv2 wurde nie offizieller Internet-Standard. Hier hat sich aber die als SNMPv2c bezeichnete Variante durchgesetzt, die im Januar 1996 über die RFCs 1901, 1905 und 1906 veröffentlicht wurde.

Überblick und Einleitung zum gegenwärtigen Internet-Standard Management Framework, SNMPv3, sind im RFC 3410 zu finden.

In der folgenden Tabelle sind die wichtigsten RFCs bzgl. des SNMP-Management Frameworks aufgelistet.

RFC-ID	Titel	gegenwärtiger Status	Datum der Veröffentlichung
SNMP Version 1 (SNMPv1)			
1155 (STD 16)	Structure and identification of management information for TCP/IP-based internets	STANDARD	Mai 1990
1156	Management Information Base for network management of TCP/IP-based internets	HISTORIC	Mai 1990
1157 (STD 15)	Simple Network Management Protocol (SNMP)	HISTORIC	Mai 1990
1212 (STD 16)	Concise MIB definitions	STANDARD	März 1991
1213 (STD 17)	Management Information Base for Network Management of TCP/IP-based internets: MIB-II	STANDARD	März 1991
Community-Based SNMP Version 2 (SNMPv2c)			
1901	Introduction to Community-based SNMPv2	HISTORIC	Januar 1996
1905	Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)	DRAFT STANDARD (obsoleted by 3416)	Januar 1996
1906	Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)	DRAFT STANDARD (obsoleted by 3417)	Januar 1996
Structure of Management Information Version 2 (SMIv2)			
2578 (STD 58)	Structure of Management Information Version 2 (SMIv2)	STANDARD	April 1999
2579 (STD 58)	Textual Conventions for SMIv2	STANDARD	April 1999
2580 (STD 58)	Conformance Statements for SMIv2	STANDARD	April 1999
SNMP Version 3 (SNMPv3)			
3410	Introduction and Applicability Statements for Internet-Standard Management Framework	INFORMATIONAL	Dezember 2002
3411 (STD 62)	An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks	STANDARD	Dezember 2002
3412 (STD 62)	Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)	STANDARD	Dezember 2002
3413 (STD 62)	Simple Network Management Protocol (SNMP) Applications	STANDARD	Dezember 2002
3414 (STD 62)	User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)	STANDARD	Dezember 2002
3415 (STD 62)	View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)	STANDARD	Dezember 2002
3416 (STD 62)	Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)	STANDARD	Dezember 2002
3417 (STD 62)	Transport Mappings for the Simple Network Management Protocol (SNMP)	STANDARD	Dezember 2002
3418 (STD 62)	Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)	STANDARD	Dezember 2002

Tabelle 4.1: Einige wichtige RFC-Dokumente zu SNMP (Quelle: www.rfc-editor.org, Stand: Juli 2006)

4.3 Das SNMP Konzept

SNMP wurde mit dem Ziel der Einfachheit ausgelegt. So besitzt das Netzwerkmanagement-Protokoll eine recht überschaubare Anzahl an Operationen über die, die am SNMP-Managementprozess beteiligten Komponenten Managementinformationen anfordern und mitteilen können. Auch die Managementarchitektur die SNMP spezifiziert ist relativ einfach und anspruchslos gehalten. Ein netzwerkfähiges Gerät benötigt daher auch nur relativ wenig Ressourcen und Leistungsreserven um zusätzlich Funktionen des SNMP-Agenten zu unterstützen. Die Implementierung der SNMP-Spezifikationen unterliegt keiner größeren Schwierigkeit. So ist es auch relativ simple neue Managementinformationen über eine Ressource einer Netzwerk-Komponente in den SNMP-Managementprozess einzubinden.

Mit SNMP wird für gewöhnlich nicht nur das eigentliche Netzwerkmanagement-Protokoll bezeichnet, sondern auch die zugehörige Rahmenarchitektur, das Standard-Internet Management Framework.

Das SNMP-Modell eines verwalteten Netzes besteht aus vier Komponenten:

- den Managementstationen, genannt Manager
- den verwalteten Netzwerk-Komponenten, genannt Agenten
- den Managementinformationen (Objekt-Ressourcen)
- dem Netzwerkmanagement-Protokoll SNMP

Die verwalteten Netzwerk-Komponenten besitzen Management-Agenten, die die Management-Funktionen ausführen. Jeder Agent verwaltet dabei eine spezifische Kollektion an Variablen, die den Zustand der Komponente beschreiben, die sogenannten Objekte. Die Ressource eines Objektes ist die Management-information, die den Zustand der Netzwerk-Komponente beschreibt oder konfiguriert. Ein Objekt organisiert, typisiert und definiert die Managementinformation, die ausgelesen oder neu gesetzt werden kann. Die Managementinformationen verarbeitet der SNMP-Agent nicht selbst weiter, sondern macht sie lediglich dem Manager und dessen Management-Applikationen zugänglich. Der Manager ist eine zentrale Rechnerstation im SNMP Management-Netzwerk. Er führt Anwendungen zur Überwachung und Steuerung der Netzwerk-Komponenten aus. Nach Bedarf fordert er bestimmte Managementinformationen von einzelnen Agenten an und protokolliert diese, zum Beispiel für die grafische oder statistische Weiterverarbeitung. Der Manager ist auch die Schnittstelle für die Konfiguration der Agenten. Über ihn können bestimmte Objekte der Agenten mit neuen Managementinformationen versehen und so der Zustand der verwalteten Netzwerk-Komponente beeinflusst werden.
Die Agenten wiederum können selbständig dem Manager Ereignismeldungen zusenden, wenn sie bestimmte Ereignisse, in dem von ihnen verwalteten System, registrieren.
Damit die Managementinformationen von allen beteiligten Komponenten des SNMP-Managementprozesses auf die gleiche Art verwendet und verstanden werden, sind ihre Objekte mit einer eindeutigen Nummer in einer textuellen Datenbank, der „Management Information Base“, zusammengefasst.
Für die Kommunikation zwischen Agenten und Manager definiert SNMP sieben Nachrichten die sich in drei Typen untergliedern lassen. Der Manager verwendet GET-Nachrichten zum Abfragen von Informationen des Agenten und SET-Nachrichten zum Übertragen von Informationen an den Agenten. Der Agent wiederum verwendet TRAP- oder auch INFORM-Nachrichten um dem Manager Informationen zu melden.
Die folgende Abbildung gibt einen Überblick über die Netzwerkmanagement-Architektur des SNMP Modells.

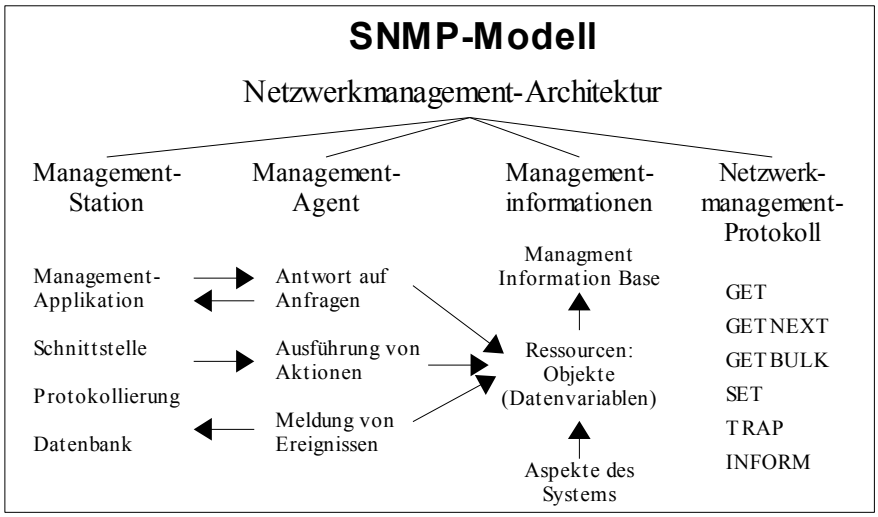


Abb. 4.4: Netzwerkmanagement-Architektur des SNMP-Modells

Die Managementinformationen werden mittels der Syntax-Notation „ASN.1“ in Objekten definiert und mittels der ASN.1 Kodierungsregeln „BER“ für die Übertragung kodiert. In TCP/IP basierten Netzen basiert SNMP auf dem verbindungslosen User Datagram Protocol (UDP). Standardmäßig verwendet SNMP den Netzwerk-Port 161 für die Übersendung von Befehlen und Antworten und den Port 162 für die Übermittlung von Meldungen.
SNMP selbst regelt nur das Sammeln und Übersenden der Informationen und dient lediglich als Schnittstelle zwischen den eigentlichen Management-Applikationen.

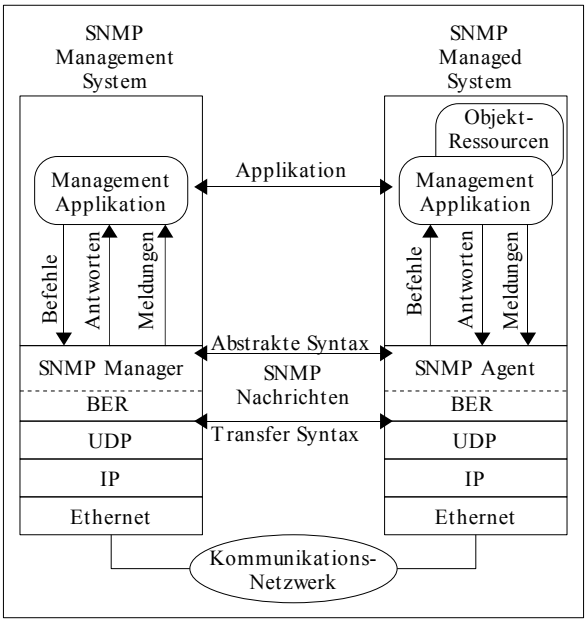


Abb. 4.5: SNMP-Architektur in TCP/IP-Netzen

4.3.1 Der Manager

Der Manager stellt die Schnittstelle zwischen dem menschlichen Netzwerk-Verwalter, bzw. den Management-Applikationen, und den Agenten dar. Im SNMP-Modell ist der Manager meist ein universeller, zentral erreichbarer Rechner im Netzwerk, auf dem verschiedene Management-Applikationen laufen deren Hauptaufgabe das Protokollieren, Verarbeiten und Veranschaulichen der gesamten Management-informationenen des Netzwerks ist. Darüber hinaus bieten diese Management-Applikationen auch die Möglichkeit einzelne Parameter der Agenten zu konfigurieren um den Zustand der verwalteten Netzwerk-Komponenten zu beeinflussen. Auf dem Manager läuft auch eine Serveranwendung, die auf eingehende Ereignismeldungen der Agenten lauscht. Diese Anwendung protokolliert die eingehenden Meldungen und macht gegebenenfalls mit akustischen oder grafischen Signalen darauf aufmerksam.
Der Manager trägt die Hauptlast im SNMP Managementprozess und im Vergleich zu den Agenten ist die Komplexität des Managers recht hoch.
Die Management-Applikationen des Managers interagieren mit den Agenten auf den verwalteten Netzwerk-Komponenten über das Netzwerkmanagement-Protokoll SNMP. Das SNMP-Modell erlaubt eine hierarchische Struktur, was bedeutet, dass Manager selbst wiederum Agenten anderer Manager sein können.
Für die volle Funktionalität der Management-Applikationen benötigt der Manager die textuelle Abbildung der „Management Information Base“ (MIB), in der die Objekte aller Managementinformationen, die die angeschlossenen Agenten bieten, über die Datendefinitionssprache SMI beschrieben sind. Für die Funktion des SNMP-Managers selbst ist das Vorhandensein einer MIB jedoch keine Voraussetzung, da die Management-informationen letztlich über eine Identifikationsnummer vom Agenten abgerufen werden, die aber aus der MIB ausgelesen werden kann. Mit sogenannten MIB-Browsern verfügt der Manager über eine Management-Applikation, die ihm die zur Verfügung stehenden Managementinformationen aufzeigt. Die MIB des Managers dient somit in erster Linie dem Menschen zum Verstehen der verwalteten Managementinformationen.

In SNMPv3 ist die Architektur des Managers modular aufgebaut. Dabei werden die Komponenten der SNMP-Manager-Architektur in drei verschiedene Teile untergliedert: dem Dispatcher, dem Message Processing System (MPS) und dem Security Subsystem.
Der Dispatcher übernimmt die Nachrichten von der Transport- und Applikationsebene und gibt sie an das MPS weiter.

Das MPS übernimmt die Nachricht vom Dispatcher, verarbeitet sie entsprechend der angegebenen SNMP Version und reicht sie dann an das Security Subsystem weiter.
Im Security Subsystem werden die Nachrichten entsprechend dem verwendeten Sicherheitsmodell ver- und entschlüsselt.

Das folgende Schema zeigt den modularen Aufbau des SNMPv3-Managers:

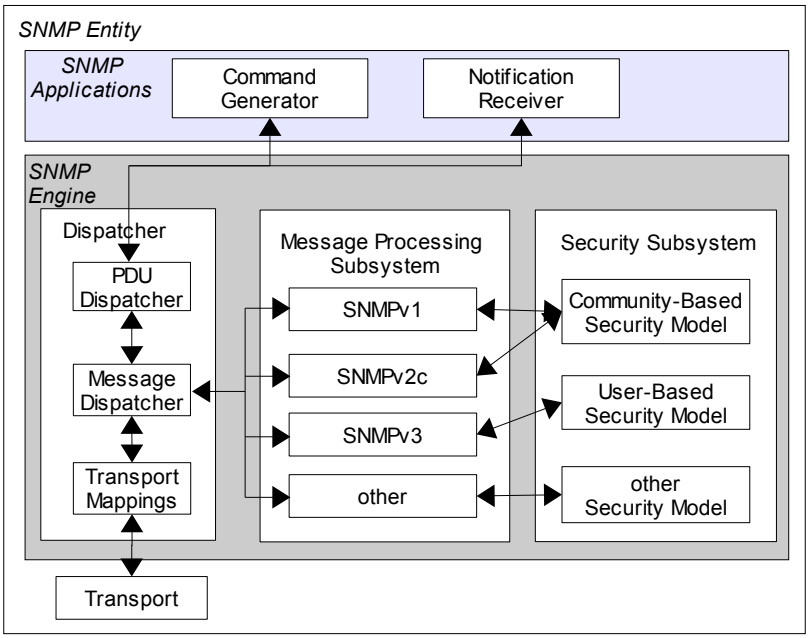


Abb. 4.6: Architektur des SNMPv3-Managers

4.3.2 Der Agent

Ein SNMP-Agent ist ein Serverdienstprogramm, das auf dem zu verwaltenden Netzwerk-Gerät installiert ist, weswegen oftmals auch das komplette Gerät als Agent bezeichnet wird. Agenten können somit Switches, Router, Bridges, Server oder sonst irgendwelche netzwerkfähigen Geräte sein. Die Agenten verfügen über Variablen, die ihren eigenen Zustand und den Zustand der verwalteten Ressourcen der Netzwerk-Komponente beschreiben und beeinflussen. Diese Variablen heißen im SNMP-Jargon Objekte. Agenten können diese Objekte auslesen und deren Werte, die Managementinformationen, ihrer Außenwelt über einen Manager mitteilen sowie Informationen empfangen und die Objekte damit versehen. Zusätzlich kann ein Agent bestimmte Objekte überwachen und bei überschreiten eines zuvor definierten Grenzwertes eine Meldung an einen Manager absenden. Alle Objekte eines Agenten sind in seiner implementierten „Management Information Base“ gesammelt und dort logisch strukturiert.
Netzwerk-Geräte, denen es nicht möglich ist einen eigenen SNMP-Agenten auszuführen, kann ein sogenannter Proxy-Agent vorgeschaltet werden der das Gerät verwaltet. Der SNMP Proxy-Agent überwacht das Gerät, zum Beispiel über Sensoren oder andere Netzwerk-Protokolle und vermittelt die vorhandene Management-information zwischen dem Manager und dem nicht SNMP-fähigen Gerät.
Für das SNMP-Modell existieren auch verschiedene Erweiterungsmodelle für den Agenten über die ein SNMP-Agent in eine verteilte Master- und Sub-Agenten Struktur aufgeteilt werden kann. Der Master-Agent dient dann als Schnittstelle zwischen seinen verschiedenen Sub-Agenten und dem Manager. Die Sub-Agenten sind zum Beispiel Sub-Prozesse unabhängiger, eigenständiger Programme, die über keinen eigenen SNMP-Agenten verfügen. Diese kommunizieren nicht über das Netzwerkmanagement-Protokoll SNMP mit ihrem Master-Agenten, sondern über spezielle Protokolle, die Erweiterungsmodelle wie SMUX (SNMP Multiplexing) oder AgentX (Agent eXtensibility) bereitstellen.

Wie der SNMP-Manager ist auch der SNMP-Agent in mehrere Einheiten untergliedert: dem Dispatcher, dem Message Processing Subsystem und dem Security Subsystem. Zusätzlich hat der Agent eine weitere Einheit: das Access Control Subsystem.
Mit diesem zusätzlichen Subsystem wird der Zugriff auf die verwalteten Objekte geregelt.

Das nachfolgende Schema zeigt den modularen Aufbau des SNMPv3-Agenten:

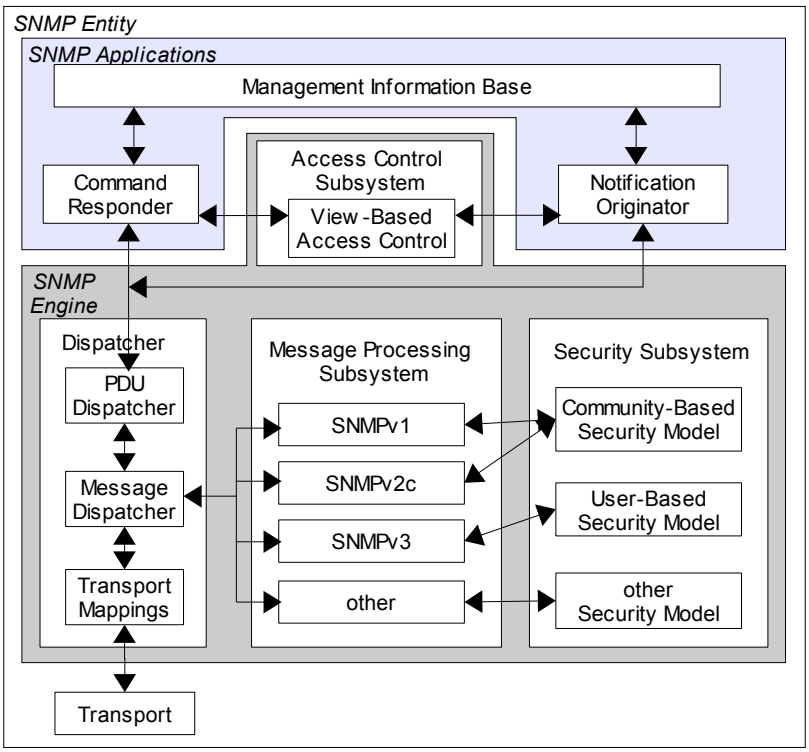


Abb. 4.7: Architektur des SNMPv3-Agenten

4.3.3 Die Managementinformationen

Managementinformationen sind Informationen, die den Zustand einer Netzwerk-Komponente beschreiben und beeinflussen. Hierzu zählen zum Beispiel Statusinformationen, wie die an einer Netzwerkschnittstelle empfangenen Datenpakete, ebenso wie Konfigurationsinformationen, wie die Möglichkeit des Aktivierens oder Deaktivierens der Netzwerkschnittstelle. Der SNMP-Agent strukturiert, implementiert und verwaltet diese Managementinformationen und macht sie dem SNMP-Manager zugänglich. Damit Manager und Agenten die gleiche Managementinformation auf die gleiche Art und Weise verstehen und verwenden muss eine gewisse Erreichbarkeit der Information gewährleistet sein. Hier kann in einer physikalischen und einer logischen Erreichbarkeit unterschieden werden. Die physikalische Erreichbarkeit stellt das Lesen der Management-information aus der verwalteten Komponente oder das Schreiben einer neue Information in die Komponente sowie auch das versenden und empfangen der Information dar. Diese Erreichbarkeit ist Teil der Implementierung des Agenten und muss plattform- und systemabhängig gelöst werden.
Die logische Erreichbarkeit der Managementinformationen ist über die Objekte realisiert. Der Begriff ist irreführend, weil dies keine Objekte im Sinne der Objektorientierung sind. Sie besitzen keine Funktionen, sondern referenzieren lediglich eine Managementinformation, die sie typisieren, umschreiben und mit einer eindeutigen Objekt-Identifikationsnummer ausstatten. Zusätzlich definieren die Objekte auch die Zugriffsrechte, die festlegen, ob die von ihnen geführte Managementinformation nur gelesen werden darf oder auch geschrieben werden kann. Alle Objekte sind schließlich über ihre Objekt-Identifikationsnummer, die über sogenannte Object Identifier (OID) zustande kommt, in einem Katalog strukturiert, einer textuellen Datenbank die „Management Information Base“ (MIB) genannt wird. In der MIB eines Agenten befinden sich alle Objekte zu den Managementinformationen dieses Agenten. Die Objekte können über die eindeutige OID in der MIB gefunden werden. Verlangt ein Manager eine bestimmte Managementinformation von einem seiner Agenten, so sendet er die OID des zugehörigen Objektes an den Agenten. Der Agent schaut anhand der OID in seiner implementierten MIB nach, welche Managementinformation verlangt wird und welchen Restriktionen diese unterliegt. Besitzt der Manager die benötigten Zugriffsrechte startet der Agent über die OID die entsprechende Sub-Routine seines Programms, um die verlangte Information aus der Komponente auszulesen und schließlich zu versenden.
Um Managementinformationen mit Hilfe von Objekten definieren zu können, bedarf es plattformübergreifender und herstellerunabhängiger Standards, die im folgenden beschrieben werden.

4.3.3.1 ASN.1 und BER

Die Objekte sind ein sehr wichtiger Bestandteil des SNMP-Modells. Sie sind Voraussetzung dafür, dass die Managementinformationen unterschiedlicher Produkte verschiedener Hersteller über eine einzige Managementplattform verwaltet werden können. Die Objekte definieren die Managementinformationen über eine unabhängige, standardisierte Datendefinitionssprache. Diese Datendefinitionssprache heißt Abstract Syntax Notation One (ASN.1). Sie ist im ISO-Standard 8824 beschrieben und wurde Anfang der 80er Jahre als eine erste Syntax-Notation für die Protokolle des OSI-Referenzmodells entwickelt. ASN.1 gilt als sehr komplexes Universalwerkzeug zur Darstellung beliebiger Daten in einer maschinenunabhängigen Form. Mit ASN.1 lassen sich hierarchisch strukturierte Datentypen beschreiben. Im wesentlichen handelt es sich dabei um eine, syntaktisch der BNF (Backus-Naur-Form) ähnlichen, Präsentationssprache mit der primitive Datentypen definiert und zu immer komplexeren zusammengefasst werden können. Für die Definition der Objekte des SNMP-Modells wird allerdings nur auf eine vereinfachte Untermenge von ASN.1 zurückgegriffen, die einer speziellen logischen Struktur, genannt Structure of Management Information (SMI), unterliegt. Auch die lexikalische Konventionen sind nicht ganz die gleichen wie in ASN.1.
Tabelle 4.2 zeigt die lexikalischen Konventionen, die in SMI gelten:

Art des Labels	lexikalische Konvention	Beispiel
Integrierte Basis-Datentypen	Alles in Großbuchstaben	INTEGER
Benutzerdefinierte Datentypen	Das Initial ist großgeschrieben	DisplayString
Bezeichner	Das Initial ist kleingeschrieben	sysDescr
Leerräume (z.B. Tabulatoren, Zeilenschaltungen, usw) haben keinerlei Bedeutung		
Kommentare beginnen mit -- und enden beim nächsten Vorkommen von --		

Tabelle 4.2: Lexikalische Konventionen von SMI

ASN.1 verfügt über eine Anzahl von Basis-Datentypen, von denen nur einige wenige auch im SNMP-Modell zulässig sind. Diese sind in der folgenden Tabelle aufgeführt.

Basis-Datentyp	Bedeutung
INTEGER	Ganzzahl mit arbiträrer Länge
BIT STRING	Zeichenkette mit 0 oder mehr Bit
OCTET STRING	Zeichenkette mit 0 oder mehr vorzeichenlosen Byte
NULL	Platzhalter
OBJECT IDENTIFIER	Objektbezeichner (offiziell definierter Datentyp)

Tabelle 4.3: Basis-Datentypen, die SNMP verwenden kann

Der INTEGER Typ kann theoretisch jeden ganzzahligen Wert annehmen. Im Beispiel wird die INTEGER Variable „count“ auf 100 gesetzt.

```
count INTEGER := 100
```

Listing 4.1: Definition eines Integers in ASN.1

Meist werden auf der Basis von INTEGER neue Sub-Datentypen definiert, deren Variablen auf bestimmte Werte oder einen bestimmten Bereich eingeschränkt sind. Im Beispiel wird ein Wert „Status“ definiert der den Wert 1 für „up“, 2 für „down“ oder 3 für „unknown“ einnehmen kann und ein Wert „PacketSize“, dessen Zahlenwert zwischen 0 und 1023 liegt.

```
Status ::= INTEGER { up(1), down(2), unknown(3) }
PacketSize ::= INTEGER (0..1023)
```

Listing 4.2: Definition eines wertebeschränkten Integer-Datentyps in ASN.1

Objekte vom Typ BIT STRING oder OCTET STRING enthalten Null oder mehr Bits, bzw. Bytes, wobei ein Bit entweder 0 oder 1 ist und ein Byte in den Bereich von 0 bis 255 fällt.

OBJECT IDENTIFIER (OID) schließlich bieten die Möglichkeit Objekte eindeutig über eine Objekt-Identifikationsnummer zu identifizieren. Der benutzte Mechanismus dient der Internationalen Organisation für Normung (ISO) zum definieren eines Registrierungsbaumes, in dem im Prinzip jedes Objekt in jedem offiziellen Standard eindeutig identifiziert werden kann. Hierbei wird über allen Objekten der Registrierungsbaum aufgebaut, der sie entsprechend seiner Verzweigungen in immer spezifischere Gruppen untergliedert. Ein Ausschnitt des Baums ist in Abbildung 4.8 dargestellt. Die oberste Ebene dieses Registrierungsbaumes listet alle wichtigen Standardisierungsorganisationen aus Sicht der ISO auf, das ist die ISO selbst und die Internationale Fernmeldeunion (ITU, vormals CCITT) sowie die Kombination der beiden. In der Gruppe der „identified organizations“ (org) sind die Standards anderer anerkannter Standardisierungsorganisationen zu finden, darunter das US-Verteidungsministerium (dod), unter dessen Leitung einst das Internet entstand. Für gewöhnlich sind die SNMP Management-Objekte hier, in den Gruppen „mgmt“, „experimental“ und „private“ zu finden. In der Gruppe „snmpV2“ sind Objekte definiert, die unter anderem die Sicherheitseinstellungen von SNMP betreffen. In der Gruppe „mib-2“ sind alle Objekte zu finden, deren Managementinformationen von der IETF über die „Management Information Base II“ definiert und standardisiert wurden. In der Gruppe „private“ können private Organisationen und Unternehmen eigene Objekte, die zum Beispiel Managementinformationen ihrer Produkte definieren, einfügen. Dafür können sie bei der IANA (Internet Assigned Numbers Authority) eigene eindeutige Objekt-Identifikationsnummern beantragen, so dass ihre Objekte auch eine offizielle Gültigkeit besitzen.

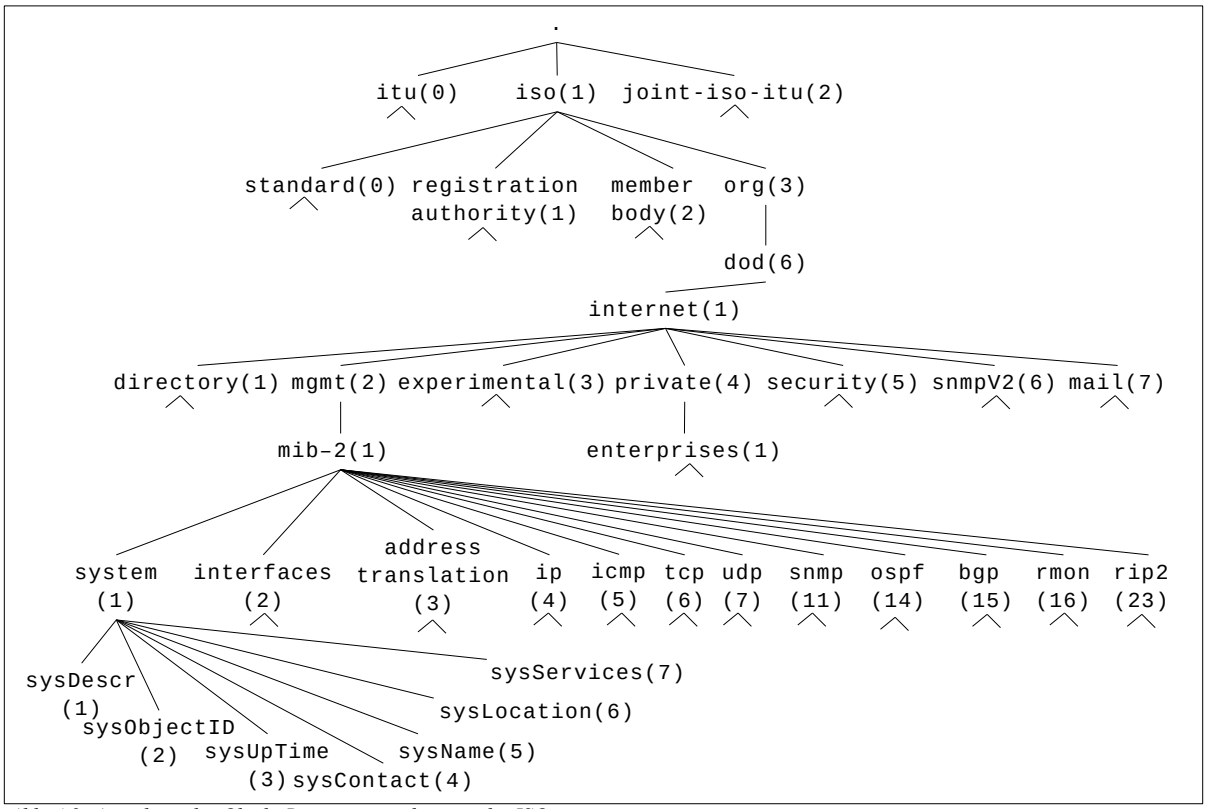


Abb. 4.8: Ausschnitt des Objekt-Registrierungsbaumes der ISO

Die OID eines Objektes ergibt sich aus den Beschriftungen oder Nummern der Knoten, die auf dem Weg von der Wurzel des Registrierungsbaumes bis zum Objekt selbst liegen, jeweils getrennt durch einen Punkt. Die Objekte der Gruppe „system“, die eine Beschreibung des SNMP-Agenten beinhalten und über die jeder SNMP-Agent verfügen sollte, sind somit alle unter der OID „1.3.6.1.2.1.1...“ zu erreichen. Das Objekt „sysDescr“ der Gruppe „system“ ist unter der OID „1.3.6.1.2.1.1.1“ oder unter „iso.org.internet.mgmt.mib-2.system.sysDescr“ zu finden. Auch Mischformen der beiden Notationen sind erlaubt. Um die Managementinformation selbst zu erhalten muss an die OID des Objektes noch ein Instanzwert angehängt werden.

Um aus den in Tabelle 4.3 aufgezählten primitiven Basis-Datentypen neue, komplexere zu bilden werden in SNMP die aus ASN.1 bekannten Konstruktionen SEQUENCE und SEQUENCE OF verwendet. SEQUENCE bildet ein geordnete Typenliste und SEQUENCE OF ein eindimensionales Array bestehend aus einem einzelnen Typ. Analog gibt es in noch SET und SET OF für ungeordnete Listen sowie CHOICE, das aus einer bestimmten Typenliste ein Union erstellt.

Eine andere Möglichkeit neue Datentypen zu erstellen ist das sogenannte Tagging der alten. ASN.1 unterschiedet hier vier Klassen von Datentypen: universelle (UNIVERSAL), anwendungsweite (APPLICATION), kontextspezifische (CONTEXT) und private (PRIVATE) Datentypen. Die universellen Datentypen, zu denen auch die Basis-Datentypen aus Tabelle 4.3 gehören, werden ausschließlich vom ASN.1 Standard festgelegt. Zwischen den anderen Datentypen-Klassen gibt es formal keinen Unterschied, weswegen sie frei verwendet werden können. Allerdings ist es übersichtlich, zwischen anwendungsweiten und auf den Kontext einzelner Datentypen bezogenen Klassen zu unterscheiden. Die Datentypen besitzen eine eindeutige Tag-Nummer. Ein Tag besteht somit aus der Klassen-Bezeichnung und einer Tag-Nummer zur Identifizierung.
Im folgenden werden drei getaggte „application-wide types“ beschrieben.
Der Datentyp „IpAddress“ beschreibt IPv4-Adressen von vier Byte Länge. Die beiden Tags „Counter32“ und „Gauge32“ definieren zwei verschiedene anwendungsweite Typen, die beide durch vorzeichenlose 32-Bit-Ganzzahlen implementiert werden, konzeptionell aber unterschiedlich sind. Der Zahlenwert des Typs „Counter32“ zählt zum Höchstwert hoch und beginnt dann wieder bei 0. Der Wert des Typs „Gauge32“ schwankt im festgelegten Bereich zwischen einem Maximal- und einem Minimalwert.

```
IpAddress ::= [APPLICATION 0] IMPLICIT OCTET STRING (SIZE (4))
Counter32 ::= [ APPLICATION 1 ] INTEGER (0..4294967295)
Gauge32 ::= [APPLICATION 2 ] INTEGER (0..4294967295)
```

Listing 4.3: Tagging von Datentypen

In einem getaggten Typ kann nach der schließenden eckigen Klammer das Schlüsselwort IMPLICIT stehen, wenn der Datentyp klar aus dem Kontext hervorgeht. Dem Empfänger muss dann nicht mitgeteilt werden, welcher Datentyp vorliegt, was zum Beispiel bei einem CHOICE nicht zutreffen würde. Damit wird eine effizientere Kodierung erreicht, da Typ und Nummer des folgenden Datentyps in der Kodierung wegbleiben können.
Inwieweit der IMPLICIT Mechanismus auch von den einzelnen SNMP-Implementierungen übernommen wird ist allerdings unklar.

ASN.1 definiert auch einen komplexen aber mächtigen Makromechanismus, der häufig in SNMP Verwendung findet. Ein Makro kann als eine Art Prototyp benutzt werden, um ein Set neuer Typen und Werte mit eigener Syntax zu erstellen. Jedes Makro definiert einige Schlüsselwörter, die im Aufruf benutzt werden, um die einzelnen Parameter zu definieren. Im Kapitel 4.3.3.2 werden einige Makros aus dem SNMP Umfeld vorgestellt.

ASN.1 definiert Informationen in einer menschenlesbaren Struktur. Für eine Übertragung in einem Netz ist ebenfalls ein Standard erforderlich, um die Informationen einheitlich zu kodieren. Damit auch ein 16 Bit Agent unmißverständlich mit einer 32 Bit Managementstation Informationen austauschen kann, benötigt es Kodierungsregeln die von allen beteiligten verstanden werden. Die Kodierungsregeln transferieren die Syntax von ASN.1 in eine eindeutige Bytefolge. Die von ASN.1 angewandten und auch im SNMP-Modell verwendeten Kodierungsregeln sind die Basic Encoding Rules (BER), die im ISO-Standard 8825 beschrieben sind. Mittels BER wird der Datensatz so kodiert, dass seine Rohdaten in der von der logischen Struktur bestimmten Ordnung rekonstruierbar sind. Die Regeln sind rekursiv, so dass das Kodieren eines strukturierten Datensatzes lediglich die Verkettung der Kodierung der einzelnen Komponenten ist.

Mit BER kodierte Daten bestehen immer aus einer Bytekette, enthalten also immer ein durch acht teilbare Anzahl von Bits. Jeder Übertragene Wert besteht dabei aus vier Teilen.

1. Bezeichner ID (Typ oder Tag)
2. Länge der Inhaltsdaten in Byte
3. Inhaltsdaten
4. Inhaltsende-Flag, nur falls die Länge der Inhaltsdaten unbekannt ist (End-of-content)

Der vierte Teil ist in ASN.1 zulässig aber nur dann vorhanden, wenn der zweite Teil statt einer Längenangabe den Hinweis „keine Längenangabe sondern End-of-content-Markierung“ enthält. In SNMP ist der vierte Teil spezifisch untersagt, weswegen die Länge des Datenfeldes immer bekannt sein muss.
Der erste Teil identifiziert den verwendeten Datentyp. Die ersten zwei High-Order-Bits identifizieren die Klasse des verwendeten Datentyps. Die verwendeten Bits sind hier 00 für universelle, 01 für anwendungsweite, 10 für kontextspezifische und 11 für private Datentypen. Das dritte Bit gibt an, ob es sich um einen einfachen (0) oder um einen zusammengesetzten Datentyp (1) handelt. Die letzten fünf Bits kodieren die Tag-Nummer, falls diese im Bereich von 0 bis 30 liegt. Liegt sie darüber enthalten diese fünf Low-Order-Bits 11111, mit dem wahren Wert der Tag-Nummer in den nächsten Bytes.

Die Kodierung der universellen Datentypen ist durch den Standard definiert. Die relevanten universellen Datentypen des SNMP-Modells sind in der Tabelle 4.4 aufgelistet.

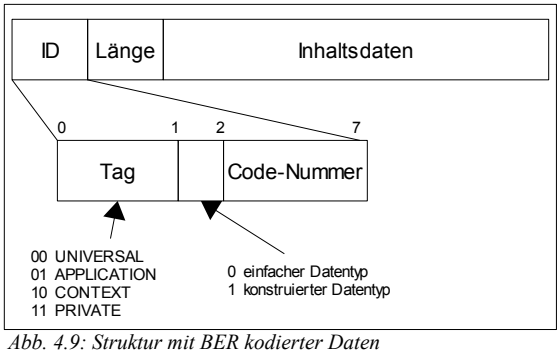


Abb. 4.9: Struktur mit BER kodierter Daten

Code-Nummer	Datentyp
UNIVERSAL 2	INTEGER
UNIVERSAL 3	BIT STRING
UNIVERSAL 4	OCTET STRING
UNIVERSAL 5	NULL
UNIVERSAL 6	OBJECT IDENTIFIER
UNIVERSAL 16	SEQUENCE und SEQUENCE-OF

Tabelle 4.4: BER-Kodierungsnummern einiger universeller Datentypen

Nach dem Bezeichner folgt die Angabe der Länge der Inhaltsdaten. Längenangaben von von weniger als 128 Byte können direkt im ersten Bytefeld kodiert werden, wobei das High-Order-Bit auf 0 gesetzt ist. Übersteigt die Anzahl der Inhaltsdaten die Länge von 128 Byte, so genügt ein Bytefeld nicht zur Angabe der Länge und es werden mehrere Bytefelder benötigt. In diesem Fall wird das erste Bytefeld dazu verwendet, die Anzahl der für die Längenangabe verwendeten Bytefelder anzugeben. Dabei ist das High-Order-Bit im ersten Bytefeld gesetzt und zeigt mit seinen restlichen 7 Bitwerten an, wie viele der noch folgenden Bytefelder allein für die Längenangabe verwendet werden. Benötigen die Inhaltsdaten zum Beispiel 1000 Byte, so steht im ersten Bytefeld der Längenangabe die Zahl 130 (10000010), um anzuzeigen, dass ein 2-Byte-Längenfeld (00000011 11101000) mit der Längenangebe 1000 folgt.

Die Kodierung der Inhaltsdaten hängt vom verwendeten Datentyp ab.
Ganzzahlen werden im Zweier-Komplement kodiert. Eine positive ganze Zahl unter 128 erfordert ein Byte, eine unter 32.768 zwei Byte, usw. .
Bitfolgen werden auch als solche kodiert. Da die Länge der Inhaltsdaten in Bytes und nicht in Bit angegeben ist, wird vor der eigentlichen Übertragung der Bitfolge ein zusätzliches Bytefeld übertragen, in dem angegeben wird, wie viele Bits im letzten Bytefeld, das die Bitfolge belegt, unbenutzt sind.
Die Bytes der Octetfolgen werden einfach standardmäßig von links nach rechts übertragen.
Der Nullwert wird angegeben in dem das Längenfeld auf 0 gesetzt wird.
Ein Object Identifier wird als Folge der darstellenden Ganzzahlen kodiert. Der Object Identifier „internet“ ist zum Beispiel { 1, 3, 6, 1 }. Da die erste Zahl allerdings immer 0, 1 oder 2 und die zweite nach Definition weniger als 40 ist, werden die ersten zwei Zahlen a und b, als 1 Byte mit dem Wert 40 * a + b kodiert. Beim Object Identifier „internet“ ist diese Zahl 43. Zahlen die größer als 127 sind werden, wie bei der Längenangabe der Inhaltsdaten, über mehrere Bytes kodiert, wobei im ersten Byte das High-Order-Bit gesetzt ist und mit den übrigen sieben Bits die Anzahl der für die Längenangabe verwendeten Bytefelder angegeben wird.

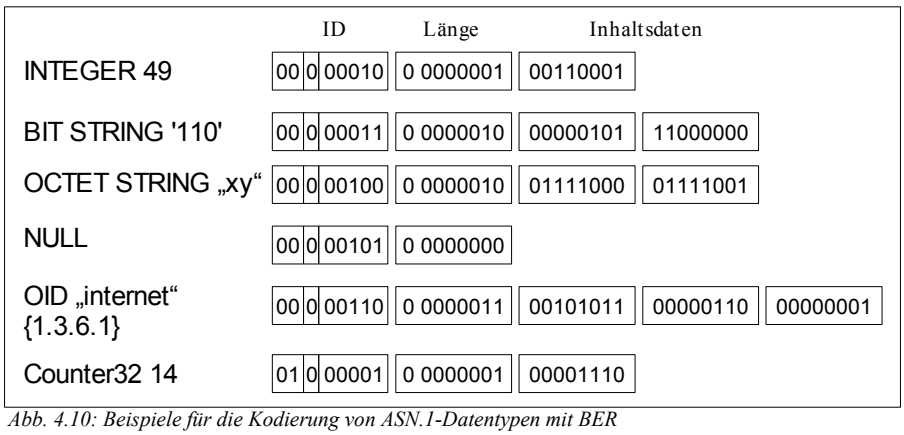


Abb. 4.10: Beispiele für die Kodierung von ASN.1-Datentypen mit BER

In der Abbildung 4.10 sind einige Kodierbeispiele aufgelistet. Beim BIT STRING Beispiel sei noch einmal erwähnt, dass im ersten Bytefeld der Inhaltsdaten die Anzahl der Bits angegeben ist, die im letzten Bytefeld der Inhaltsdaten nicht belegt sind. Hier im Beispiel sind die fünf Lower Bits im letzten Bytefeld unbelegt. Dies ist notwendig da die Angabe der Länge des Wertes in Byte und nicht in Bit ist. Zum OCTET STRING Beispiel sei erwähnt, dass der Buchstabe „x“ den ASCII Hex-Wert 78 und „y“ den Hex-Wert 79 hat.

Im folgenden Beispiel wird ein komplexerer Datensatz in ASN.1 strukturiert und anschließend mittels den BER Kodierungsregeln kodiert. Jede Maschine, die über diese ASN.1 Datentypdefinition verfügt, kann den Datensatz vollständig rekonstruieren.

Datendefinition:

```
Host:
Hostname:       Server13
IP-Address:     192.168.1.13
```

ASN.1 Datentypdefinition:

```
Host ::= [APPLICATION 10] SEQUENCE {
Hostname        HostID,
IP-Address      IpAddress
}

HostID ::= [APPLICATION 11] SEQUENCE {
Name            OCTET STRING
Number          INTEGER
}

IpAddress ::= [APPLICATION 0] IMPLICIT OCTET STRING( SIZE( 4 ) )
```

BER-Kodierung:

Host	Länge	Inhalt									
6a	16	SEQ.	Länge	Inhalt							
		30	14	HostID	Länge	Inhalt					
				6b	0d	SEQ.	Länge	Inhalt			
						30	0b	Name	Länge	Inhalt	
								04	06	{Server}	
								Number	Länge	Inhalt	
								02	01	{13}	
				IP-Address	Länge	Inhalt					
				IMPLICIT(40)	04	{192 168 1 13}					

vollständige Hex-Darstellung der BER-Kodierung:

```
6a 16 30 14 6b 0d 30 0b 04 06 53 65 72 76 65 72 02 01 0d 04 c0 a8 01 0d
```

Abb. 4.11: Beispiel für die Kodierung eines komplexeren ASN.1 Datensatzes mit BER

Der Datensatz enthält eine einfache Host-Rechner Beschreibung, bestehend aus einem Namen und einer IP-Adresse. Der Host-Rechner hier besitzt den Namen „Server13“ mit der IP-Adresse „192.168.1.13“.
Der Datensatz definiert dafür ein Datum „Host“, welches aus einem „Hostname“ und einer „IP-Adress“ besteht. Der „Hostname“ wiederum besteht aus einem „Name“ vom Typ „OCTET STRING“ und aus einer „Number“ vom Typ „INTEGER“. Die „IP-Address“ ist vom Typ „IpAddress“, die wiederum ein vier Byte langer „OCTET STRING“ ist. Über „SEQUENZEN“ werden die einfachen Datentypen zu komplexeren anwendungsweiten Datentypen strukturiert.

4.3.3.2 Structure of Management Information

Structure of Management Information (SMI) ist die Struktur, in der die Managementinformationen für den Umgang mit SNMP definiert sein müssen. Es handelt sich dabei um eine Norm, die Regeln dafür aufstellt, wie die Managementinformationen, unter Verwendung von ASN.1, in Objekten organisiert, typisiert und beschrieben werden. Der SMI Ansatz ist zwar etwas bürokratisch aber auch wichtig, damit hunderte von Produkten verschiedenster Hersteller über eine Managementplattform kommunizieren können. SMI liegt derzeit in der Version zwei vor, genannt SMIv2, und wurde von der IETF über die RFCs des STD 58 standardisiert. Die vorherige Version SMIv1 ist nach wie vor ein Standard (STD 16), wird allerdings nicht mehr zur Anwendung empfohlen.
In der folgenden Tabelle sind in SMIv2 gültigen Datentypen aufgelistet.

Das Suffix 32 oder 64 bei manchen Bezeichnungen gibt die zu verwendende Bitzahl für den Datentyp an.

	Bezeichnung	**Datentyp**	**Beschreibung**
Einfache Typen (Primitive Types)	INTEGER	Numerisch	Ganze Zahl, ohne Definition einer speziellen Beschränkung betreffend der Größe dieser Zahl. Zum Beispiel die Anzahl der von einem Gerät unterstützten Ports. INTEGER kann auch zur Aufzählung von Auswahlmöglichkeiten benutzt werden.
	OCTET STRING	Zeichenkette	Bytefolge mit variabler Länge. Enthält Null oder mehr Bits bzw. Byte. Ein Bit ist entweder 0 oder 1. Ein Byte fällt in den Bereich von 0 bis 255.
	OBJECT IDENTIFIER	Zeichenkette	Ein durch Punkte getrennter String aus ganzen Zahlen, der sich aus der Position des Objektes im ISO-Registrierungsbaum ergibt und die Mangementinformationen der Agenten eindeutig identifiziert.
Einfach konstruierte Typen (Simply Constructed Types)	Integer32	Numerisch	Ganzahl zwischen -2147483648 und 2147483647 (-2^{31} und 2^{31}-1), auch in 64 Bit Systemen
Anwendungsweite Typen (Application-wide Types)	Unsigned32	Numerisch	Ähnlich Integer32, jedoch vorzeichenlos
	Gauge32	Numerisch	Pegel, vorzeichenlose 32-Bit Zahl, kann steigen oder fallen, behält aber den erreichten Höchstwert bei und beginnt nicht automatisch von vorn, Zum Beispiel die Länge der Ausgabe-Warteschlange an einer Schnittstelle.
	Counter32 / Counter64	Numerisch	Zähler, steigt bis zu einem Maximalwert an und beginnt dann automatisch wieder bei Null. Zum Beispiel die Anzahl der an einem Port empfangenen Datenpakete
	TimeTicks	Numerisch	Zeiteinheit, eine nicht negative ganze 32-Bit Zahl, welche die Zeit in hundertstel Sekunden zählt. Zum Beispiel die Berechnung der Systembetriebszeit
	IpAddress	Zeichenkette	Eine 32 Bit-IPv4-Adresse.
	Opaque	Zeichenkette	Datentyp für beliebige Kodierung. Erlaubt das Einpacken jedes beliebigen ASN.1 Typs in einen OCTET STRING
Pseudo Types	BITS	Zeichenkette	Eine Aufzählung von benannten Bits.
Listentypen (Constructor Types)	SEQUENCE		Liste, die null oder mehr Objekte enthält
	SEQUENCE OF		Array, eine Liste von Listen

Tabelle 4.5: Einige SMIv2 Datentypen für SNMP Management-Objekte

Zusätzlich definiert SMIv2 über die RFCs des STD 58 noch einige Textkonventionen, die eine abstraktere Definition von Objekten erlauben und auf den Basis-Datentypen aufbauen. Einige dieser Textkonventionen sind in der folgenden Tabelle aufgelistet:

Name	SMI Basistyp	Beschreibung
DisplayString	OCTET STRING	Ein aus NVT ASCII-Zeichen bestehende Zeichenkette von maximal 255 Zeichen Länge
PhysAddress	OCTET STRING	Eine physikalische Adresse
MacAddress	OCTET STRING	Eine durch OCTET STRING der Länge 6 dargestellte MAC-Adresse nach dem IEEE 802 Standard (Hardware Adresse einer Ethernet Karte)
TruthValue	INTEGER	Über zwei Integer Werte werden Boolean Werte definiert. (1 = true, 2 = false)
TestAndIncr	OBJECT IDENTIFIER	Ermöglicht atomare Operationen und verhindert, dass mehrere Manager auf ein Objekt gleichzeitig zugreifen (im Sinne einer Semaphor).
Autonomie-Type	OBJECT IDENTIFIER	Stellt einen unabhängigen, erweiterbaren Typ-Identifikations-Wert dar. Ermöglicht einen Object Identifier zur Definition eines Unterbaumes mit zusätzlichen Definitionen.
VariablePointer	OBJECT IDENTIFIER	Ein Zeiger auf eine bestimmte Instanz eines Objektes
RowPointer	OBJECT IDENTIFIER	Ein Zeiger auf eine Tabellenzeile
RowStatus	INTEGER	Überwacht den Status und die Konsistenz einer Tabelle, insbesondere, wenn mehrere Manager darauf zugreifen und die Spalten einer Tabelle verändern. Besitzt die folgenden Status angaben: active(1), notInService(2), notReady(3), createAndGo(4), createAndWait(5), destroy(6)
TimeStamp	TimeTicks	Der sysUpTime Objekt Wert, zum Zeitpunkt eines bestimmten Ereignisses. Misst also die Zeit vom Start des Systems bis zum Eintritt des Ereignisses.
TimeInterval	INTEGER	Eine Zeitperiode, gemessen in Einheiten von 0,01 Sekunden
DateAndTime	OCTET STRING	Beschreibt Datum und Zeit
StorageType	INTEGER	Beschreibt die Art von Speicher, die ein Agent benutzt. Mögliche Werte sind [RFC 2579]: other(1) - eh? volatile(2) - Daten gehen bei Reboot verloren (bsp.: RAM) nonVolatile(3) - Daten bleiben permanent erhalten (bsp.: NVRAM) permanent(4) - Daten können verändert aber nicht gelöscht werden (bsp.: z.T. in ROM) readOnly(5) - Daten können weder verändert noch gelöscht werden (bsp.: ROM)
TDomain	OBJECT IDENTIFIER	Beschreibt eine Art von Transport Service
TAddress	OCTET STRING	Beschreibt eine Transport Service Adresse

Tabelle 4.6: Einige SMIv2 Textkonventionen (Entnommen RFC 2579)

Darüber hinaus können über ein Makro auch eigene Textkonventionen, die auf vorhandenen Datentypen oder anderen Textkonventionen aufbauen, definiert werden. Im Beispiel rechts wird die Textkonvention „RunState“, die vier verschiedene Zustände einnehmen kann, auf Basis des Datentyps INTEGER definiert. Die Parameter des Makros sind weitestgehend selbsterklärend, werden aber im Zusammenhang mit einem weiteren wichtigen Makro des SNMP-Modells im folgenden noch erläutert.

```
RunState ::= Textual Convention
STATUS current
DESCRIPTION "..."
SYNTAX INTEGER {
       running(1)
       runable(2)
       waiting(3)
       exiting(4) }
```

Listing 4.4: Definition der Textkonvention „RunState“

SMI gibt vor, dass die Managementinformationen auf unterster Ebene über einzelne Objekte, auch als Skalare bezeichnet, definiert sind. Diese Skalare können in Tabellen strukturiert sein, wobei Tabellen wiederum zweidimensionale Listen von Skalaren sind. Das SNMP Netzwerkmanagement-Protokoll kann somit nur Skalare, bzw. eine Liste von Skalaren austauschen.
Zusammenhängende Objekte werden in Gruppen und Gruppen wiederum zu Modulen zusammengefasst. Gruppen existieren zum Beispiel für IP-Objekte und TCP-Objekte. Eine SNMP-fähige Netzwerk-Komponente sollte nach Möglichkeit alle Objekte einer Gruppe unterstützen, muss allerdings nicht alle Gruppen eines Moduls unterstützen, wenn sie nicht auf die Ressourcen der Netzwerk-Komponente zutreffen.
Alle Objekt-Definitionen werden in ASCII Text-Code innerhalb einer oder mehrerer Textdateien notiert, die hier schlicht MIB-Dateien genannt werden. Das Gerüst, das alle Objekt-Definitionen innerhalb der MIB-Dateien umgibt, ist die „Management Information Base“ (MIB).

```
<Name der MIB>  DEFINITIONS ::=
BEGIN
       Importierte Datentypen
       Definition des Moduls über MODULE-IDENTITY
       Definition aller Baum-Knoten über OBJECT-IDENTITY
       Definition aller Objekte über OBJECT-TYPE (oder NOTIFICATION-TYPE)
       Definition benötigter Implementierungs-Anforderungen und Textkonventionen
END
```

Listing 4.5: Textuelles Gerüst der MIB-Dateien

Der „Name der MIB“ entspricht dabei immer dem Namen der Datei. Zu Beginn können bereits definierte Datentypen aus anderen MIB-Dateien importiert werden. Anschließend folgt die Definition des Modules.
Module beginnen mit dem Aufruf des Makros MODULE-IDENTITY. Seine Parameter liefern Name und Adresse des Implementers, Revisionsgeschichte und andere hauptsächlich administrative Informationen. Normalerweise folgt darauf ein Aufruf des Makros OBJECT-IDENTITY, das Auskunft darüber gibt, an welcher Stelle das Modul in dem ISO-Registrierungsbaum aus Abbildung 4.8 steht. Über das Makro OBJECT-TYPE wird das Management-Objekt definiert. Über das Makro NOTIFICATION-Type wird ein spezielles Management-Objekt definiert, das vom SNMP-Agenten überwacht und für SNMP-Meldungen eingesetzt werden kann. OBJECT-TYPE hat vier zwingende Parameter und wird für gewöhnlich wie folgt definiert.

```
<Name des Objektes> OBJECT-TYPE
SYNTAX <Datentyp des Objektes>
MAX-ACCESS <Zugriffsbeschränkung>
STATUS <Gültigkeit>
DESCRIPTION "textuelle Beschreibung des Objektes"
::= { <OID des Objektes> }
```

Listing 4.6: Definition eines Objektes in SMIv2

Die einzelnen Parameter des Makros OBJEKT-TYPE bedeuten dabei das folgende:

- **SYNTAX**
  Dieser Parameter definiert den Datentyp des Objektes. Datentypen und Textkonventionen die hier aufgeführt werden können sind zum Beispiel in den Tabellen 4.5 und 4.6 aufgelistet.

- **MAX-ACCESS**
  Enthält Informationen über die Zugriffsrechte auf das Objekt, ob es zum Beispiel nur lesbar oder auch beschreibbar ist oder bei Tabellen neue Zeilen in die Tabelle eingefügt werden dürfen.
  Gültige Zugriffsrechte in SMIv2 sind:
  read-only, read-write, read-create, accessible-for-notify, not-accessible.
  SMIv1 kennt stattdessen den Parameter ACCESS, der sich bei SMIv2 in MIN-ACCESS und MAX-ACCESS aufteilt, wobei MIN-ACCESS eine optionale Zusatzangabe über die minimalen Zugriffsrechte des Objektes ist, die bei fehlender Angabe mit den MAX-ACCESS Werten gleichgesetzt werden.

- **STATUS**
  Hat in SMIv2 die drei gültige Optionen CURRENT, OBSOLETE und DEPRECATED. Ein „aktuelles“ (CURRENT) Objekt entspricht der aktuellen SNMP-Spezifikation. Eine „veraltetes“ (OBSOLETE) Objekt ist nicht übereinstimmend, war aber mit einer älteren Version kompatibel und sollte von einem Agenten nicht mehr unterstützt werden. Dazwischen liegen „missbilligte“ (DEPRECATED) Objekte. Sie befinden sich in einer Vorstufe zum veralteten Zustand und wurden bereits durch neuere Objekte ersetzt, werden in der Praxis allerdings noch zu intensiv benutzt, um ausgemustert zu werden.
  Ein Agent sollte immer mindestens all die Objekte eines MIB-Modules unterstützen, die den Status CURRENT innehaben.

SMIv1 kennt die Optionen MANDATORY, OPTIONAL und OBSOLETE. Wobei ein Agent zwingend alle mit MANDATORY gekennzeichneten Objekte unterstützen sollte.

- **DESCRIPTION**
  Dieser letzte zwingende Parameter enthält eine Beschreibung darüber, was das Objekt, bzw. dessen Managementinformation macht. Dieser Parameter dient ausschließlich dem Menschen, da es ihm die Funktion des Objektes erläutert.

Über diese vier zwingende Parameter hinaus gibt es noch weitere, optionale Parameter, mit denen eine recht detaillierte Definition von Managementinformationen über Objekte möglich ist.

Im folgenden ist ein Auszug der „SNMPv2-MIB“-Datei beispielhaft für die SMI-Objekt-Definition aufgeführt.

```
SNMPv2-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, NOTIFICATION-TYPE,
    TimeTicks, Counter32, snmpModules, mib-2                  FROM SNMPv2-SMI
    DisplayString, TestAndIncr, TimeStamp                     FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP       FROM SNMPv2-CONF;

snmpMIB MODULE-IDENTITY
    LAST-UPDATED "200210160000Z"
    ORGANIZATION "IETF SNMPv3 Working Group"
    CONTACT-INFO "..."
[...]
  ::= { snmpModules 1 }

snmpMIBObjects OBJECT IDENTIFIER ::= { snmpMIB 1 }
[ . . . ]
-- the System group
--
-- a collection of objects common to all managed systems.

system   OBJECT IDENTIFIER ::= { mib-2 1 }

sysDescr OBJECT-TYPE
    SYNTAX      DisplayString (SIZE (0..255))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "A textual description of the entity. This value should
            include the full name and version identification of
            the system's hardware type, software operating-system,
            and networking software."
    ::= { system 1 }
[...]
```

Listing 4.7: Auszug aus der SNMPv2-MIB (Entnommen RFC 3418)

In der Datei SNMPv2-MIB sind Objekte zu finden, die Managementinformationen über die Systemumgebung liefern, in der die SNMP-Agentenanwendung läuft. Diese Objekte sind in der Gruppe „system“ zusammengefasst. Unter der OID „iso.org.dod.internet.mgmt.mib-2.system.sysDescr“ bzw. „1.3.6.1.2.1.1.1“ ist das Objekt „sysDescr“ zu finden, welches eine textuelle Beschreibung der Agenten-Systemumgebung definiert. Um diese Beschreibung selbst abzurufen muss an das Ende der OID des Objektes noch ein Instanzwert angehängt werden, so dass die eigentliche Managementinformationsansteuerung die Form OID.y hat. Bei einfach strukturierten Objekten ist die angehängte Instanz für gewöhnlich die Zahl 0, so dass hier in diesem Fall die Beschreibung des Systems unter der OID und der Instanz „1.3.6.1.2.1.1.1.0“ zu finden ist. Weitere Instanzen besitzt ein solches Objekt nicht. Bei Objekten die zu Tabellen strukturiert sind, hängt die Instanz von der Stelle der Managementinformation in der Struktur ab. Hier muss zwischen den verschiedenen Instanzen unterschieden werden, was beim Betrachten der Definition von Tabellen deutlich wird.

```
[...]
-- the Interfaces table
-- The Interfaces table contains information on the entity's
-- interfaces.  Each sub-layer below the internetwork-layer
-- of a network interface is considered to be an interface.

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

ifEntry OBJECT-TYPE
    SYNTAX      IfEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "An entry containing management information applicable to a
            particular interface."
    INDEX   { ifIndex }
    ::= { ifTable 1 }

IfEntry ::=
    SEQUENCE {
        ifIndex                 InterfaceIndex,
        ifDescr                 DisplayString,
        ifType                  IANAifType,
        ifMtu                   Integer32,
        ifSpeed                 Gauge32,
        ifPhysAddress           PhysAddress,
        ifAdminStatus           INTEGER,
        ifOperStatus            INTEGER,
        [...]
    }

ifIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "A unique value, greater than zero, for each interface.  It
            is recommended that values are assigned contiguously
            starting from 1.  The value for each interface sub-layer
            must remain constant at least from one re-initialization of
            the entity's network management system to the next re-
            initialization."
    ::= { ifEntry 1 }

ifDescr OBJECT-TYPE
    SYNTAX      DisplayString (SIZE (0..255))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "A textual string containing information about the
            interface.  This string should include the name of the
            manufacturer, the product name and the version of the
            interface hardware/software."
    ::= { ifEntry 2 }
[...]
```

Listing 4.8: Auszug der Definition der Tabelle „ifTable“ (Entnommen RFC 2863 - The Interfaces Group)

Das Beispiel in Listing 4.8 zeigt einen Auszug des Objektes „ifTable“, das in der Gruppe „interfaces“ innerhalb der IF-MIB definiert ist. Das Objekt „ifTable“ ist als Tabelle strukturiert und enthält viele weitere Objekte die Eigenschaften und Statusinformationen von Netzwerkschnittstellen definieren. In der praktischen Anwendung ist dann für jede vorhandene Netzwerkschnittstelle des Gerätes eine Zeile in der „ifTable“ belegt, deren Inhalt über die Instanz angesteuert werden kann. Das Objekt „ifTable“ ist definiert als eine Liste über „ifEntry“-Objekte. Das Objekt „ifEntry“ wiederum ist definiert als eine Liste über verschiedene Objekte, die die eigentlichen Managementinformationen einer Netzwerkschnittstelle definieren. Jeder Zelle der Tabelle ist somit eine OID zugeordnet, über die die enthaltene Managementinformation angesteuert werden kann.

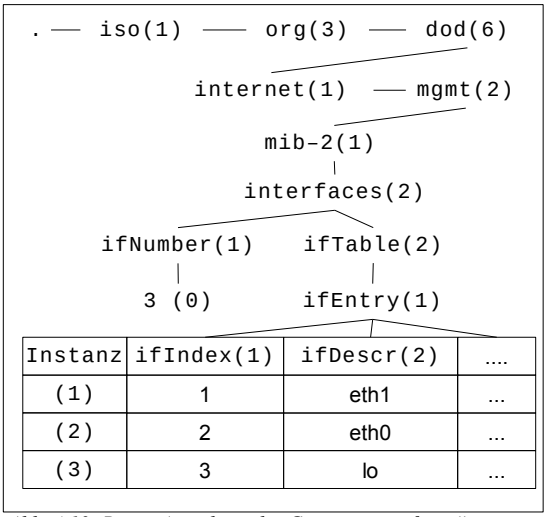


Abb. 4.12: Baum-Ausschnitt der Gruppe „interfaces“

OID	Objekt Wert (Managementinformation)
...	...
1.3.6.1.2.1.2	{ interfaces }
1.3.6.1.2.1.2.1	{ ifNumber }
1.3.6.1.2.1.2.1.0	3
1.3.6.1.2.1.2.2	{ ifTable }
1.3.6.1.2.1.2.2.1	{ ifEntry }
1.3.6.1.2.1.2.2.1.1	{ ifIndex }
1.3.6.1.2.1.2.2.1.1.1	1
1.3.6.1.2.1.2.2.1.1.2	2
1.3.6.1.2.1.2.2.1.1.3	3
1.3.6.1.2.1.2.2.1.2	{ ifDescr }
1.3.6.1.2.1.2.2.1.2 .1	eth1
1.3.6.1.2.1.2.2.1.2 .2	eth0
...	...

Tabelle 4.7: Als Liste dargestellter Ausschnitt der Gruppe „interfaces“

SMIv2 erlaubt auch die Definition von Ereignismeldungen. Diese sind dann ebenfalls in der „Management Information Base“ (MIB) abgelegt und mit einer OID versehen, die sie eindeutig Kennzeichnen. In dem folgenden Beispiel wird eine Meldung definiert, die den Manager darüber aufklärt, das eine Netzwerkschnittstelle in der Systemumgebung des Agenten aktiviert wurde.

```
linkUp NOTIFICATION-TYPE
  OBJECTS {ifIndex}
  STATUS current
  DESCRIPTION
       "A linkUp trap signifies that the
       entity has detected that the
       ifOperStatus object has changed to Up"
::= {snmpTraps 4}
```

Listing 4.9: Definition eines Notification-Objektes für SNMP-Meldungen

Wird die Netzwerkschnittstelle durch den Agenten überwacht, so sendet er dem Manager, nach Aktivierung der Schnittstelle, die OID des Notification-Objektes und eventuell auch weitere beliebige Management-informationen über eine Ereignismeldung zu. Der Manager kann dann über die empfangene OID die zugehörige Notification-Objekt-Definition aus seiner eigenen MIB für weitere Erläuterungen heraussuchen.

Im Kapitel 6 finden sich einige praktische Beispiele im Bezug zur Software Net-SNMP, die die praktische Anwendung von SMIv2 und die Funktion der Objekte noch etwas verdeutlichen.

4.3.3.3 Die Management Information Base

Die von SNMP verwalteten Managementinformationen werden über Objekte in der „Management Information Base“ (MIB) definiert. Diese textuelle Datenbank ergibt sich zwangsläufig aus der strukturellen Vorgabe SMI für die Objekt-Definitionen. Die MIB ist somit eine logische Struktur, in der die Managementinformation zur Identifizierung abgebildet werden. Die MIB ist Bestandteil des Objekt-Registrierungsbaumes der ISO. Es gibt eine von der IETF standardisierte Ausgabe der „Management Information Base“, in der standardisierte Objekte gelistet sind. Diese Objekte sind unter der OID „iso.org.dod.internet.mib-2“ bzw. „1.3.6.1.2“ im ISO-Registrierungsbaum zu finden. Sie definieren allgemeingültige, herstellerunabhängige Management-informationen bestimmter technischer Bereiche des Internet und sind entsprechend ihrer Zugehörigkeit in Kategorien unterteilt und zu Gruppen zusammengefasst. Seit der Veröffentlichung von SNMPv2 ist die MIB in Version 2 unter dem Namen „MIB-II“ von der IETF standardisiert. Die „MIB-II“ bietet eine Art Grundlage dessen, was eine Managementstation verstehen sollte und ein Agent, sofern seine Funktionen den technischen Bereich abdecken, an Managementinformationen bieten sollte.

Einige Gruppen der MIB-II sind hier nun aufgezählt.

Gruppe (OID)	Beschreibung
system (1)	Diese Gruppe muss auf jedem SNMP Agenten vorhanden sein. Sie beinhaltet Informationen über den Agenten selbst wie zum Beispiel Betriebszeit, Systemname und -beschreibung, usw.
interfaces (2)	Beinhaltet die Anzahl und Beschreibung der Netzwerkschnittstellen des Agenten, sowie Status- und Verkehrsflussinformationen
at (3)	Zeigt die Adressübersetzung an (zum Beispiel von Ethernet- in IP-Adressen)
ip (4)	Beinhaltet Tabellen, die die IP-Adressen sowie IP-Routing Informationen des Agenten anzeigen, desweiteren Informationen über die IP-Paketstatistik, zum Beispiel die Anzahl der verworfenen IP-Pakete.
icmp (5)	Enthält eine Statistik über empfangene und gesendete ICMP Nachrichten-Pakete
tcp (6)	Die TCP Gruppe enthält Managementinformationen, die den TCP-Algorithmus und seine Parameter betreffen. Hier wird die aktuelle und kumulierte Anzahl von offenen Verbindungen überwacht
udp (7)	Enthält eine UDP-Verkehrsstatistik. Die Anzahl der übertragenen und unzustellbaren UDP-Pakete wird hier protokolliert. Enthält auch ein Liste von IP-Adressen und Ports, über die lokale UDP Services erreichbar sind.
egp (8)	Die Verkehrsstatistik über das externe Gateway-Protokoll. Dient Routern die EGP unterstützen.
[cmot (9)]	Wird nicht mehr verwendet und ist hier nur der Vollständigkeit halber aufgeführt. CMOT ist der Managementstandard CMIP des OSI-Management-Modells für TCP/IP
transmission (10)	Diese Gruppe ist ein Platzhalter für medienspezifische Objekte. Hier können zum Beispiel ethernetspezifische Statistiken geführt werden
snmp (11)	Beinhaltet Statistiken über den SNMP Betrieb. Listet Fehlermeldungen und Anzahl der verwendeten Nachrichten auf. Alle Agenten sollten diese Gruppe unterstützen

Tabelle 4.8: Einige Gruppen der MIB-II

Neue Objekte der „MIB-II“ werden von der IETF standardisiert und über RFCs veröffentlicht, weswegen es hierzu eine Fülle von RFC-Dokumenten gibt, die hin und wieder durch Aktualisierungen erneuert werden.
Firmen können für ihre Produkte eigene Objekte in eigenen MIBs definieren und in den „enterprises“ Ast des Registrierungsbaumes hängen. Für die offizielle Gültigkeit können sie für ihre MIB und die darin enthaltenen Objekte eine eindeutige OID bei der IANA beantragen. Vom System her ähnelt dies anderen eindeutigen Zuweisungen im Internet, wie zum Beispiel der Internet-Adressen oder der MAC-Adressen Vergabe.
Im Internet gibt es mit dem „MIB-Depot“ eine MIB-Suchmaschine, in der nach MIB-Dateien und deren Objekten gesucht werden kann. Das MIB-Depot ist unter der Internet-Adresse „http://www.mibdepot.com“ zu finden. Es enthält eine Fülle freier MIB-Dateien, die in erster Linie für die Verwendung durch Management-Applikationen auf Managern gedacht sind.
Die Funktion der MIB ist auch ein Kritikpunkt im SNMP-Modell. Damit die Management-Applikationen eines Managers stets reibungslos mit den Agenten kommunizieren können, müssen die Definitionen der Objekte in der MIB des Managers stets mit den Managementinformationen des Agenten übereinstimmen, sonst werden diese falsch interpretiert. Dies kann allerdings nicht immer gewährleistet werden.

4.3.4 Das Netzwerkmanagement-Protokoll SNMP

Das Netzwerkmanagement-Protokoll SNMP dient der Kommunikation zwischen dem Manager und seinen Agenten. Üblicherweise sendet der Manager dem Agenten eine Anfrage und bittet diesen um Managementinformationen. Der Manager kann dem Agenten auch neue Informationen zusenden, die dieser in seinem System übernehmen soll. Der Agent antwortet mit der angeforderten Managementinformation oder bestätigt, dass er seinen Zustand wie angefordert aktualisiert hat. Der Agent kann auch bei Eintritt eines bestimmten Ereignisses unaufgefordert dem Manager Informationen zusenden. Daneben können auch verschiedene Fehler gemeldet werden, zum Beispiel „notWritable“ um anzuzeigen, dass eine Managementinformation nur gelesen werden kann und nicht wie versucht, neu geschrieben werden darf. Für die Übertragung werden die Managementinformationen mit den ASN.1 Kodierungsregeln BER kodiert.

Bezüglich der Verwendung von SNMP können die Nachrichten des Netzwerkmanagement-Protokolls in drei Gruppen unterteilt werden.

- **Befehl (Request)**
  Nachricht vom Manager an den Agenten
- **Antwort (Response)**
  Antwort des Agenten auf einen Befehl des Managers
- **Meldung (Trap/Inform)**
  unaufgeforderte Nachricht des Agenten an den Manager

Der Aufbau des Netzwerkmanagement-Protokolls SNMP ist recht einfach gehalten und kann in drei Teile untergliedert werden.

1) verwendete SNMP Version
2) Informationen bzgl. des verwendeten Sicherheitsmodells
3) Protocol Data Unit (SNMP-PDU)

Die Versionsangabe ist im Anfangsteil einer jeden SNMP Nachricht und gibt die verwendete SNMP-Version (SNMPv1, -v2, -v3) an. Danach folgen Informationen bzgl. des Sicherheitsmodells, das sich bei SNMPv1 und und SNMPv2c in einem einfachen, unverschlüsselten ASCII-Passwort, dem sogenannten „Community-Namen“ erschöpft, während SNMPv3 hier ein wesentlich umfangreicheres und effizienteres Sicherheitsmodell bietet. Darauf folgt der eigentliche Kern der SNMP Nachricht, die „Protocol Data Unit“ (PDU). Im Allgemeinen ist eine PDU ein Block von Informationen, der zwischen zwei oder mehr Einheiten, die von einem Kommunikationsprotokoll verwendet werden, ausgetauscht wird. Die SNMP-PDU beinhaltet den Typ der Nachricht, die OID des Objektes sowie die eigentliche Managementinformation.

Im folgenden werden nun die verschiedenen SNMP-Nachrichtentypen vorgestellt, die für die Kommunikation zwischen SNMP-Manager und SNMP-Agent verwendet werden können.

- **Response**
  Die Response Nachricht ist die Antwort, die meist vom Agenten an den Manager nach Erhalt eines Befehls gesendet wird. Sie beinhaltet die OIDs und die Managementinformationen der vom Manager zuvor angesprochenen Objekte. Ist eine Managementinformation nicht verfügbar, so beinhaltet die Response Nachricht eine der in SNMP standardisierten Fehlermeldungen.
  Sendet der Agent dem Manager eine InformRequest Nachricht, so bestätigt der Manager diese dem Agenten ebenfalls mit einer Response Nachricht.

- **GET**
  Mit der GetRequest Nachricht fordert der Manager den Agenten auf, Managementinformationen auszulesen und zu übertragen. Neben der OID des zur Managementinformation gehörigen Objektes muss der Manager auch die Instanznummer der Managementinformation kennen. Der Manager kann über eine GetRequest Nachricht auch mehrere Managementinformationen auf einmal abfragen.

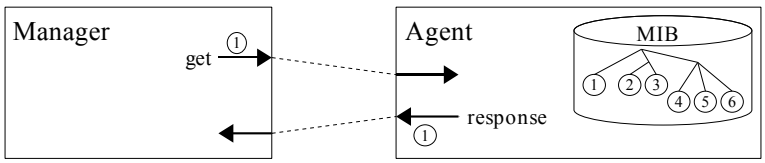


Abb. 4.13: Ablauf einer GetRequest Kommunikation

- **GETNEXT**
  Die GetNextRequest Nachricht ähnelt der GetRequest Nachricht. Hierbei liefert der Agent dem Manager allerdings die Managementinformation, deren Instanznummer der angesprochenen OID als nächstes folgt. Bezüglich des ISO-Registrierungsbaumes wird der Inhalt des nächsten Blattes übertragen. Wird ein Blatt direkt angesprochen so wird der Inhalt des nächstfolgenden Blattes übertragen. Die GetNextRequest Nachricht ist gut für das Auslesen zusammenhängender Managementinformationen geeignet, deren Objekte strukturell aufeinanderfolgen oder die über verschiedene Instanznummern einem Objekt untergliedert sind.

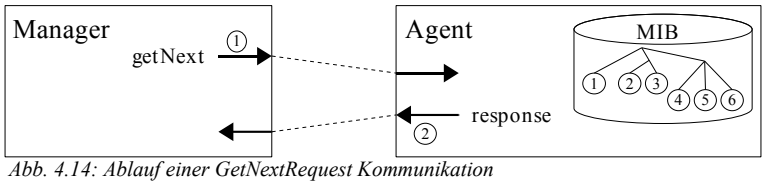


Abb. 4.14: Ablauf einer GetNextRequest Kommunikation

- **GETBULK**
  Die GetBulkRequest Nachricht wurde erst mit SNMPv2 eingeführt um die Effizienz von SNMP, vor allem im Umgang mit Tabellen, zu steigern. GetBulkRequest baut auf GetNextRequest (-PDUs) Nachrichten auf. Für die Abwärtskompatibilität zu SNMPv1 wird allerdings ein Protokollumwandler benötigt.
  Bei der GetBulkRequest Nachricht muss der Manager zusätzlich, zu den OIDs der Objekte deren Managementinformation er abfragen möchte, noch zwei Integervariablen mit übersenden. Die erste Integervariable „non-repeaters“ gibt die Anzahl der angefragten OIDs auf der übersandten Abfrage-OID-Liste an, die mit einer gewöhnlichen GetNextRequest-PDU angesprochen werden. Die zweite Integervariable „max-repetitions“ gibt die Anzahl der GetNextRequest-PDUs an, die dann auf alle verbleibenden OIDs auf der übersandten Abfrage-OID-Liste angewandt werden. Die GetBulkRequest Nachricht ist Dank der beiden Integervariablen gut dafür geeignet Managementinformationen aus Tabellen auszulesen. Die Objekte der Tabelle, die in einer Sequenz strukturiert sind und die Spalten der Tabelle darstellen, und die verschiedenen Instanzen der Objekte, die den Zeilen der Tabelle entsprechen, können so zusammenhängend angesprochen werden.

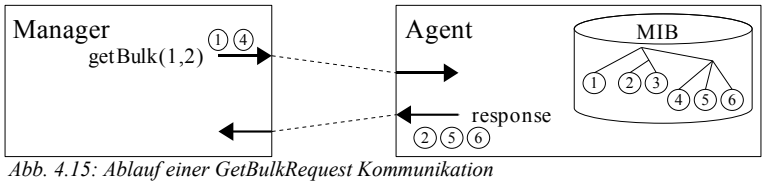


Abb. 4.15: Ablauf einer GetBulkRequest Kommunikation

- **SET**
  Die SetRequest Nachricht ist die einzige Schreiboperation die SNMP bietet. Sie benötigt die OID des Objektes und die Instanz sowie natürlich die neu zu setzende Managementinformation. Außerdem muss auch der Datentyp angegeben werden. Der Agent bestätigt die Übernahme der Managementinformation und die Aktualisierung seines Zustandes mit einer Response Nachricht, die die aktuelle Managementinformation des Objektes enthält. Treten Fehler während der Aktualisierung auf, antwortet der Agent mit einer der in SNMP standardisierten Fehlermeldungen.

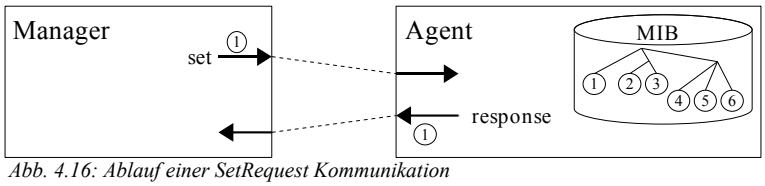


Abb. 4.16: Ablauf einer SetRequest Kommunikation

- **SNMPv1-Trap und SNMPv2-Trap**
  Traps sind unaufgeforderte Ereignismeldungen eines Agenten an einen Manager. Sie sind als unbestätigte Nachrichten konzipiert somit weiß ein Agent nicht, ob seine Trap-Nachricht den Manager auch erreicht hat. Die Trap-Nachricht ermöglicht den Einsatz von SNMP für Aufgaben des Fehlermanagements. Die Managementinformation eines Objektes wird durch den Agenten überwacht und bei übertreten eines Grenzwertes kann die OID des Objektes sowie die zugehörige Managementinformation an den Manager übertragen werden. Es können mehrere OIDs von Objekten und zugehörige Managementinformationen in einer Trap-Nachricht eingebettet werden. In SNMPv1 werden Trap-Nachrichten speziell definiert und gehören einem der folgenden Typen an.

Nummer	Trap	Beschreibung
0	coldstart	Kaltstart des Agenten
1	warmstart	Warmstart des Agenten
2	linkdown	Kommunikationsverbindung deaktiviert
3	linkup	Kommunikationsverbindung aktiviert
4	authentication failure	Authentifizierungsfehler, z.B. unbekannter Community-Name
5	egpneighbourloss	EGP-Nachbar verloren
6	enterprisespecifictrap	firmenspezifische Meldung

Tabelle 4.9: Die in SNMPv1 definierten Trap-Nachrichten

Mit Einführung von SNMPv2 wurde die Trap-Nachricht überarbeitet und so die SNMPv2-Trap eingeführt. Dank SMIv2 ist die Definition neuer Trap-Nachrichten flexibler und diese können über Notification-Objekte in die MIB eingegliedert werden. Die SNMPv2-Trap Nachricht besitzt im Gegensatz zu ihrem Vorgänger einen Aufbau, der dem der anderen Nachrichtentypen des SNMP Netzwerkmanagement-Protokolls gleicht. Für die Abwärtskompatibilität zu SNMPv1 wird ein Protokoll-umwandler benötigt.

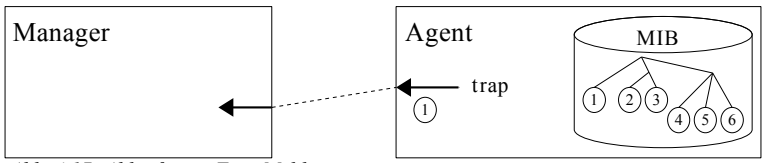


Abb. 4.17: Ablauf einer Trap Meldung

- **Inform**

  InformRequests sind, ähnlich den Traps, Meldungen eines Agenten an einen Manager mit dem Hinweis auf ein eingetretenes Ereignis. Der Agent verlangt hier jedoch eine Empfangsbestätigung über eine Response Nachricht. InformRequests wurden mit SNMPv2 eingeführt. Sie werden meist auch bei der Manager-zu-Manager Kommunikation eingesetzt, um Daten zwischen den Managern abzugleichen. Die InformRequest Nachricht erlaubt somit eine hierarchische Gliederung der Manager-Struktur. Dabei tritt ein hierarchisch untergeordneter Manager gegenüber einem höher gestellten als Agent auf.

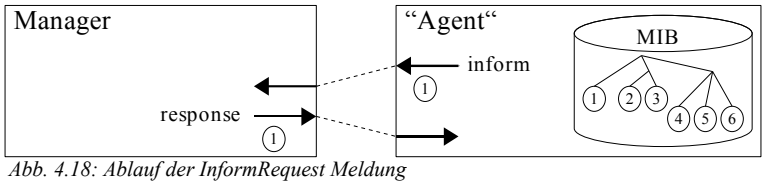


Abb. 4.18: Ablauf der InformRequest Meldung

- ***Report***

  Zusammen mit SNMPv2 wurde auch die Report Nachricht eingeführt. Die Report Nachricht hat keinerlei nähere Spezifikationen erhalten, wird aber oftmals als Erweiterung zur Response Nachricht eingesetzt, mit dem Unterschied, dass der Agent seinem Manager in erster Linie SNMP-Interne Protokolleinstellungen oder Fehlermeldungen mitteilt und nicht direkt Managementinformationen der verwalteten Komponente. Die Report Nachricht nimmt damit einen gewissen Sonderstatus gegenüber den anderen SNMP Nachrichten ein und wird hier nur der Vollständigkeit halber aufgezählt. In der praktischen Anwendung von SNMP ist sie für den Nutzer eher irrelevant.
  Unter SNMPv3 findet zum Beispiel vor der eigentlichen SNMP Request/Response Kommunikation zwischen Manager und Agent eine GetRequest/Report Kommunikation statt, mit der Informationen bzgl. des verwendeten Sicherheitsmodells ausgetauscht werden.

Dies sind alle Nachrichtentypen, die die Standard-Spezifikation von SNMP definiert.

Wie bereits erwähnt, enthält der als SNMP-PDU bezeichnete Teil des Protokolls die verschiedenen Nachrichtentypen, die das Netzwerkmanagement-Protokoll SNMP bietet.

Die SNMP-PDU ist dabei für die einzelnen SNMP-Nachrichtentypen wie folgt aufgebaut:

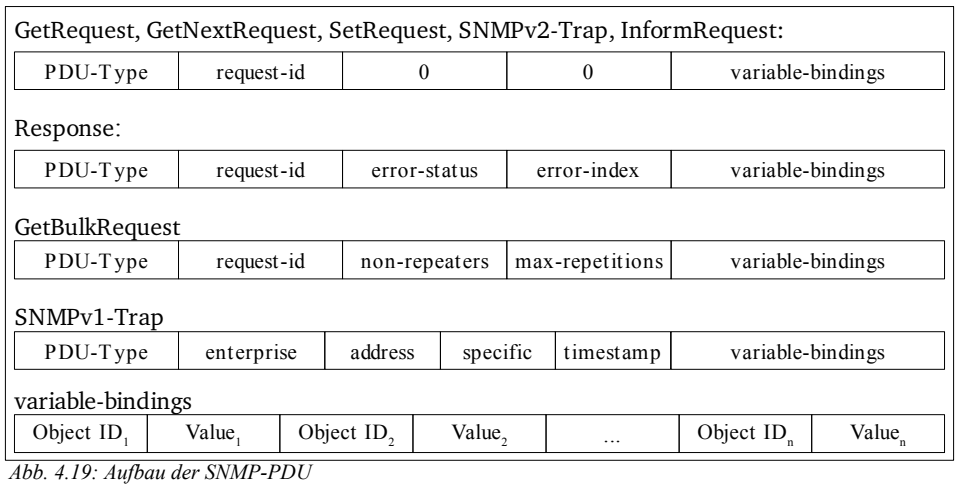


Abb. 4.19: Aufbau der SNMP-PDU

Im folgenden werden die vier Felder einer (Response) SNMP-PDU erklärt:

- **request-id**
  Der Wert dieses Feldes einer Response Nachricht muss dem Wert des Feldes der dazugehörigen Request Nachricht entsprechen. Der Manager kann so die Antworten eines Agenten den einzelnen, zuvor von ihm gesendeten Befehlen, zuordnen.
- **error-status**
  SNMP definiert eine Reihe von Fehlercodes, die im „error-status“ Feld der Response-Nachricht enthalten sein können. Mit der Einführung von SNMPv2 wurden der Umfang der Fehlercodes erheblich erweitert. So können in SNMPv2 und SNMPv3 auch die folgenden standardisierten Fehlermeldungen mit der Response Nachricht als Antwort übermittelt werden.

Fehlermeldung (Code)	**GetRequest GetNextRequest GetBulkRequest**	**SetRequest**	**InformRequest**
noError (0) (auch SNMPv1)	X	X	X
tooBig (1) (auch SNMPv1)	X	X	X
noSuchName (2) (nur SNMPv1)			
badValue (3) (nur SNMPv1)			
readOnly (4) (nur SNMPv1)			
genError (5) (auch SNMPv1)	X	X	X
noAccess (6)		X	
wrongType (7)		X	
wrongLength (8)		X	
wrongEncoding (9)		X	
wrongValue (10)		X	
noCreation (11)		X	
inconsistentValue (12)		X	
resourceUnavailable (13)		X	
commitFailed (14)		X	
undoFailed (15)		X	
authorizationError (16)	X	X	X
notWritable (17)		X	
inconsistentName (18)		X	

Tabelle 4.10: „error-status“ Codes der Response Nachricht

- **error-index**
  Wenn das „error-status“ Feld nicht leer ist, identifiziert der „error-index“ das Objekt in der Objekt-Variablen-Liste, das den Fehler verursacht. Verursacht das erste Objekt in der Liste den Fehler, so ist der Index 1, das zweite Objekt hat den Index 2, usw. .
- **variable-bindings**
  Die Objekt-Variablen-Liste ist eine Liste aus Paar-Elementen. Das erste Element ist die OID des referenzierten Objektes. Das zweite Element entspricht einem der folgenden Werte:
  - value: Datentyp und zugehörige Managementinformation des referenzierten Objektes
  - unSpecified: Null-Wert, wird in den Lese-Anfragen des Managers verwendet.
  - noSuchObject: Zeigt an, dass das Objekt im Agenten nicht implementiert ist
  - noSuchInstance: Zeigt an, dass die Instanz des Objektes für die Operation nicht gültig ist
  - endOfMibView: Zeigt an, dass das Objekt außerhalb der (zugriffsberechtigten) MIB liegt

Der Aufbau der kompletten SNMP-Nachricht unterscheidet sich zwischen den Versionen. Die SNMP Versionen SNMPv1 und -v2c besitzen neben der SNMP-PDU lediglich noch die beiden Felder für die Angabe der Version und dem Community-Namen.
Der Aufbau der kompletten Nachricht dieser Versionen sieht wie folgt aus.

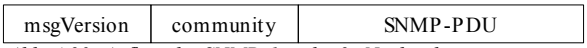


Abb. 4.20: Aufbau der SNMPv1 und -v2c Nachrichten

SNMPv3 Nachrichten zeichnen sich durch einen komplexeren Aufbau aus, der dem besseren Sicherheitsmodell und dem modularen Aufbau der SNMPv3 Kommunikationsteilnehmer geschuldet ist.

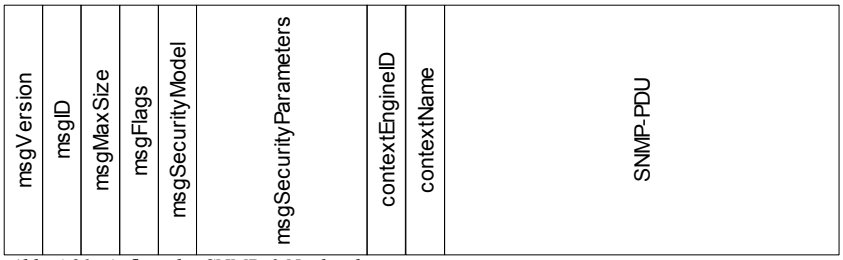


Abb. 4.21: Aufbau der SNMPv3 Nachrichten

Ursprünglich wurde SNMP nur für die TCP/IP Protocol-Suite konzipiert. Mittlerweile ist es aber auch in anderen Protokollumgebungen, wie zum Beispiel AppleTalk, einsetzbar. Für diese Studienarbeit interessiert allerdings nur der Einsatz in TCP/IP Netzen.
In TCP/IP Netzen baut SNMP auf dem verbindungslosen Kommunikationsprotokoll UDP (User Datagramm Protocol) auf. Dies bedeutet zwar für die Kommunikationsteilnehmer zum einen, dass es keine Garantie gibt, dass ihre Nachricht das Ziel auch erreicht. Zum anderen erhält SNMP dadurch aber auch eine gewisse Robustheit, da die Kommunikationsteilnehmer nicht aufeinander angewiesen sind um zu funktionieren. Sollte es so in einem Netzwerk tatsächlich zu einer Problemsituation kommen, wird das Netz nicht auch noch durch das Überwachungswerkzeug weiter belastet, indem dieses unaufhörlich versucht seine Verbindungen aufrecht zu halten.
Mit Ausnahme der Trap-Nachricht wird ohnehin immer auf eine Bestätigung der Nachricht gewartet, in der zum Beispiel die angefragte Managementinformation enthalten ist.
Für Befehle und Antworten nutzt SNMP standardmäßig den Port 161. Meldungen werden über den Port 162 versandt.

Im folgenden ist nun eine komplette SNMP-Nachricht aus einem TCP/IP basierten Netzwerk abgebildet. Die Nachricht wurde mit dem Protocol-Analyzer Programm „Ethereal“ entschlüsselt. Es handelt sich um die Antwort auf die SNMPv1 Anfrage des Net-SNMP Managers „uml1“ (IP: 10.0.0.1) an seinen Net-SNMP Agenten „uml2“ (IP: 10.0.0.2) nach der Bezeichnung einer seiner Netzwerkschnittstellen und der darauf eingegangenen Bytepakete.

Die SNMP-Kommunikation mit Hilfe von Net-SNMP ist im folgenden dargestellt:

```
uml1 # snmpget  -v1 -c public 10.0.0.2 .1.3.6.1.2.1.2.2.1.2.1 .1.3.6.1.2.1.2.2.1.10.1
IF-MIB::ifDescr.1 = STRING: "eth1"
IF-MIB::ifInOctets.1 = Counter32: 3319
```

Befehl 4.1: SNMPv1 Anfrage, ausgeführt mit Net-SNMP

Hier ist nun die entschlüsselte Antwort (Response) des Agenten auf die Anfrage des Managers.

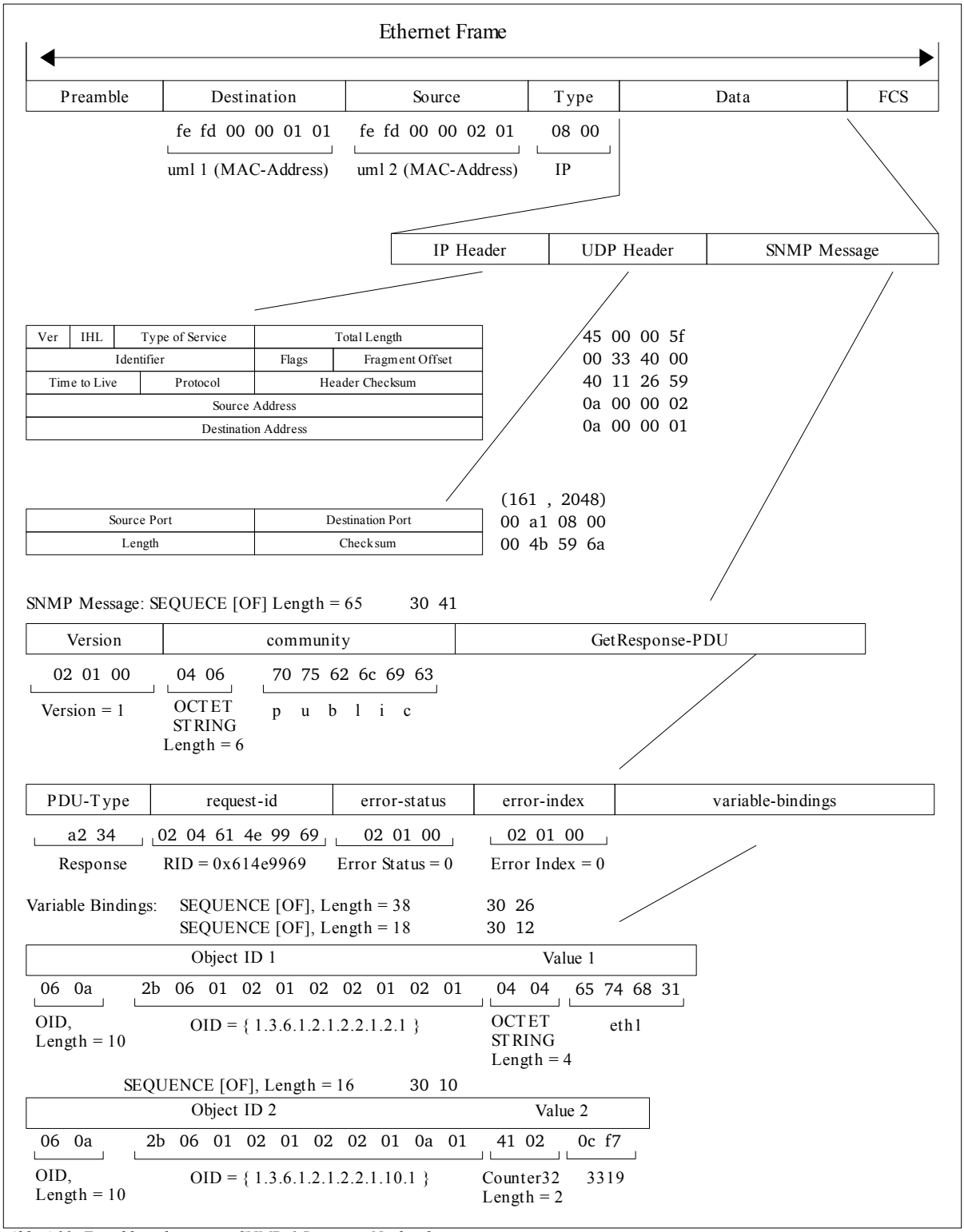


Abb. 4.22: Entschlüsselung einer SNMPv1 Response Nachricht

4.4 Sicherheit in SNMP

Das Simple Network Management Protocol dient der Verwaltung von Netzwerk-Komponenten von einem Management-Arbeitsplatzrechner aus. Es bietet eine Fülle von Informationen über Struktur und Eigenschaften des Netzwerks und seiner Komponenten. Lange war der unzureichende Schutz der mit SNMP verwalteten Managementinformationen vor unberechtigtem Zugriff ein Kritikpunkt, was sicherlich mit ein Grund dafür ist, das SNMP in der Praxis meist nur eingeschränkt, im Sinne des Leistungsmanagements, genutzt wird.

4.4.1 SNMPv1 und SNMPv2c

SNMPv1 und SNMPv2c nutzen ein Community-basierendes Administration Framework für die Sicherheit der Kommunikation zwischen den verschiedenen Komponenten des SNMP Netzwerkes. Eine Community besteht dabei aus mindestens einem Agenten und einem Manager. Der Community-Name ist eine ASCII-Zeichenkette und Bestandteil jeder SNMPv1 und SNMPv2c Nachricht. Er dient der Authentifizierung. Der Manager muss in jeder Anfrage an den Agenten dessen Community-Namen angeben. Besitzt der Agent den Community-Namen nicht, so antwortet er nicht auf die Anfrage. Es können mehrere Communities in einem SNMP-Agenten eingerichtet werden. Unterschiedlichen Community-Namen können zudem unterschiedliche Zugriffsrechte zugeteilt werden. Dies ermöglicht die Vergabe individueller Zugangsberechtigungen für bestimmte Teilbereiche des verwalteten Netzwerks. Die Community besitzt somit die Funktion eines Zugangskontos, wobei der Community-Name auch gleichzeitig das Passwort des Kontos darstellt. In der einfachsten und weit verbreitetsten Form gibt es lediglich zwei Community-Zugriffsberechtigungen:

- read-only - erlaubt nur lesenden Zugriff
- read-write - erlaubt auch das konfigurieren von Parametern

Viele Hersteller SNMP-fähiger Produkte pflegen hier die Unsitte die Standardvorgabe „public“ und „private“ für die beiden Communities zu verwenden. Doch auch spezielle Community-Namen bieten keinen ernsthaften Schutz, da SNMPv1 und SNMPv2c keinen internen Verschlüsselungsmechanismus bieten. Die komplette SNMP-Nachricht sowie die enthaltenen Daten werden für gewöhnlich im Klartext übertragen. Viele Protocol-Analyzer wie zum Beispiel „Tcpdump“ oder „Ethereal“ haben keine Schwierigkeiten damit die relevanten Informationen aus den SNMP-Nachrichten heraus zu filtern. Vor allem aufgrund der unzureichenden Sicherheit des weit verbreiteten SNMPv1 wurde der Begriff SNMP auch als Abkürzung für „security is not my problem“ verspottet. Einer der wesentlichen Gründe für die Weiterentwicklung von SNMP war die Lösung des Sicherheitsproblems, was allerdings erst mit der Einführung von SNMPv3 zufriedenstellend gelang.

4.4.2 SNMPv3

Mit der Einführung von SNMPv3 sollte die unzureichende Sicherheitsfunktionalität in SNMP beseitigt werden. Vor allem die folgenden Aspekte waren bezüglich des Sicherheitskonzeptes zu erfüllen.

- Verhinderung der Veränderung von Managementinformationen durch Unbefugte
- Verhinderung der Einspeisung von falschen oder veränderten Nachrichten durch Unbefugte
- Verhinderung von falschen Manager-Identitäten
- Optionale Verschlüsselung der Kommunikation

SNMPv3 bietet die Möglichkeit Authentifizierungs- und Verschlüsselungstechniken einzubinden. Vor allem aufgrund der modularen Architektur von SNMPv3 können dabei unterschiedlich realisierte Sicherheitstechniken verwendet werden. Im Rahmen der SNMPv3 Entwicklung wurden zwei Sicherheitsmodelle spezifiziert, das „View-Based Access Control Model“ und das „User-Based Security Model“, die im folgenden näher erläutert werden.

4.4.2.1 Das User-Based Security Model

Das „User-Based Security Model“ (USM) ist das Sicherheitsmodell von SNMPv3. Das Modell ist im RFC 3414 (STD 62) spezifiziert. Es baut auf einem benutzer-basierten Konzept auf. Dem Benutzer kann, neben seinem Namen, auch ein „Authentification-Key“ für die Authentifizierung und ein „Privacy-Key“ für die

Verschlüsselung zugeteilt werden. Standardmäßig wird für die Verschlüsselung der DES-Algorithmus (Data Encryption Standard) verwendet und für die Authentifizierung die Hashfunktionen MD5 (Message Digest Algorithm 5) und SHA-1 (Secure Hash Algorithm). Es können jedoch auch andere Techniken eingebunden werden. Über die Authentifizierung können die SNMP-Nachrichten auf Modifikationen überprüft werden und die korrekte Identität des Benutzers kann sichergestellt werden. Die Verschlüsselung schützt den Inhalt der SNMP-Nachricht vor der Enthüllung durch Unbefugte Parteien. Zusätzlich wird über einen timestamp auf den zeitlichen Bezug der SNMP-Nachricht geachtet, als Schutz gegen späteres oder erneutes versenden.

4.4.2.2 Das View-Based Access Control Model

Das „View-Based Access Control Model“ beschreibt Elemente der Zugriffskontrolle auf die Managementinformationen eines Agenten über die OID der Objekte. Mit Hilfe des Models kann bestimmt werden, wer in welcher Form auf welche Managementinformationen Zugriff hat. Das gegenwärtige Model wird im RFC 3415 (STD 62) beschrieben. In diesem RFC wird auch ein standardisiertes MIB-Modul, das SNMP-VIEW-BASED-ACM-MIB Modul, spezifiziert, dessen Objekte die Verwaltung der Parameter via SNMP ermöglichen.
VACM besitzt die folgenden Konfigurationsparameter, die in einem SNMPv3 Agenten konfiguriert werden können:

- **Groups**
  Unter diesem Parameter können Zugriffsrechte unter einem Gruppennamen zusammengefasst werden. Dabei bezieht sich eine Gruppe auf eine Menge von Tupeln der Form <securityModel, securityName>. Umgekehrt kann eine Tupelkombination höchstens einer Gruppe zugeordnet werden. Der Begriff securityModel steht dabei für das verwendete Sicherheitsmodell, zum Beispiel SNMPv1, SNMPv2c oder USM. Der Begriff securityName steht für eine eindeutige Kennung.

- **securityLevel**
  Das securityLevel definiert die Sicherheitsstufe, die beim Zugriff auf ein Objekt aktiviert ist. So kann zum Beispiel ein Schreibzugriff auf ein Objekt eine Authentifizierung erfordern, während der Lesezugriff ohne erlaubt ist. Für gewöhnlich können drei verschiedene Sicherheitsstufen definiert werden, die sich auf die Authentifizierung und die Verschlüsselung beziehen.
  - noAuthNoPriv - keine Sicherheitsfunktionen sind aktiviert
  - authNoPriv - Authentifizierung über ein Passwort wird verlangt
  - authPriv - sowohl die Authentifizierung als auch die Verschlüsselung wird verlangt

  Verschlüsselung ohne Authentifizierung wurde für nicht sinnvoll befunden, daher gibt es den securityLevel „noAuthPriv“ nicht.

- **Contexts**
  Ein Context fasst mehrere Objekte zusammen. Die Zugriffsbestimmung zu den einzelnen Objekten ist hiermit einfacher möglich. Ein Objekt kann in mehreren Contexts vorkommen, die verschiedenen Zugriffsrechten unterliegen.

- **MIB Views**
  MIB Views sind Sichten auf die MIB. Dabei wird ein bestimmter Ausschnitt auf die MIB definiert, der dann einer Gruppe zugeordnet werden kann. So kann recht einfach der Zugriff einer Gruppe auf die Objekte eines bestimmten Ausschnitts der MIB geregelt werden.

- **Access Policy**
  Über die Access Policy wird schließlich die eigentliche Zugriffskontrolle definiert. Es fasst die obigen Parameter zusammen und definiert so den Zugriff auf die Managementinformationen des Agenten. Somit muss zum Beispiel eine MIB View nur einmal definiert werden und kann dann verschiedenen Gruppen mit unterschiedlichen Zugriffsrechten zugeteilt werden.

Die SNMP-Software Net-SNMP unterstützt VACM. Ein praktisches Beispiel für die Konfiguration von VACM in einem Net-SNMP Agenten ist im Kapitel 6.5.4.1 zu finden.

4.5 Erweiterungsmodelle für den SNMP-Agenten

Um auf Managementinformationen zugreifen zu können benötigt ein SNMP Agent entsprechende Schnittstellen zu den Ressourcen, die diese Managementinformationen stellen. Diese Schnittstellen können und müssen nicht immer Teil der Implementierung des Agenten selbst sein.
Ein Problem ergibt sich vor allem, wenn mehrere voneinander unabhängige Anwendungsprogramme auf einem netzwerkfähigen Gerät über eine Netzwerkschnittstelle Managementinformationen via SNMP bereitstellen wollen. Diese können nicht alle parallel einen eigenen SNMP-Agenten betreiben, da auf einem netzwerkfähigen Gerät für gewöhnlich nur ein einziger SNMP-Agent bereitstehen sollte, mit dem ein Manager interagieren kann. Um die Managementinformationen der Anwendungsprogramme trotzdem via SNMP einem Manager verfügbar zu machen, benötigt es in diesem Fall einen SNMP Master-Agenten, über den verschiedene Sub-Agenten, die dann Teil der Implementierung der verschiedenen Anwendungsprogramme sind, mit dem Manager kommunizieren. Dem Manager bleibt diese verteilte Struktur des Agenten verborgen. Er sieht nur den SNMP Master-Agenten mit dem er via SNMP kommuniziert. Der Master-Agent liest die vom Manager kommenden SNMP-Nachrichten aus und leitete sie, entsprechend den angesteuerten Managementinformationen, an den jeweiligen Sub-Agenten in einem speziellen Protokollformat weiter.

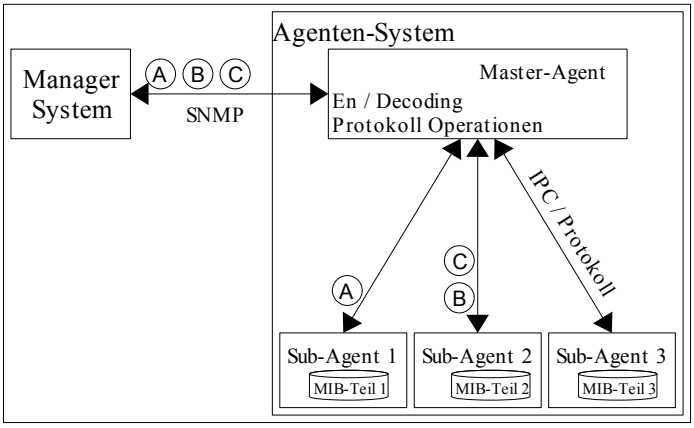


Abb. 4.23: Master/Sub-Agenten Architektur

Zwei plattformunabhängige Erweiterungsprotokolle für SNMP-Agenten werden hier nun kurz vorgestellt. Dabei handelt es sich um das ältere „SNMP Multiplexing Protocol“ (SMUX) und das modernere „Agent eXtensibility Protocol“ (AgentX).

4.5.1 SMUX

Das SNMP Multiplexing Protocol (SMUX) ist ein älteres Modell zur Einbindung von Programmen als Sub-Agenten an SNMP Master-Agenten. Es wurde bereits 1991 im RFC 1227 spezifiziert. Es handelt sich dabei um ein Netzwerk-Protokoll das standardmäßig den Port 199 benutzt. Der SNMP Agent benutzt SMUX um an einem „User-Level-Prozess“ SNMP Anfragen zu richten. Die Sub-Agenten werden hier meist als SMUX Peers bezeichnet. Zum Beispiel kann die Routing-Software Quagga via SMUX über einen SNMP Agenten vom Manager aus angesteuert werden. Die Sicherheit gleicht der in SNMPv1 und -v2c, ein „Community-Name“ dient als Passwort und wird standardmäßig zusammen mit allen anderen Daten im Klartext übertragen.

4.5.2 AgentX

Das „Agent eXtensibility Protocol“ (AgentX) wurde in den RFCs 2741 und 2742 spezifiziert und Anfang des Jahres 2000 veröffentlicht. Es handelt sich dabei um ein moderneres Erweiterungsprotokoll für SNMP Agenten. AgentX verfügt über ein effizientes Nachrichten-Format und eine effiziente Kodierung, unabhängig von der ASN.1/BER Kodierung. AgentX kennt mehrere Möglichkeiten der Kommunikation zwischen Sub- und Master-Agent, zum Beispiel über Netzwerk, via TCP und Port 705, sowie lokal über IPC (Interprozesskommunikation). AgentX überlässt die Sicherheit dem Master-Agenten, der den Zugriff auf die AgentX-Schnittstelle verwaltet.

4.6 Zusammenfassung

In diesem Kapitel wurde der Aufbau und die Funktionsweise von SNMP ausführlich vorgestellt. Die theoretischen Grundlagen von SNMP, die in diesem Kapitel beschrieben sind, erleichtern das Verständnis für die Funktion der SNMP-Implementierung Net-SNMP, die im weiteren Verlauf der Arbeit noch vorgestellt wird, und zeigen auch die grundsätzlichen Möglichkeiten des Einsatzes von SNMP im Netzwerkmanagement auf.

5 Netzwerk-Simulationen mit Virtual Network User Mode Linux

In diesem Kapitel wird das Netzwerk-Simulationsprogramm Virtual Network User Mode Linux (VNUML) vorgestellt. Das Programm erlaubt die Simulation selbst größerer Linux-Rechnernetze, mit Schwerpunkt auf das Testen und Analysieren darin laufender Netzwerk-Anwendungen. An einem Beispiel wird der Aufbau einer Netzwerk-Simulation und die Funktionsweise von VNUML dargestellt. Die Virtualisierungs-Software User Mode Linux (UML), auf der VNUML basiert wird näher betrachtet um die Vorgehensweise von VNUML und die Möglichkeiten, die dieser Netzwerk-Simulator bietet, besser verständlich zu machen. Auch auf die Routing-Software Quagga, die „dynamisches Routing“ in den VNUML Netzwerk-Simulationen erlaubt, wird näher eingegangen.

5.1 Einleitung

Virtual Network User Mode Linux (VNUML) wurde von Fermín Galán und David Fernández am Telematics Engineering Department (DIT) an der Technischen Universität von Madrid im Rahmen des Euro6IX Forschungsprojektes entwickelt. Dieses Projekt beschäftigt sich mit der Einführung von IPv6 in Europa. VNUML entstand ursprünglich um IPv6 Netzwerk-Szenarios auf Linux-Rechnern mit den Zebra/Quagga Routing-Diensten zu testen.
Das VNUML-Projekt ist im Internet unter der Adresse „http://jungla.dit.upm.es/~vnuml“ zu finden.
VNUML wird von seinen Autoren als „Allzweckvirtualisierungswerkzeug“ bezeichnet, da sich auf schnellem und einfachem Weg Linux-Rechnernetze beliebiger Topologie, in denen beliebige Linux-Software laufen kann, simulieren lassen. Diese Netzwerk-Simulationen stellen in erster Linie eine praxisnahe Umgebung für das Entwickeln, Analysieren und Testen von Linux Netzwerk-Anwendungen dar. Über eine VNUML eigene Beschreibungssprache wird die Topologie eines Netzwerks, die beteiligten Linux-Rechner und ihre Netzwerkverbindungen zueinander, beschrieben. Aus der Beschreibung wird die entsprechende Netzwerk-Simulation aufgebaut. VNUML basiert auf der Linux-Kernel Virtualisierung „User Mode Linux“ (UML), das die virtuellen Maschinen innerhalb einer Netzwerk-Simulation generiert. UML ermöglicht das Emulieren eines voll funktionalen Linux-Rechners. Das UML-Projekt ist ein Open-Source Projekt und unabhängig von der VNUML Entwicklung. Ursprünglich diente UML lediglich dem Testen von Software in einer sicheren aber praxisnahen Linux-Betriebssystemumgebung. Mittlerweile wird UML auch recht häufig als „Honeypot“ in Netzwerken eingesetzt um Hacker-Angriffe abzulenken und zur Analyse auf sich zu ziehen oder auch um eine virtuelle Server-Umgebung zu betreiben.

5.2 VNUML-Grundlagen

Im folgenden werden nun die Komponenten von VNUML und einige Begrifflichkeiten erläutert. Die Begriffe werden von den VNUML-Autoren selbst und auch im weiteren Verlauf dieser Arbeit verwendet.

- **Host** (Host-System)
  Hiermit ist der meist physikalische Linux-Rechner in dem VNUML läuft gemeint. Auf einem Host wird ein Netzwerk-Szenario, welches aus mehreren virtuellen Maschinen bestehen kann, simuliert. Auch der Host selbst kann Teil eines Netzwerk-Szenarios sein. Ein solches Netzwerk-Szenario kann sich nicht über mehrere Hosts verteilen. Allerdings können mehrere Netzwerk-Szenarios, die auf verschiedenen Hosts simuliert werden, über real existierende Netzwerkverbindungen zu einem größeren Netzwerk zusammengeschlossen werden. VNUML richtet auf dem Host für eine Netzwerk-Simulation virtuelle Netzwerkschnittstellen ein, über die auch ein Management-Netzwerk zu den Rechnern der Simulation aufgebaut wird.

- **UML-Rechner** (virtuelle Maschine)
  Dies sind die virtuellen Linux-Rechner, die VNUML für eine Simulation generiert und zu einem Netzwerk verbindet. Sie werden auf Basis von User Mode Linux erzeugt. Aus Sicht des Host-Systems sind UML-Rechner normale Anwendungsprozesse. Ein UML-Rechner besteht aus einem UML-Kernel und einem UML-Root-Dateisystem. In das Root-Dateisystem kann beliebige Linux Software installiert werden, die dann während einer Netzwerk-Simulation verwendet werden kann. UML-Rechner funktionieren auch unabhängig von VNUML. Während der Installation von VNUML 1.5 werden die beiden Verzeichnisse „/usr/local/share/vnuml/kernels/“ und „/usr/local/share/vnuml/filesystems/“ angelegt, in die dann die jeweilige Komponente des UML-Rechners abgelegt werden kann.

- **Virtuelles Netzwerk**
  Die virtuellen Netzwerke die VNUML erzeugt sind Netzwerke, die aus mehreren UML-Rechnern innerhalb eines Host-Systems bestehen. Software-Bridges verbinden die Netzwerkschnittstellen der einzelnen UML-Rechner miteinander oder mit real existierenden Netzwerkschnittstellen des Hosts. So kann ein virtuelles Netzwerk auch an ein real existierendes Netzwerk angeschlossen werden. Die Netzwerkverbindungen zwischen den einzelnen UML-Rechnern sind stark abstrahiert und stellen lediglich idealisierte Leitungsverbindungen dar. Sie besitzen keine Merkmale realer Netze, wie zum Beispiel Latenzschwankungen. Ihre Leistung ist hauptsächlich von den Leistungsressourcen des Hosts abhängig.

- **Szenario**
  Ein VNUML Netzwerk-Szenario wird durch die VNUML eigene Beschreibungssprache beschrieben und in einer XML-Datei abgelegt. Dabei wird unter anderem die Anzahl der virtuellen Maschinen, deren Verbindungen zueinander und die daraus resultierende Netzwerktopologie festgelegt. Anwendungsprogramme, die während einer Simulation auf den einzelnen UML-Rechnern ausgeführt werden sollen, und Dateien, die vom Host aus auf die UML-Rechner kopiert werden sollen, sind ebenfalls Bestandteil der Szenario-Beschreibung. Somit kann ein Szenario nicht nur die Topologie einer Netzwerk-Simulation umschreiben, sondern auch Kommandos definieren, die den Ablauf der Simulation bestimmen.
  VNUML in der Version 1.5 enthält einige Beispiel-Szenarios, die während der Installation in das Verzeichnis „/usr/local/share/vnuml/examples/“ abgelegt werden.

- **Simulation**
  VNUML simuliert ein Szenario. Mit dem Start einer Simulation wird die in einem Szenario beschriebene Netzwerktopologie aus UML-Rechnern im Host aufgebaut. Anwendungsprogramme, die in den UML-Rechnern installiert sind, können während einer Simulation, über im Szenario definierte Simulations-Kommandos, gestartet und gestoppt werden. Über solche Simulations-Kommandos können auch Dateien vom Host aus auf einzelne UML-Rechner kopiert werden, die dort zum Beispiel der Konfiguration der Anwendungsprogramme dienen. Mit dem Beenden der Simulation werden die einzelnen UML-Rechner heruntergefahren und das virtuelle Netz im Host wird abgebaut. Eine Netzwerk-Simulation aus UML-Rechnern ist auch ohne den Einsatz von VNUML möglich. VNUML vereinfacht jedoch die Vorgehensweise dabei erheblich und versteckt die komplexen technische Details hinter der einfachen VNUML-Sprache.

- **Management Netzwerk**
  Hierbei handelt sich um virtuelle Netzwerkverbindungen zwischen dem Host und den einzelnen UML-Rechnern der Netzwerk-Simulation. VNUML verwendet in den UML-Rechnern standardmäßig die „eth0“-Netzwerkschnittstelle für die Management-Netzwerkverbindung zum Host und IP-Adressen aus dem IP-Adressbereich 192.168.0.0/16. Durch das Management-Netzwerk ist es VNUML während einer Simulation erst möglich, über die im Szenario definierten Simulations-Kommandos, Dateien auf die einzelnen UML-Rechner zu kopieren und installierte Anwendungsprogramme zu verwenden. VNUML verwendet für das Management der UML-Rechner standardmäßig SSH (Secure Shell).

- **Virtuelle Netzwerkschnittstelle**
  Eine virtuelle Netzwerkschnittstelle ist ein reines Anwendungsprogramm, das eine Netzwerkschnittstelle lediglich emuliert. Beim Start einer Netzwerk-Simulation erzeugt VNUML die im Szenario festgelegte Anzahl an virtuellen Netzwerkschnittstellen in den jeweiligen UML-Rechnern und verbindet sie über Software-Bridges, entsprechend der im Szenario definierten Netzwerktopologie. Sämtliche Netzwerkschnittstellen und Bridges eines VNUML Szenarios sind im Host abgebildet und können auch dort während der laufenden Simulation konfiguriert und zum Beispiel mit dem Befehl „ifconfig“ angezeigt werden. Mit Protocol-Analyzern, wie „Ethereal“ oder „Tcpdump“, können die virtuellen Netzwerkschnittstellen und ihr Datenverkehr im Host und innerhalb des jeweiligen UML-Rechners analysiert werden.

VNUML baut aus der Beschreibung des Netzwerk-Szenarios die Simulation auf. Es startet die UML-Rechner, erzeugt deren virtuelle Netzwerkschnittstellen und verbindet sie zu einem virtuellen Netzwerk innerhalb des Host-Systems. Zwischen dem Host und den UML-Rechnern baut VNUML ein virtuelles Management-Netzwerk auf, über das die UML-Rechner während einer Simulation angesteuert werden können.

Im Bezug auf die Erstellung eines Szenarios und die Durchführung der Simulation heben die Autoren von VNUML zwei Komponenten hervor.

- **VNUML-Sprache**
  Diese VNUML eigene Beschreibungssprache ist eine XML-Sprache mit der die Netzwerk-Szenarios für die VNUML Simulationen beschrieben werden. Ein Netzwerk-Szenario ist innerhalb einer XML-Datei über die XML-Tags der VNUML-Sprache definiert. Eine Ausführliche Referenz über Syntax und Semantik der VNUML-Sprache ist auf der Internetseite von VNUML und im Quelltext-Verzeichnis des Programms zu finden. Über die VNUML-Sprache lässt sich nicht nur die Netzwerk-Topologie, mit der Anzahl der Netzwerkschnittstellen, deren IP-Adressen und den statischen Routen festlegen, sondern auch speziellere Eigenheiten der an der Simulation beteiligten UML-Rechner, die ihnen eine bestimmte Funktion im simulierten Netzwerk zuteilen. Bislang wurde die VNUML-Sprache mit jeder weiteren Version von VNUML um Funktionen erweitert.

- **VNUML-Parser**
  Der VNUML-Parser ist der Kern von VNUML und das eigentliche Simulationsprogramm. Es handelt sich dabei im wesentlichen um ein Perl-Script, welches das in der VNUML-Sprache beschriebene Szenario aus der XML-Datei ausliest und entsprechend den Vorgaben daraus, unter Mithilfe externer Software wie den „Usermode-Utilities“, den „Bridge-Utils“ und den „Linux Net-Tools“, die gewünschte Netzwerk-Simulation aufbaut. Über den VNUML-Parser wird die laufende Simulation auch verwaltet. Der komplette Ablauf einer Netzwerk-Simulation kann über den VNUML-Parser und der im Szenario definierten Simulations-Kommandos gesteuert werden. Einzelne Zustände einer Simulation können so immer wieder erreicht und durchgespielt werden.

Um eine Simulation über einem Szenario erfolgreich zu starten, muss die VNUML-Sprache, in der das Szenario beschrieben ist, die gleiche Version besitzen, wie der VNUML-Parser, mit dem die Simulation durchgeführt wird.

VNUML verwendet UML-Rechner zum Aufbau der Netzwerk-Simulationen. UML-Rechner funktionieren auch unabhängig von VNUML und können zu virtuellen Netzwerken zusammengeschlossen werden. VNUML versteckt jedoch die komplexen Arbeitsschritte, die dafür notwendig sind, hinter seiner relativ einfachen Beschreibungssprache und macht so den Einsatz von UML-Rechnern für immer wieder rekonstruierbare Netzwerk-Simulationen schnell, dynamisch, überschaubar und komfortable.
Die UML-Rechner sind nicht im eigentlichen VNUML-Programm enthalten und müssen zusätzlich beschafft werden. Die VNUML-Autoren stellen auf ihrerer Internetseite für ihre beiliegenden Beispiel-Szenarios UML-Rechner zum Download zur Verfügung. Diese können meist einfach und ausreichend erweitert und modifiziert werden. Es ist allerdings auch möglich UML-Rechner komplett selbst zu erstellen.

UML-Rechner bestehen aus zwei Komponenten die auch VNUML für seine Funktion benötigt und die in jedem Netzwerk-Szenario explizit angegeben werden müssen.

- **User Mode Linux Kernel** (UML-Kernel)
  Ein UML-Kernel ist ein Anwendungsprogramm, das die Virtualisierung eines Linux-Kernels darstellt. Seit der Vanilla Kernel Version 2.6.0 ist der UML-Kernel auch Bestandteil der offiziellen Linux-Kernel Entwicklung und Teil des Kernel-Quellcodes. Es handelt sich hierbei um einen „normalen“ Linux-Kernel, der als UML-Kernel kompiliert wurde. Der UML-Kernel hat, im Gegensatz zum „normalen“ Linux-Kernel, nur eine abstrakte, rudimentäre Hardware-Unterstützung da er als ein gewöhnliches Anwendungsprogramm im Host-System läuft. Auf der Internetseite des VNUML Projektes sind einige UML-Kernel für verschiedene VNUML-Versionen zu finden, die die VNUML-Autoren zum durchspielen ihrer Beispiel Netzwerk-Szenarios bereitstellen.

- **User Mode Linux Root-Dateisystem** (UML-Root-Dateisystem)
  Beim Root-Dateisystem handelt sich um die System- und Programmsoftware eines Linux-Systems, das in einer Imagedatei abgelegt ist. Es handelt sich dabei unter anderem um das Verzeichnissystem und die allgemein verwendeten Systemanwendungen, die zur Funktion eines Linux-Betriebssystems notwendig sind. Natürlich kann auch weitere, beliebige Linux Software installiert werden. Grundsätzlich bietet ein solches UML-Root-Dateisystem die gleichen Funktionen, die auch eine „normale“ Installation eines Linux Betriebssystems, auf einem reellen Rechner, hat. Einschränkungen gibt es hauptsächlich bei allzu hardwarespezifischen Anwendungen, wie zum Beispiel Hardwaretreibern.

Verschiedene UML-Root-Dateisysteme für die unterschiedlichen VNUML-Versionen zum durchspielen der Beispiel-Szenarios stehen auf der VNUML Internetseite zum Download bereit. Diese basieren auf der Linux Distribution „Debian“ und können weitestgehend beliebig modifiziert und erweitert werden.

VNUML kann für eine Simulation auf verschiedene UML-Rechner unterschiedlicher „Bauart“ zurückgreifen. Es können also verschiedene UML-Kernel und dazugehörige UML-Root-Dateisysteme, die zusammen die einzelnen UML-Rechner einer VNUML Netzwerk-Simulation bilden, in einem Szenario angegeben werden. Die von den VNUML-Autoren bereitgestellten UML-Kernel und UML-Root-Dateisysteme sind von der verwendeten VNUML-Version abhängig.

Bei der VNUML Installation für die praktische Ausarbeitung dieser Studienarbeit wurden keine speziellen Einstellungen vorgenommen, weswegen der Installationsanleitung auf der Internetseite des VNUML-Projektes gefolgt werden kann, da die Installation sich von Version zu Version auch etwas unterscheidet. Für die praktische Ausarbeitung wurde VNUML in der Version 1.5 verwendet. Hier werden nun keine näheren Angaben zur Installation von VNUML selbst gegeben. Stattdessen wird direkt mit der Erläuterung der Durchführung einer VNUML-Simulation fortgefahren.
Allgemeine Hinweise zur Installation von Anwendungsprogrammen in die UML-Rechner werden im Kapitel 5.6, zur Installation der Routing-Software Quagga im Kapitel 5.7 und zur Installation der SNMP-Software Net-SNMP in Kapitel 6.3, behandelt. Eine Anleitung zum Erstellen eines UML-Rechners auf Basis der Linux-Distribution Gentoo ist im Anhang dieser Ausarbeitung, im Kapitel 7, zu finden.

5.3 Durchführung einer VNUML Simulation

Im folgenden wird nun der Aufbau eines Szenarios und die Durchführung einer Simulation an einem einfachen Beispiel beschrieben.
Die VNUML Autoren empfehlen die Durchführung einer Netzwerk-Simulation in drei Phasen zu unterteilen.

- **Entwurfs Phase**
  Zuallererst muss ein Entwurf des Netzwerk-Szenarios erstellt werden. Die Anzahl der UML-Recher, die das Netzwerk aufspannen und wie sie miteinander verbunden sein sollen, also die Topologie des Netzes, muss feststehen. Den einzelnen Netzwerkschnittstellen der UML-Rechner müssen IP-Adressen zugeordnet werden. Auch muss feststehen, welche Anwendungsprogramme auf den einzelnen UML-Rechnern ausgeführt werden sollen und welche Funktion die Rechner somit in dem simulierten Netzwerk einnehmen. Ziel der Entwurfs Phase ist zum Beispiel eine möglichst detailreiche, grafische Ansicht der Topologie des zu simulierenden Rechnernetzes.

- **Implementierungs Phase**
  Nachdem das zu simulierende Netzwerk-Szenario entworfen wurde beginnt die Implementierung des Szenarios in VNUML-Sprache. Syntax und Semantik der VNUML-Sprache sind in der VNUML Sprach-Referenz festgelegt. Die Version der VNUML-Sprache die verwendet wird, muss die gleiche sein, wie die Version des verwendeten VNUML-Parsers. Ziel der Implementierungs Phase ist eine XML-Datei, in der eine komplette und gültige Beschreibung des zu simulierenden Netzwerk-Szenarios in VNUML-Sprache abgelegt ist.

- **Ausführungs Phase**
  Während der Ausführungs Phase findet die Netzwerk-Simulation statt. Der VNUML-Parser liest das implementierte Szenario aus der XML-Datei aus. Entsprechend den Vorgaben der Implementierung baut der VNUML-Parser die Simulation auf und führt sie durch.
  Die Ausführungs Phase kann in drei Stufen unterteilt werden, die jeweils im Bezug zu einer Befehlsoption des VNUML-Parsers stehen.

  1. Aufbau des Szenarios: Über die Befehlsoption -t startet der VNUML-Parser die Simulation. Die einzelnen UML-Rechner werden gestartet und die virtuellen Netzwerkschnittstellen werden erstellt und über Software-Bridges, entsprechend den Vorgaben des Szenarios, miteinander verbunden. IP-Adressen, statische Routen und andere Parameter werden in den einzelnen UML-Rechnern konfiguriert. Ist die Simulation fertig aufgebaut kann in die beteiligten UML-Rechner über aktivierte Terminals oder via SSH durch das Management-Netzwerk vom Host aus eingeloggt werden. Für die UML-Rechner, die VNUML für das Durchspielen seiner Beispiel-Szenarios bereitstellt, kann der Benutzer „root“ mit dem Passwort „xxxx“ verwendet werden.

2. Ausführen von Simulations-Kommandos: Über die Befehlsoption -x können die Simulations-Kommandos, die ebenfalls im Szenario bei den jeweiligen virtuellen Maschinen implementiert sein müssen, ausgeführt werden. Diese Simulations-Kommandos können zum Beispiel mit in den UML-Rechnern installierten Anwendungsprogrammen, eigenen Programm-Scripten oder sonstigen Linux-Befehlen verknüpft sein, die dann während der Simulation, nach Ausführung des Kommandos, auf den entsprechenden UML-Rechnern ausgeführt werden. So können zum Beispiel verschiedene Programme auf mehreren UML-Rechnern ungefähr gleichzeitig, vom Host aus über den VNUML-Parser, gestartet oder gestoppt werden. Auch der Vorgang des Kopierens von Dateien kann mit Simulations-Kommandos verknüpft werden, so dass bestimmte Dateien vom Host aus nur auf bestimmte UML-Rechner kopiert werden und dort zum Beispiel als Konfigurationsdateien für Anwendungsprogramme vorliegen.

3. Stoppen der Simulation: Über die Befehlsoption -d stoppt der VNUML-Parser die Simulation wieder. Die einzelnen UML-Rechner werden heruntergefahren und die virtuellen Netzwerkschnittstellen werden entfernt. Dabei wird eine Simulation so gestoppt, das die gegenwärtige Konfiguration der einzelnen UML-Rechner für den nächsten Start erhalten bleibt. Über die Befehlsoption -P wird die Simulation abgebrochen und all ihre Backup-Dateien werden gelöscht, so dass sie sich beim nächsten Start wieder im ursprünglichen Ausgangszustand befindet.

Nur der Vollständigkeit halber soll an dieser Stelle „vnumlgui“ erwähnt werden. Das Programm „vnumlgui“ ist eine Grafische Oberfläche für VNUML. Die Entwicklung geschieht aber bislang in einem von der VNUML-Entwicklung unabhängigen Open-Source Projekt, das im Internet auf der Entwicklerplattform „Sourceforge“ unter der Adresse „http://sourceforge.net/projects/vnumlgui“ zu finden ist. Zur Zeit der Ausarbeitung dieser Studienarbeit unterstützte „vnumlgui“ lediglich die Version 1.5 von VNUML vollständig. Der Vorteil von „vnumlgui“ ist, dass die drei Arbeits-Phasen zur Durchführung von Netzwerk-Simulationen mit VNUML über eine übersichtliche, grafische Oberfläche erledigt werden können und so nur ein abstraktes Hintergrundwissen über Syntax und Semantik der VNUML-Sprache vorhanden sein muss.

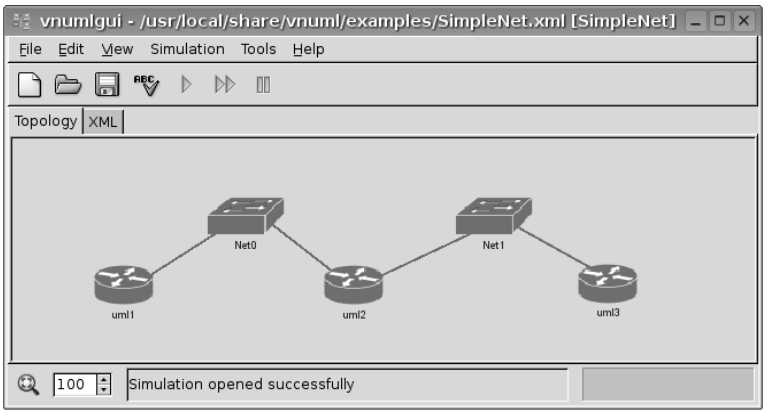


Abb. 5.1: vnumlgui, ein grafisches Benutzerinterface zu VNUML

5.3.1 Entwurfs Phase

In dieser Phase wird ein Szenario entworfen über dem später die Netzwerk-Simulation gestartet wird. Das hier verwendete Beispiel Szenario, mit der Bezeichnung „SimpleNet“, besteht lediglich aus drei UML-Rechnern, die in Reihe geschaltet sind. Die UML-Rechner bekommen die einfachen Namen „uml1“, „uml2“ und „uml3“. Der UML-Rechner „uml1“ wird im Verlauf dieser Studienarbeit als SNMP-Manager eingesetzt, der UML-Rechner „uml2“ als SNMP-Agent. Auf dem Rechner „uml2“ läuft noch ein RIP-Router. Der UML-Rechner „uml3“ hat keine speziellen Funktionen. Zwischen „uml1“ und „uml2“ ist das 10.0.0.0/30 er Netz und zwischen „uml2“ und „uml3“ das 164.1.2.0/30 er Netz aufgespannt. Die beiden UML-Rechner „uml1“ und „uml3“ sind als End-Rechner des Netzwerks zu sehen, während „uml2“ als Router, die Schnittstelle zwischen den beiden ist.

Die Topologie des Netzes sieht somit wie folgt aus:

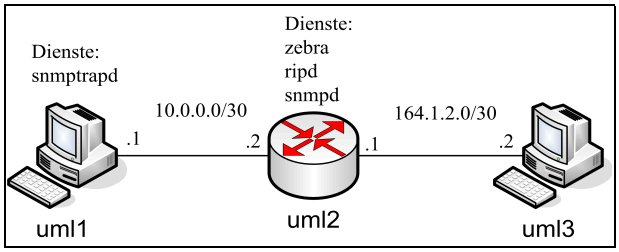


Abb. 5.2: Topologie des Netzwerk-Szenarios „SimpleNet“

Damit der UML-Rechner „uml1“ den UML-Rechner „uml3“ erreichen kann und umgekehrt, muss auf beiden eine Default-Route mit der IP-Adresse der jeweils verbundenen „uml2“ Netzwerkschnittstelle eingerichtet sein. Auf dem UML-Rechner „uml2“ verwaltet der RIP-Router (ripd) der Routing-Software Quagga die Routen. Der Zebra Router (zebra) schreibt sie in die Kernel-Tabelle des UML-Rechners. Damit IP-Pakete auch weiter geleitet werden können muss auf diesem UML-Rechner die IP-Weiterleitung (IP-Forwarding) eingeschaltet sein. Zusätzlich läuft auf „uml2“ noch der SNMP-Agent (snmpd) der Software Net-SNMP. Auf dem UML-Rechner „uml1“ läuft noch der Net-SNMP Trap-Server (snmptrapd), eine Serveranwendung, die SNMP-Meldungen empfangen kann. Der UML-Rechner „uml3“ hat in diesem Beispiel keine weitere Funktion.
In dieser Abbildung nicht zu sehen ist das Management-Netzwerk, das der VNUML-Parser zwischen dem Host und den einzelnen UML-Rechnern automatisch aufbaut um diese per SSH während einer Simulation zu verwalten.

5.3.2 Implementierungs Phase

In dieser Phase wird das Szenario nun implementiert. Die drei beteiligten UML-Rechner, ihre Netzwerkschnittstellen und deren IP-Adressen werden in VNUML-Sprache in einer XML-Datei mit Namen „SimpleNet.xml“ implementiert. Anwendungsprogramme, die auf den einzelnen UML-Rechnern während der Simulation ausgeführt werden sollen, und Dateien, die auf die UML-Rechner kopiert werden sollen, müssen mit entsprechenden Simulations-Kommandos verknüpft werden.

Die Syntax der VNUML-Sprache, die Tags, werden von den VNUML-Autoren in drei Typen unterteilt.

- **Strukturelle Tags**
  Diese Tags organisieren die XML-Datei in verschiedene Sektionen, die wiederum weitere, spezifischere Tags beinhalten. Der VNUML-Parser verwendet sie während der Ausführungs Phase, um sich an der Struktur der XML-Datei zu orientieren. Zu den Strukturellen Tags gehören <vnuml>, <global>, <vm>, <boot>, <net> und einige weitere.

- **Topologie Tags**
  Diese Tags beschreiben die Topologie des Szenarios. Der VNUML-Parser verwendet sie während des Startens und Stoppens der Netzwerk-Simulation und ignoriert sie während der Ausführungs Phase. Zu den Topologie Tags gehören <kernel>, <filesystem>, <con0>, <xterm>, <ssh_key>, <automac>, <if>, <ipv4>, <forwarding>, <route>, <host_mapping> und noch einige mehr.

- **Simulations Tags**
  Diese Tags verwalten den Ablauf der Simulation. Der VNUML-Parser verwendet sie während der Ausfühungs Phase und ignoriert sie während des Startens und Stoppens der Simulation. Simulations-Kommandos können auch während einer laufenden Simulation in die Szenario-Implementierung hinzugefügt oder verändert werden, ohne die Stabilität der Simulation zu gefährden. Zu den Simulations Tags gehören <exec>, <filetree> und einige mehr.

Zusätzlich gibt es noch Tags wie zum Beispiel <simulation_name> und <version>, die zu den Simulations sowie auch zu den Topologie Tags gehören.

Die Beschreibung eines Szenarios in der XML-Datei beginnt xml-typisch mit der Angabe der verwendeten XML-Version und der Kodierung sowie der Angabe in welcher Datei die Definition der VNUML-XML-Tags zu finden ist. Die Struktur der XML-Tags der VNUML-Sprache ist in Document Type Definition (DTD) in der

Datei „vnuml.dtd“ definiert. Danach folgt auch schon das erste XML-Tag der VNUML-Sprache. Das Tag <vnuml> ist das Wurzel-Tag und beinhaltet alle weiteren Definitionen eines Szenarios. Direkt unterhalb folgt das Tag <global>, das alle Einstellungen beinhaltet, die sich „global“ auf das Szenario beziehen und sich allgemein auf die Simulation auswirken. So ist hier zum Beispiel die zu verwendende VNUML-Parser Version und der Name des Szenarios zu finden. Da der VNUML-Parser die einzelnen UML-Rechner vom Host aus via SSH verwaltet, ist hier Angegeben, in welcher Datei der SSH-Schlüssel des VNUML-Benutzers auf dem Host zu finden ist. Das Tag <automac> legt fest, dass alle MAC-Adressen der Netzwerkschnittstellen der beteiligten UML-Rechner automatisch generiert werden und nicht manuell gesetzt werden müssen. Das Tag <host_mapping> trägt alle an der Simulation beteiligten UML-Rechner in die Datei „/etc/hosts“ auf dem Host-System ein. So können die einzelnen UML-Rechner nicht nur über ihre IP-Adresse, sondern auch über ihren Hostnamen vom Host-System aus über das Management-Netzwerk erreicht werden. Die Tags <filesystem> und <kernel> geben die beiden Bestandteile der UML-Rechner, UML-Root-Dateisystem und UML-Kernel, an. Diese Tags werden für jede virtuelle Maschine der Netzwerk-Simulation angegeben, so dass auch UML-Rechner verschiedener „Bauart“ für die einzelnen virtuellen Maschinen eingesetzt werden können. Innerhalb des „global“ Bereiches kann aber auch, wie im Listing 5.1.1 dargestellt, über das Tag <default_kernel> ein UML-Kernel und über das Tag <default_filesystem> ein UML-Root-Dateisystem angegeben werden, die für alle virtuellen Maschinen des Netzwerk-Szenarios gelten und somit alle UML-Rechner der Simulation aufbauen. Darüber hinaus gibt es noch viele weitere Tags, die die Simulation global beeinflussen können.
Im Beispiel müssen an Stelle der Pfadangaben mit den einfachen Hochzeichen absolute Pfadangaben gemacht werden, die zum UML-Kernel und zum UML-Root-Dateisystem führen und hier nur der Übersicht wegen durch Platzhalter ersetzt wurden.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">

<vnuml>
 <global>
  <version>1.5</version>
  <simulation_name>SimpleNet</simulation_name>
  <ssh_key>/root/.ssh/identity.pub</ssh_key>
  <automac />
  <host_mapping />
  <default_filesystem type="cow">'/DirPath/UML-Root-Filesystem'</default_filesystem>
  <default_kernel>'/DirPath/UML-Kernel'</default_kernel>
 </global>
```

Listing 5.1.1: Inhalt der Datei SimpleNet.xml, Teil 1

Im Anschluss an die „globale“ Beschreibung folgt über das Tag <net> die Definition der einzelnen Netze zwischen den virtuellen Maschinen des Szenarios. Hier im Beispiel werden die beiden Netze „Net0“ und „Net1“ definiert. Das XML-Attribut „mode“ gibt die softwaretechnische Umsetzung an, mit der die Netze erstellt werden. Hier im Beispiel werden sie über das Programm „brctl“, das aus den „Bridge-Utils“ stammt, erstellt. Nachteil hierbei ist, dass die Rechte des Superusers („root“) für den Start der Netzwerk-Simulation auf dem Host benötigt werden. Es gibt auch andere Möglichkeiten die UML-Rechner der Netzwerk-Simulation miteinander zu verbinden, zum Beispiel über die Mode-Anweisung „uml_switch“, wodurch ein Programm der „Usermode-Utilities“ zum erstellen der virtuellen Netzverbindungen verwendet wird.

```
 <net name="Net0" mode="virtual_bridge" />
 <net name="Net1" mode="virtual_bridge" />
```

Listing 5.1.2: Inhalt der Datei SimpleNet.xml, Teil 2

Nun folgt die Beschreibung der einzelnen virtuellen Maschinen über das Tag <vm>. Dieses Tag umschließt alle Einstellungen, die die einzelnen virtuellen Maschinen betreffen. Auch der UML-Rechner kann hier speziell für eine virtuelle Maschine definiert werden. Da er aber in diesem Beispiel bereits global für die komplette Simulation definiert wurde muss er hier nicht mehr angegeben werden. Das Tag <boot> beinhaltet Boot-Optionen der einzelnen UML-Rechner. Hier im Beispiel wird jeder UML-Rechner mit einem eigenen X-Terminal „xterm“ gestartet, über das sich ein Benutzer direkt in den UML-Rechner einloggen kann. Anstatt eines X-Terminals können hier auch andere Terminals oder Login-Möglichkeiten definiert werden. Diese Angabe ist aber nicht zwingend, da sich Benutzer generell auch via SSH über das VNUML Management-Netzwerk in die einzelnen UML-Rechner einloggen können. Das Tag <if> definiert die Netzwerkschnittstellen der UML-Rechner. Die Schnittstellen-Nummer, die IP-Adresse und das angeschlossene Netz müssen angegeben werden. Die MAC-Adressen der Netzwerkschnittstellen werden hier wegen des vorherigen Tags <automac/> automatisch generiert. Die Netzwerkschnittstelle mit der Nummer 0 ist standardmäßig die Verbindung des UML-Rechners zum VNUML Management-Netzwerk und damit zum Host und wird hier deswegen nicht vergeben. Die Tags <forwarding> und <route> beeinflussen das Routing verhalten des UML-

Rechners. Auf den beiden UML-Rechern „uml1“ und „uml3“ wird über das Tag <route> ein Default-Gateway eingerichtet. Auf dem UML-Rechner „uml2“ wird über <forwarding> die IP-Weiterleitung aktiviert. Das Routing auf „uml2“ übernimmt während der Simulation der Quagga RIP-Router „ripd“. Die Tags <filetree> und <exec> sind Tags der Ausführungs Phase der VNUML-Simulation. Hier werden Linux-Befehle, die auf den einzelnen UML-Rechnern ausgeführt werden sollen, und Kopieraufforderungen von Dateien mit Simulations-Kommandos verknüpft. Das Tag <filetree> definiert ein Verzeichnis auf dem Host-System dessen Inhalt, bei Eingabe des entsprechenden Simulations-Kommandos, in das mit dem XML-Attribut „root“ angegebene Verzeichnis, auf den UML-Rechner, kopiert wird. Dies vereinfacht zum Beispiel den Umgang mit Konfigurationsdateien, die über einen handlichen Editor an einer zentralen Stelle auf dem Host erstellt und verändert werden können und bei Bedarf dann einfach an ihren Platz auf den oder auch die UML-Rechner kopiert werden. Hier im Beispiel wird das Simulations-Kommando „start“ für alle Kopiervorgänge definiert. Nachdem die Netzwerk-Simulation fertig aufgebaut ist bekommen die benötigten Dienste der einzelnen UML-Rechner ihre Konfigurationsdatei nach Aufruf des Simulations-Kommandos. Über das Tag <exec> werden weitere Simulations-Kommandos definiert, die mit Programmausführungen verknüpft sind. So können einzelne Programme, an zentraler Stelle über den VNUML-Parser vom Host aus, ausgeführt werden. Alle SNMP-Dienste können auf den UML-Rechnern während der Simulation vom Host aus mit dem Kommando „startSNMP“ gestartet und mit den Kommando „stopSNMP“ wieder beendet werden. Ebenso kann mit den Routing-Diensten verfahren werden, die mit „startRouter“ gestartet und „stopRouter“ beendet werden. Mit dem Simulations-Kommando „readLogs“ kann der Inhalt aller Log-Dateien der verwendeten Dienstprogramme ausgelesen und auf dem Host angezeigt werden. Hier wird nun auch noch das Simulations-Kommando „mgmt“ definiert, das von „uml1“ aus die Managementinformationen aller Objekte von „uml2“ abfragt, die sich innerhalb des „internet“ MIB-Moduls (OID: 1.3.6.1.2) befinden. Das Ergebnis wird auf dem Host angezeigt.

```
 <vm name="uml1">
  <boot><con0>xterm</con0></boot>
  <if id="1" net="Net0">
       <ipv4 mask="255.255.255.252" >10.0.0.1</ipv4>
  </if>
  <route type="inet" gw="10.0.0.2">default</route>
  <filetree root="/etc/snmp" when="start">/'UML1ConfigfilesDir'/snmp</filetree>
  <exec seq="startSNMP" type="verbatim">snmptrapd -c /etc/snmp/snmptrapd.conf</exec>
  <exec seq="stopSNMP" type="verbatim">killall snmptrapd</exec>
  <exec seq="readLogs" type="verbatim">cat /var/log/snmptrapd.log</exec>
  <exec seq="mgmt" type="verbatim">snmpwalk -v1 -cpublic 10.0.0.2 .1.3.6.1.2</exec>
 </vm>

 <vm name="uml2">
  <boot><con0>xterm</con0></boot>
  <if id="1" net="Net0">
       <ipv4 mask="255.255.255.252" >10.0.0.2</ipv4>
  </if>
  <if id="2" net="Net1">
       <ipv4 mask="255.255.255.252" >164.1.2.1</ipv4>
  </if>
  <forwarding type="ip"/>
  <filetree root="/etc/snmp" when="start">/'UML2ConfigfilesDir'/snmp</filetree>
  <filetree root="/etc/quagga" when="start">/'UML2ConfigfilesDir'/quagga</filetree>
  <exec seq="startSNMP" type="verbatim">snmpd -c /etc/snmp/snmpd.conf </exec>
  <exec seq="stopSNMP" type="verbatim">killall snmpd</exec>
  <exec seq="startRouter" type="verbatim">zebra -f /etc/quagga/zebra.conf -d</exec>
  <exec seq="startRouter" type="verbatim">ripd -f /etc/quagga/ripd.conf -d</exec>
  <exec seq="stopRouter" type="verbatim">killall ripd </exec>
  <exec seq="stopRouter" type="verbatim">killall zebra</exec>
  <exec seq="readLogs" type="verbatim">cat /var/log/snmpd.log</exec>
  <exec seq="readLogs" type="verbatim">cat /var/log/quagga/zebra.log</exec>
  <exec seq="readLogs" type="verbatim">cat /var/log/quagga/ripd.log</exec>
 </vm>

 <vm name="uml3">
  <boot><con0>xterm</con0></boot>
  <if id="1" net="Net1">
       <ipv4 mask="255.255.255.252" >164.1.2.2</ipv4>
  </if>
  <route type="inet" gw="164.1.2.1">default</route>
 </vm>

</vnuml>
```

Listing 5.1.3: Inhalt der Datei SimpleNet.xml, Teil 3

Damit ist die Implementierung des Szenarios „SimpleNet“ abgeschlossen und die Ausführungs Phase mit der Durchführung der Netzwerk-Simulation kann angegangen werden.

5.3.3 Ausführungs Phase

Wie bereits erwähnt wird die Ausführungs Phase der Netzwerk-Simulation in drei Stufen unterteilt. Zu Beginn der Ausführungsphase wird das implementierte Szenario durch den VNUML-Parser zu einer Netzwerk-Simulation aufgebaut. Der VNUML-Parser überprüft vor dem Start die Implementierung des Szenarios in der XML-Datei nach Fehlern und bricht gegebenenfalls den Startvorgang ab.
Die Netzwerk-Simulation wird in dem Verzeichnis, in dem sich die XML-Datei mit dem implementierten Szenario befindet, über den folgenden Befehl gestartet.

```
host # vnumlparser.pl -t SimpleNet.xml -vB
```

Befehl 5.1: Starten der Netzwerk-Simulation „SimpleNet“

Die Befehlsoptionen -vB am Ende der Eingabe stehen für „Verbose“ und „Blocking“. Der „Verbose“ Mode aktiviert eine wesentlich ausführlichere Ausgabe während des Startens der Simulation um Fehler, die während des Startvorgangs auftreten, erkennen zu können. Der „Blocking“ Mode stellt das Starten der einzelnen UML-Rechner an das Ende des Startvorgangs der Simulation. Der VNUML-Parser bereitet so zuerst das Host-System komplett auf die bevorstehende Simulation vor und startet erst dann die UML-Rechner gleichzeitig. So kann es nicht passieren, dass die Netzwerk-Simulation halb aufgebaut abbricht. Außerdem erhöht dies die Geschwindigkeit des Startvorgangs erheblich, da der VNUML-Parser die UML-Rechner nicht einzeln startet und wartet bis ein UML-Rechner komplett hochgefahren ist bevor er den nächsten vorbereitet und startet.
Das komplette Netzwerk, inklusive Management-Netzwerk, das der VNUML-Parser für das Beispiel „SimpleNet“ aufbaut, sieht wie folgt aus.

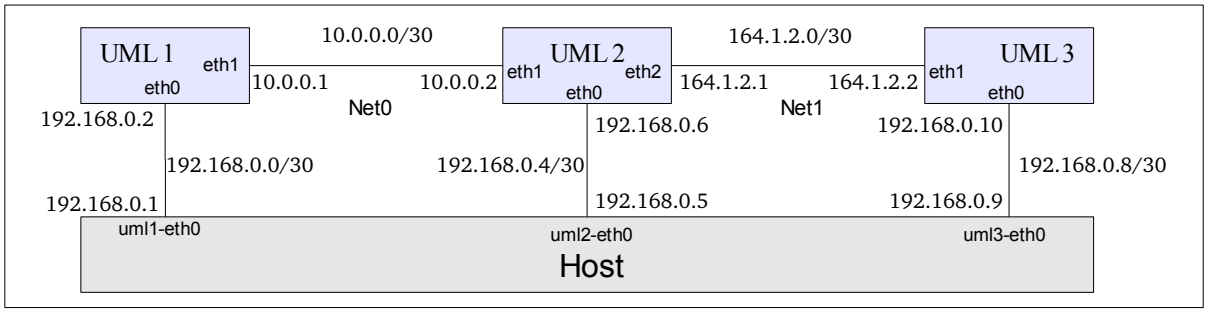


Abb. 5.3: Komplette virtuelle Netzwerk-Topologie der VNUML Simulation des „SimpleNet“-Szenarios

Das VNUML Management-Netzwerk zum verwalten der einzelnen UML-Rechner erhält automatisch die IP-Adressen aus dem Netz-Bereich 192.168.0.0/16. Dies kann aber über ein XML-Tag der VNUML-Sprache beeinflusst werden. VNUML in der Version 1.5 verwaltet die UML-Rechner ausschließlich via SSH, weswegen auf dem Host auch SSH zur Verfügung stehen muss. Funktioniert SSH oder das Management-Netzwerk nicht, können die folgenden Simulations-Kommandos des „SimpleNet“ Beispiels auch nicht ausgeführt werden.

Das Simulations-Kommando „start“ des Beispiels kopiert die Quagga und SNMP Konfigurationsdateien auf die jeweiligen UML-Rechner.

```
host # vnumlparser.pl -x start@SimpleNet.xml -v
```

Befehl 5.2: Kopieren der benötigten Konfigurationsdateien vom Host auf die UML-Rechner

Die nächsten beiden Simulations-Kommandos starten den Net-SNMP Trap-Server auf „uml1“ und den Net-SNMP Agenten sowie die Quagga Routing Dienste „zebra“ und „ripd“ auf „uml2“, in eben dieser Reihenfolge.

```
host # vnumlparser.pl -x startSNMP@SimpleNet.xml
host # vnumlparser.pl -x startRouter@SimpleNet.xml
```

Befehl 5.3: Start der Dienste auf den UML-Rechnern

Ausgaben der Simulations-Kommandos werden direkt in der Konsole des Hosts angezeigt. Ein kleiner Ausschnitt des Ergebnisses der SNMP-Anfrage von „uml1“ an die Management-Objekte der „internet“-Gruppe von „uml2“ über das Kommando „mgmt“ wird hier angezeigt.

```
host # vnumlparser.pl -x mgmt@SimpleNet.xml

SNMPv2-MIB::sysDescr.0 = STRING: Linux uml2 2.6.12.6-1m #5 Mon Jul 3 23:32:43 UTC 2006 i686
SNMPv2-MIB::sysObjectID.0 = OID: SNMPv2-SMI::dod.0.0.0.0.0.0.0.0
SNMPv2-MIB::sysUpTime.0 = Timeticks: (24358) 0:04:03.58
[ . . . ]
```

Befehl 5.4: Ausschnitt der Ausgabe des Simulations-Kommandos „mgmt“

Wurde ein Szenario mit seinen Simulations-Kommandos fertig durchgespielt und soll die Netzwerk-Simulation beendet werden, kann sie über die folgende Befehlsoption des VNUML-Parsers gestoppt werden.

```
host # vnumlparser.pl -d SimpleNet.xml  -vB
```

Befehl 5.5: Beenden einer Netzwerk-Simulation

Nach dem Stoppen der Simulation behält VNUML allerdings noch einige Backup-Dateien, die es zum Beispiel erlauben die beteiligten UML-Rechner und somit den letzten Zustand der Simulation zu speichern. Treten Fehler auf oder soll eine Simulation von neuem beginnen und in den Ausgangszustand versetzt werden, so kennt VNUML die Befehlsoption -P, die alle Backup-Dateien löscht und auch eine laufende Netzwerk-Simulation zum Abbruch zwingt.

```
host # vnumlparser.pl -P SimpleNet.xml  -v
```

Befehl 5.6: Abbruch einer Netzwerk-Simulation und Löschen aller Backup-Dateien

Die einfache Eingabe des Befehls „vnumlparser.pl“ listet alle möglichen Befehlsoptionen mit einer kurzen Erläuterung auf.

5.4 Das VNUML Management-Netzwerk

Da diese Studienarbeit das Management-Protokoll SNMP in einer VNUML Simulation zum Thema hat, wird hier auch näher auf das Management-Netzwerk, das VNUML zwischen dem Host und den UML-Rechnern aufbaut, eingegangen. VNUML in der Version 1.5 baut zwischen jedem UML-Rechner und dem Host eine „private“ Netzwerkverbindung auf, so dass jedem UML-Recher eine IP-Adresse auf dem Host eingerichtet wird, wie in Abbildung 5.3 zu sehen ist. Der VNUML-Parser verwendet für die Kommunikation mit den UML-Rechnern das SSH Protokoll. Deswegen benötigen die VNUML Netzwerk-Simulationen auch den SSH-Schlüssel des Host-Benutzers, der über das Tag <ssh_key> im Szenario implementiert sein muss.
Um Dateien auf die UML-Rechner zu kopieren oder Anwendungsprogramme auf den UML-Rechnern auszuführen, müssen nicht unbedingt entsprechende Simulations-Kommandos im Szenario definiert werden. Dank SSH können die UML-Rechner auch unabhängig vom VNUML-Parser konfiguriert werden. Dabei wird dem SSH Befehl einfach der auf dem UML-Rechner auszuführende Befehl angehängt.
Der folgende Befehl führt vom Host aus eine Verfolgung der Route von „uml1“ zu „uml3“ durch.

```
host # ssh root@192.168.0.2  'tracepath 164.1.2.2'
root@192.168.0.2's password: xxxx
 1:  uml1 (10.0.0.1)                       2.050ms pmtu 1500
 1:  10.0.0.2 (10.0.0.2)                   0.977ms
 2:  164.1.2.2 (164.1.2.2)                 1.049ms reached
    Resume: pmtu 1500 hops 2 back 2
```

Befehl 5.7: Routenverfolgung von „uml1“ zu „uml3“, vom Host via „ssh“ aus initiiert

Der Befehl „tracepath 164.1.2.2“ wird via SSH an den UML-Rechner „uml1“ gesendet, dort ausgeführt und das Ergebnis wieder via SSH an den Host gesendet und dort auf der Konsole ausgegeben. Mit SSH allein, ohne VNUML ist so auch eine Konfiguration der UML-Rechner vom Host aus möglich, ohne die Szenario Beschreibung ändern zu müssen.

Auch Dateien können via SSH, über den Befehl „scp“ (Secure Copy), auf einzelne UML-Rechner kopiert werden. So kann die Quagga Konfigurationsdatei des RIP-Routers, mit Namen „ripd.conf“, die sich auf dem Host im Verzeichnis „UML2Configs/quagga/“ befindet, wie folgt auf den UML-Rechner „uml2“ und dort in das Verzeichnis „/etc/quagga/“ kopiert werden.

```
host # scp UML2Configs/quagga/ripd.conf  root@192.168.0.6:/etc/quagga/
root@192.168.0.6's password: xxxx
ripd.conf                                     100% 293    0.3KB/s   00:00
```

Befehl 5.8: Kopieren der Datei „ripd.conf“ auf den UML-Rechner „uml2“ mittels „scp“

VNUML selbst benutzt im Hintergrund ebenfalls SSH um die UML-Rechner anzusteuern. Der Vorteil der definierbaren Simulations-Kommandos in den Szenarios gegenüber der direkten Anwendung von SSH ist, dass mehrere Befehle und Kopiervorgänge unter einem Kommando zusammengefasst und gleichzeitig für mehrere UML-Rechner ausgeführt werden können. Auch sind sie fest mit dem Szenario verbunden und damit kann die Simulation immer wieder mit den selben Simulations-Kommandos durch die gleichen Zustände geführt werden.

Mit SSH und dem darauf aufbauende VNUML-Management ist ein umfassendes Netzwerkmanagement einer VNUML Simulation und der beteiligten UML-Rechner möglich. VNUML verwendet SSH jedoch hauptsächlich im Sinne des Konfigurationsmanagements, zum Starten und Stoppen von Programmen. Zwar können auch Log-Dateien ausgelesen und auf dem Host angezeigt werden, wichtige Information darin müssten allerdings erst heraus gefiltert werden, um sie zum Beispiel statistisch oder in einer grafischen Leistungskurve zu verarbeiten.
Wird SNMP nicht nur in den UML-Rechnern, sondern auch auf dem Host verfügbar gemacht, so kann es als Erweiterung des VNUML-Managements dienen, das vor allem die Bereiche des Leistungs- und des Fehlermanagements sehr viel besser abdeckt. Zumal es einige auf SNMP Software wie Net-SNMP aufbauende Management-Programme gibt, die die Managementinformationen aus den SNMP-Nachrichten auslesen und grafisch oder statistisch direkt weiterverarbeiten können.

5.5 User Mode Linux Rechner

Die Basissoftware auf der VNUML aufbaut liefert User Mode Linux (UML). UML ist eine Kernel Virtualisierung, die ein vollständiges Linux-Betriebssystem emuliert. Prinzipiell handelt es sich bei UML um eine Portierung des Linux-Kernels auf seine eigene „Systemcall“-Schnittstelle. Dies ermöglicht es einen Linux-Kernel als normalen Prozess in einem laufenden Linux-Betriebssystem zu starten.
Ein UML-Rechner kann gut zum Testen von Linux-Software verwendet werden, da er nicht nur eine abgesicherte Linux-Betriebssystemumgebung bereitstellt, sondern auch die immer gleiche Linux-Betriebssystemumgebung, mit dem gleichen Ausgangszustand und den selben Einstellungen liefern kann.
Zu UML gehört das Softwarepaket „Usermode-Utilities“, das viele Linux-Distributionen unter diesem oder einem ähnlichen Namen in ihrer Software-Distributionsliste zur automatischen Installation bereitstellen. Das Softwarepaket enthält einige Anwendungsprogramme mit denen UML-Rechner modifiziert und konfiguriert werden können.
Die Software von UML wird im gleichnamigen Open-Source Projekt gepflegt, das im Internet unter der Adresse „http://user-mode-linux.sourceforge.net“ zu finden ist.

Das Linux-Betriebssystem, das ein UML-Rechner liefert, ist ähnlich aufgebaut wie das eines reellen Rechners. Ein gewöhnliches Linux-Betriebssystem kann in zwei Komponenten unterteilt werden.

- **Einem Linux-Kernel,**
  der hardwareabstrahierenden Schicht des Betriebssystems in der die Ansteuerung der Geräte sowie die Speicher- und Prozessverwaltung stattfindet.

- **Einem Linux Root-Dateisystem,**
  in dem sich alle Anwendungsprogramme für den Benutzer befinden und auch die grundlegenden System-Programme, die das Betriebssystem erst funktionsfähig machen.

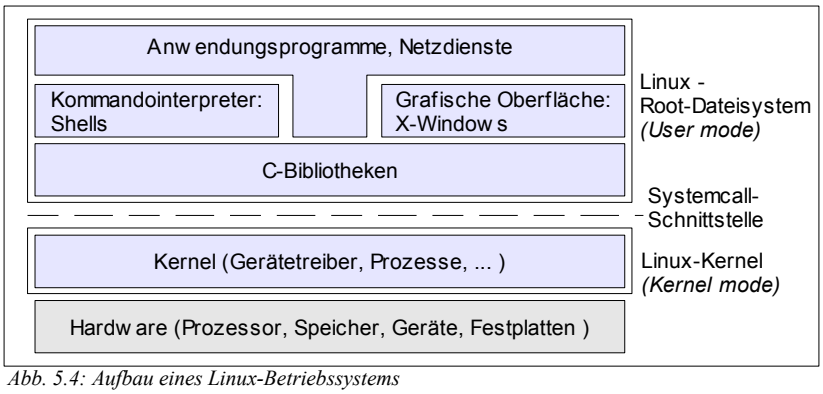


Abb. 5.4: Aufbau eines Linux-Betriebssystems

Beim starten eines Linux-Betriebssystems auf einem reellen Rechner wird zuerst der Linux-Kernel über den „Boot-Loader“ in den Hauptspeicher des Rechners geladen. Dieser Linux-Kernel befindet sich für gewöhnlich in der sogenannten Boot-Partition („/boot/“). Nach und nach steuert der Linux-Kernel nun die verschiedenen Hardwarekomponenten des Rechners an, um sie dem Betriebssystem zugänglich zu machen, indem er deren Treiberprogramme lädt. Die Schnittstellen zu den Geräten sind dann für gewöhnlich im Verzeichnis „/dev/“

(Devices) zu finden. So wird zum Beispiel die erste Partition auf der am „Primary Master IDE“ im Rechner angeschlossenen Festplatte unter „/dev/hda1“ in das Linux System angeschlossen, die zweite Partition unter „/dev/hda2“, usw. Die erste Partition auf der am „Secondary Master IDE“ im Rechner angeschlossenen Festplatte wird unter „/dev/hdb1“ angeschlossen. Die Partitionen müssen aber auch in das Betriebssystem eingebunden werden, sofern sie dafür vorgesehen sind. Unter Linux geschieht dies über die Datei „fstab“, die sich im Verzeichnis „/etc/“ befindet oder manuell über den Befehl „mount“. Verschiedene Festplattenpartitionen bekommen einen „Mountpunkt“ zugewiesen unter dem sie in das Betriebssystem eingebunden werden. Die Festplattenpartition in der das Linux Root-Dateisystem installiert ist, wird als Wurzelverzeichnis („/“) eingebunden. Ist dies geschehen und sind alle notwendigen Treiber geladen, können beliebige weitere Anwendungsprogramme gestartet werden, zum Beispiel ein X-Server für grafische Ausgaben.

Bei User Mode Linux, das ein Linux-Betriebssystem emuliert, läuft der Vorgang ähnlich ab. Auch hier werden die beiden Komponenten Linux-Kernel und Linux-Root-Dateisystem benötigt. Während der User Mode Linux Kernel speziell kompiliert werden muss gibt es beim User Mode Linux Root-Dateisystem kaum Unterschiede zum Root-Dateisystem einer gewöhnlichen Linux Installation. Allerdings ist das Root-Dateisystem für User Mode Linux innerhalb einer Imagedatei, einer Art virtuellen Festplattenpartition, abgelegt die vom UML-Kernel als UML Block Device („ubd“) geladen wird.

Ein User Mode Linux Kernel kann seit der Vanilla Kernel Version 2.6.0 aus dem original Linux-Kernel-Quellcode erstellt werden. Für ältere Kernel Versionen wird ein Patch benötigt. Der Vanilla Linux Kernel ist im Internet auf der Seite „http://www.kernel.org“ zu finden.

Um einen UML-Kernel zu erstellen muss allen kernelspezifischen Befehlen die Option „ARCH=um“ angehängt werden. Vor allem die folgenden Befehle sind für die Erstellung eines UML-Kernels zu erwähnen, die innerhalb des Kernel-Quellcode-Verzeichnisses ausgeführt werden müssen:

- **make menuconfig ARCH=um** (alternativ.: make xconfig ARCH=um, make gconfig ARCH=um)
  Hiermit wird eine grafische Oberfläche zum konfigurieren des UML-Kernels gestartet. Die fertige Konfiguration wird dann in die Datei „.config“ innerhalb des Kernel-Verzeichnis gespeichert.

- **make oldconfig ARCH=um**
  Ist eine Konfigurationsdatei („.config“) für die UML-Kernel Konfiguration bereits vorhanden, zum Beispiel legen die VNUML-Autoren ihren UML-Kerneln eine Konfigurationsdatei bei, so können die Einstellungen mit diesem Befehl in den neuen Kernel übernommen werden.

- **make ARCH=um**
  Mit diesem Befehl wird der Linux-Kernel als UML-Kernel kompiliert. Das Ergebnis ist ein Anwendungsprogramm, das unter dem Namen „linux“ oder „vmlinux“ im Verzeichnis zu finden ist.

- **make modules_install INSTALL_MOD_PATH='Root-Dateisystem-Mountpunkt' ARCH=um**
  Mit diesem Befehl werden die benötigten Kernel-Module in das Verzeichnis „'Root-Dateisystem-Mountpunkt'/lib/modules/'Kernel' “ installiert. Kernel-Module sind spezielle Programmteile des Kernels, die im laufenden Betrieb in den Speicher geladen und auch wieder daraus entfernt werden können.
  In dem Verzeichnis „Root-Dateisystem-Mountpunkt“ ist hier die Imagedatei, die das Root-Dateisystem des UML-Rechners beinhaltet, im Host-System eingebunden.

Die VNUML-Autoren stellen für die verschiedenen VNUML-Versionen unterschiedliche UML-Kernel auf ihrerer Internetseite zur Verfügung. Für die VNUML Version 1.5 bietet sich der UML-Kernel „linux-2.6.10-1m“ an. Meist sind die UML-Kernel von VNUML als Tarball gepackt und können über den Befehl „tar xfjv UML-Kernelname.tar.bz2“ entpackt werden. Neben dem UML-Kernel selbst liegt noch die Konfigurationsdatei des Kernels, die Kernel-Module sowie eine Textdatei mit Erläuterungen der weiteren Vorgehensweise bei.

An dieser Stelle sei angemerkt, dass mit der Veröffentlichung der Vanilla Linux-Kernel Version 2.6.13 eine Veränderung in der Konfiguration des Linux Geräte Managements stattgefunden hat.
Geräte (Devices) werden unter Linux über sogenannte Gerätedateien angesteuert, die für gewöhnlich im Verzeichnis „/dev/“ zu finden sind. Bei älteren Linux-Kerneln, mit Versionen kleiner 2.6.13, kann unter anderem noch eingestellt werden, dass das Anschließen von Geräten in das System mit der älteren „devfs“-Kernelunterstützung, dem „Linux Device Filesystem“, geschieht, bei den neueren Kerneln kann hier nur noch die „udev“-Userlandunterstützung, das „Dynamic Device Management“ eingestellt werden. Mit der Kernel Version 2.6.14 wurde „devfs“ ganz aus dem Linux-Kernel entfernt. Beim Geräte-Management mit

„devfs“ werden die System-Schnittstellen zu den angeschlossenen Geräten vom Linux-Kernel fest vorgegeben und sind während des Betriebs unveränderlich. So ist die erste Partition der am „Primary Master IDE“ angeschlossenen Festplatte unter der Schnittstellen-Bezeichnung „/dev/hda1“, die zweite Partition unter „/dev/hda2“, die erste Partition der am „Primary Slave IDE“ angeschlossenen Festplatte unter „/dev/hdc1“, usw. zu finden und unveränderlich. Für Festplatten, die während des laufenden Betriebs nicht gewechselt werden, stellt dies kein Problem dar. Aber für sogenannte „Pluggable Devices“ oder „Hotplug Geräte“, die zum Beispiel über USB, IEE1394 oder hot-swappable PCI an das System angeschlossen werden, kann hier keine genaue Aussage darüber gemacht werden, unter welcher System-Schnittstelle der Kernel das betreffende Gerät anschließt, da die Schnittstellen der zeitlichen Reihenfolge des Anschließens nach vergeben werden. So werden USB-Geräte meist unter der Schnittstelle „/dev/sda1“, „/dev/sda2“, usw angeschlossen, wobei keine genaue Aussage darüber gemacht werden kann, welches USB-Gerät, ob Drucker, Scanner oder Digitalkamera, hinter einer Schnittstellen-Bezeichnung steht.
Beim Geräte Management mit „udev“ kann jedes Gerät, nach dem es erkannt wurde, automatisch einer vorher vom Benutzer definierten und speziell benannten System-Schnittstelle zugeordnet werden. Mit „udev“ ist somit eine Erstellung und Entfernung sowie eine konsequente Benennung dynamischer Gerätedateien möglich.
User Mode Linux kann mit dem Geräte Management „udev“ umgehen, VNUML in der Version 1.5 allerdings nicht. Der Hauptunterschied hier ist, dass „devfs“ die Imagedateien, in denen sich unter anderem das UML-Root-Dateisystem befindet, unter den Schnittstellen „/dev/ubd/0“, „/dev/ubd/1“, usw, ansteuert, während „udev“ „/dev/ubda“, „/dev/ubdb“, usw, dafür verwendet. Daher muss bei einem Wechsel des Kernels ab der Version 2.6.13 und darüber hinaus zum einen „udev“ anstelle von „devfs“ im Root-Dateisystem verwendet werden und zum anderen müssen die entsprechenden Einträge in der Datei „/etc/fstab“, in der die „Mountpunkte“ angegeben sind, verändert werden.

```
#FS            Mountpoint    Type          Options     dump/pass
/dev/ubd/0     /             ext2          defaults       0 1
/dev/ubd/1     /mnt/vnuml    ext2          defaults       0 0
proc           /proc         proc          defaults       0 0
devpts         /dev/pts      devpts        mode=0622      0 0
```

Listing 5.2: Inhalt der Datei „/etc/fstab“ des UML-Root-Dateisystems bei Verwendung von „devfs“

VNUML selbst hat nun noch das Problem, dass innerhalb des VNUML-Parser Perl-Scriptes „vnumlparser.pl“, an einer Stelle mit dem Eintrag „root=/dev/ubd/0“ das zu verwendende Wurzelverzeichnis für die UML-Rechner angegeben ist. Sollen neuere Linux-Kernel, größer gleich der Version 2.6.13 verwendet werden, die nur noch „udev“ unterstützen, so müsste diese Stelle zu „root=/dev/ubda” verändert werden.

Die zweite Komponente des UML-Rechners, das UML-Root-Dateisystem, ist weitaus schwieriger selbst zu erstellen, da die Installation und Konfiguration des Linux-Grundsystems meist manuell durchgeführt werden muss und nicht auf grafische Installationsroutinen zurückgegriffen werden kann, wie zum Beispiel bei Linux Installationen auf reellen Rechnern. Hier können verschiedene Linux Distributionen verwendet und so auf die Vorteile der Einzelnen zurückgegriffen werden. So bieten einige Linux Distributionen, wie Debian oder Gentoo, „Online-Installer“ an. Bei diesen beiden Distributionen kann eine sehr große Anzahl von Anwendungsprogrammen mit Hilfe einer Installationsroutine von Datei-Servern aus dem Internet heruntergeladen und automatisch in das lokale Root-Dateisystem installiert werden.
Die VNUML-Autoren verwenden für ihre UML-Root-Dateisysteme die Debian Linux Distribution. Für die VNUML Version 1.5 ist das Root-Dateisystem mit Namen „root_fs_tutorial-0.2.3“ auf den VNUML Internetseiten zu finden. Die UML-Root-Dateisysteme, die die VNUML-Autoren bereitstellen, sind meist mit bzip2 gepackt und können über den Befehl „bzip2 -d Root-Dateisystemname.bz2“ entpackt werden.

Mit den beiden UML-Rechner Komponenten der VNUML Internetseite, dem UML-Kernel „linux-2.6.10-1m“ und dem Root-Dateisystem „root_fs_tutorial-0.2.3“ kann nun bereits mit dem folgenden Befehl ein einzelner UML-Rechner gestartet werden.

```
host # ./linux-2.6.10-1m ubd0=uml_cow_fs,root_fs_tutorial-0.2.3 con0=null con=xterm
```

Befehl 5.9: Starten eines UML-Rechners

Nach der Eingabe fährt der UML-Rechner hoch und ein X-Terminal öffnet sich. Bei dem Root-Dateisystem von VNUML kann immer mit dem Benutzernamen „root“ und dem Passwort „xxxx“ eingeloggt werden. Heruntergefahren, bzw. gestoppt, wird der UML-Rechner mit dem Befehl „shutdown -h now“, eingegeben innerhalb des X-Terminals.

Die Imagedatei mit dem Root-Dateisystem des UML-Rechners, hier „root_fs_tutorial-0.2.3“, wird über die Befehlsoption ubd0 angegeben. Sie ist damit die erste Partition des UML Block-Devices (ubd).

UML bietet beim Umgang mit Root-Dateisystemen den COW-Mechanismus an, dem hier kurz Beachtung geschenkt werden soll. COW steht für „Copy On Write“ und gibt der Basis-Imagedatei, die das Root-Dateisystem enthält, eine Art Backup-Funktion, da dann nicht schreibend, sondern nur lesend darauf zugegriffen werden kann. Alle Veränderungen innerhalb des Root-Dateisystems werden in einer COW-Imagedatei abgelegt. Existiert diese Datei beim Start nicht, wird sie automatisch angelegt. Die COW-Imagedatei ist so groß wie die Basis-Imagedatei, enthält aber fast nur Nullen, weswegen sie platzsparend auf dem Host-System gespeichert werden kann.

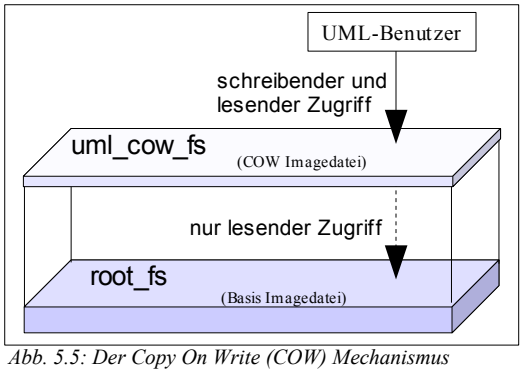


Abb. 5.5: Der Copy On Write (COW) Mechanismus

Die Veränderungen der Daten im UML-Rechner haben nur Bestand, wenn die zugehörige COW-Imagedatei beim Start immer mit angegeben wird. Beim Verlust der COW-Imagedatei gehen auch alle Veränderungen verloren. Die COW-Imagedatei kann nur mit ihrer Basis-Imagedatei und nicht mit einer anderen oder alleine funktionieren. Die Angabe einer neuen COW-Imagedatei startet den UML-Rechner mit dem ursprünglichen, unveränderten Root-Dateisystem, so dass der Ausgangszustand des UML-Rechners wieder hergestellt ist.

UML bietet mit dem Tool „uml_moo“, das Bestandteil der „Usermode-Utilities“ ist, die Möglichkeit eine COW-Imagedatei mit seiner Basis-Imagedatei zu verschmelzen und so die Veränderungen im UML-Root-Dateisystem bleibend zu übernehmen.

```
host # uml_moo  uml_cow _fs root_fs_tutorial-0.2.3
```

Befehl 5.10: Verschmelzen einer Basis-Imagedatei mit der zugehörigen COW-Imagedatei

Mehrere UML-Rechner können zur gleichen Zeit gestartet werden und auf das gleiche Root-Dateisystem, das sich in der selben Basis-Imagedatei befindet, zugreifen. Jedem UML-Rechner wird so nur eine eigene COW-Imagedatei erstellt, in die er seine Daten ablegt. Da alle UML-Rechner nur lesend auf das original Root-Dateisystem in der Basis Imagedatei zugreifen, kann es hier auch nicht zu Konflikten zwischen den einzelnen UML-Rechnern kommen. Meist genügt ohnehin bei mehreren UML-Rechnern die zu einem Netzwerk zusammengeschlossen werden, wie VNUML das automatisch macht, ein einzelnes Root-Dateisystem, so dass alle UML-Rechner über die gleichen Anwendungsprogramme verfügen, die dann lediglich unterschiedlich konfiguriert werden müssen. Da eine Basis-Imagedatei mit einem Root-Dateisystem meist mehrere 100 Megabyte groß ist und die dazugehörige COW-Imagedatei im Vergleich dazu sehr klein ist, nur nach und nach größer wird und eigentlich nie ihre maximale Größe, die der Größe der Basis-Imagedatei entspricht, erreicht, spart der COW-Mechanismus auch eine Menge Speicherkapazität auf dem Host ein.

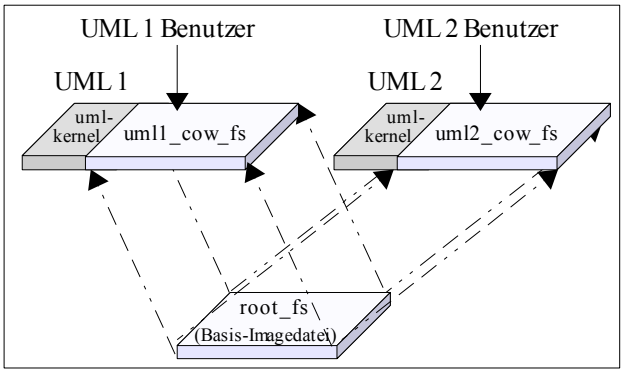


Abb. 5.6: Mehrere COW-Imagedateien über einer Basis-Imagedatei

Neben dem COW-Mechanismus gibt es noch andere Möglichkeiten, die auch über die VNUML-Sprache eingestellt werden können. Zum Beispiel gibt es die Möglichkeit, die Basis-Imagedatei beim Starten des UML-Rechners komplett zu kopieren und nicht wie bei COW, nur die Veränderungen. Vorteil hier ist, dass jeder UML-Rechner eine eigenständige Imagedatei mit enthaltenem Root-Dateisystem bekommt. Der Nachteil ist, dass eventuell sehr viel Speicherplatz benötigt wird. Auf diese Möglichkeiten soll hier nicht näher eingegangen werden, da der COW-Mechanismus auch standardmäßig in VNUML eingestellt ist.

VNUML ist ein Perl-Programm, das ein virtuelles Netzwerk aus UML-Rechnern aufbaut. Dabei verwendet es neben den Programmpaketen „Usermode-Utilities“ und „Bridge-Utils“ hauptsächlich Linux Standard Software. Die eigentliche Innovation ist User Mode Linux, das den virtuellen Linux Rechner im reellen Linux Rechner erst möglich macht. VNUML versteckt im Grunde lediglich eine Reihe recht komplizierter Befehlsfolgen, die auf dem Host wie auch in den einzelnen UML-Rechnern ausgeführt werden müssen, um ein solches virtuelles Netzwerk zu erstellen, hinter der vergleichsweise einfachen VNUML-Beschreibungssprache. VNUML macht das Erstellen von virtuellen Linux Rechnernetzen somit einfach, dynamisch und übersichtlich und bietet zudem eine gute Schnittstelle für die Anbindung einer grafischen Benutzeroberfläche, wie „vnumlgui“.

Ein virtuelles Netzwerk aus UML-Rechnern kann auch gänzlich ohne VNUML aufgebaut werden. Um die Funktionsweise von VNUML besser verständlich zu machen wird im folgenden ein virtuelles Netzwerk aus zwei UML-Rechnern ohne den Einsatz von VNUML aufgebaut. Dabei wird allerdings die gleiche Software verwendet und auf die selbe Art und Weise vorgegangen, wie auch VNUML in der Version 1.5 vorgeht.

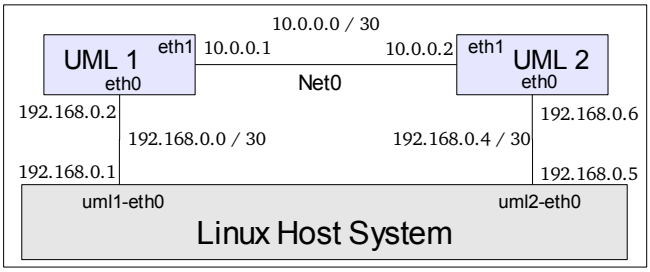


Abb. 5.7: Manuell erstellter Teil des „SimpleNet“-Szenarios

Das virtuelle Netzwerk, das hier aufgebaut wird, ist ein Teil des Netzwerks, das bereits zuvor als das Beispiel „SimpleNet“ mit VNUML aufgebaut wurde. Es handelt sich um die beiden darin enthaltenen UML-Rechner „uml1“ und „uml2“ mit den entsprechenden Netzwerkverbindungen zueinander und zum Host. Um dieses virtuelle Rechnernetz aufzubauen muss ein Teil der Befehle auf dem Host eingegeben werden und ein anderer Teil in den UML-Rechnern, nach dem sie gestartet worden sind. VNUML in der Version 1.5 geht dabei so vor, das es eine weitere Imagedatei erstellt, in die es eine Script-Datei mit verschiedenen Befehlen, zum Beispiel zum setzen der Netzwerkschnittstellen oder der „Default Route“ einträgt. Diese Imagedatei wird beim Hochfahren des UML-Rechners als „Sekundäre Partition“ in die Verzeichnis-Struktur eingebunden und die darin enthaltene Script-Datei, mit den Befehlen, wird ausgeführt. Auf diese Art und Weise können quasi beliebige, individuelle Einstellungen automatisch vorgenommen werden. Die folgenden Befehle, die hier verwendet werden, zeigt auch VNUML in der Version 1.5 während des Startvorgangs an, wenn die Befehlsoption -v (verbose) angegeben wird. Alle Befehle werden auf dem Host-System ausgeführt.

Zu Beginn müssen alle virtuellen Netzwerkschnittstellen des Netzwerks auf dem Host eingerichtet werden. In diesem Beispiel vier Stück. Dafür wird das Programm „tunctl“ verwendet, das dem Softwarepaket „Usermode-Utilities“ beiliegt und auf dem Host „TUN/TAP Devices“ generiert, den IP-Adressen zugeordnet werden können. Direkt im Anschluß wird den virtuellen Netzwerkschnittstellen, die nur in den UML-Rechnern aktiv sein sollen, die IP-Adresse 0.0.0.0, die ein Platzhalter für „alle Adressen“ ist, zugeordnet und der „promiscuous Modus“ aktiviert. Ist dieser eingeschaltet werden alle Pakete vom Netzwerk empfangen, unabhängig davon, ob sie an diese Netzwerkschnittstelle adressiert sind. Dies geschieht mit dem Programm „ifconfig“, das ein Linux Standard Programm zum konfigurieren von Netzwerkschnittstellen ist. Die virtuellen Netzwerkschnittstellen, die zusätzlich für das Management-Netzwerk verwendet werden, also eine Netzwerkverbindung zum Host bieten, erhalten im weiteren Verlauf die entsprechenden IP-Adressen aus dem festgelegten IP-Adressbereich. Mit dem Programm „brctl“ wird die Netzwerk-Bridge, hier mit Namen „Net0“, generiert. Das „Spanning Tree Protocol“ (STP) wird abgeschaltet da es standardmäßig aktiviert ist, hier aber nicht benötigt wird. Normalerweise dient STP der Vermeidung redundanter Pfade (Schleifen) im Netzwerk. Auch der Bridge wird die IP-Adresse 0.0.0.0 mit aktivem „promiscuous Modus“ zugeteilt.

```
tunctl -t uml1-eth0 -f /dev/net/tun
tunctl -t uml1-eth1 -f /dev/net/tun
ifconfig uml1-eth1 0.0.0.0 promisc up

tunctl -t uml2-eth0 -f /dev/net/tun
tunctl -t uml2-eth1 -f /dev/net/tun
ifconfig uml2-eth1 0.0.0.0 promisc up

brctl addbr Net0
brctl stp Net0 off
ifconfig Net0 0.0.0.0 promisc up
```

Befehl 5.11.1 : Aufbau des virtuellen Netzes im Host

Den virtuellen Netzwerkschnittstellen, die eine Verbindung zum Host aufbauen, werden IP-Adressen des festgelegten IP-Adressbereichs zugeteilt. Vom UML-Rechner „uml1“ ist der Host dann unter der IP-Adresse 192.168.0.1 zu erreichen und von „uml2“ unter der IP-Adresse 192.168.0.5 .

```
ifconfig uml1-eth0 192.168.0.1 netmask 255.255.255.252 up
ifconfig uml2-eth0 192.168.0.5 netmask 255.255.255.252 up
```

Befehl 5.11.2 : Konfiguration der Management-Netzwerkschnittstellen des Hosts

Die Gegenstellen mit den IP-Adressen 192.168.0.2 und 192.168.0.6 müssen später, im entsprechenden UML-Rechner, nach dessen Start, eingerichtet werden. Die virtuellen Netzwerkschnittstellen der UML-Rechner, die miteinander in Verbindung stehen sollen und somit das virtuelle Netz aufspannen, müssen der Software-Bridge

zugeteilt werden. Hier werden die beiden Netzwerkschnittstellen, die jeweils als „eth1“-Schnittstelle in den beiden UML-Rechnern aktiv sind, der Bridge „Net0“ zugeteilt. Damit ist die grundlegende Voraussetzung für eine erfolgreiche Netzwerk-Kommunikation zwischen den beiden Schnittstellen gegeben.

```
brctl addif Net0 uml1-eth1
brctl addif Net0 uml2-eth1
```

Befehl 5.11.3 : Hinzufügen der Netzwerkschnittstellen zur Bridge

Die virtuellen Netzwerkschnittstellen benötigen allerdings noch ihre IP-Adressen, die sie im virtuellen Netzwerk besitzen sollen. Diese müssen ebenfalls innerhalb der UML-Rechner gesetzt werden.

Als nächstes werden die Imagedateien erstellt, mit deren Hilfe VNUML die Konfiguration der einzelnen UML-Rechner der Netzwerk-Simulation durchführt. Jedem UML-Rechner wird eine eigene Imagedatei für die Konfiguration erstellt, in der die individuellen Daten abgelegt werden. Die Imagedatei ist mit einem Megabyte zwar recht klein, beinhaltete aber auch nur kleine Datenmengen. Sie wird mit dem Programm „dd“ (Disk Dump) erstellt und anschließend mit dem „ext2“-Filesystem formatiert. Um Dateien in diese Imagedatei ablegen zu können, muss sie in die Verzeichnis-Struktur des Host-Systems, hier in das Verzeichnis „mntuml1“ für den UML-Rechner „uml1“, eingebunden werden.

```
dd if=/dev/zero of=uml1-opt-fs count=1 bs=1M
mke2fs -F uml1-opt-fs
mkdir mntuml1
mount -o loop uml1-opt-fs mntuml1
```

Befehl 5.11.4: Erstellen der Imagedatei für das Konfigurations-Script des UML-Rechners „uml1“

Der öffentliche SSH-Schlüssel des Benutzers auf dem Host, der mit dem virtuellen Netzwerk arbeitet, wird ebenfalls in die eingebundene Imagedatei kopiert. Im VNUML-Szenario wird er über das Tag <ssh_key> angegeben. Hier wird der SSH-Schlüssel des Benutzers „root“ aus der Datei „identity.pub“ in die Datei „authorized_keys“ innerhalb der neu erstellten Imagedatei kopiert. Dies erlaubt ein reibungsloses Arbeiten mit SSH, da der Benutzer „root“ des Hosts nun bereits bei den SSH-Diensten der UML-Rechner autorisiert ist.

```
cat /root/.ssh/identity.pub >> mntuml1/authorized_keys
```

Befehl 5.11.5: Kopieren des SSH-Schlüssels

Im Anschluß daran wird das Bash-Script „umlboot“ innerhalb der Imagedatei erstellt. In diese Script-Datei werden die Linux Befehle eingetragen, die den UML-Rechner für seine Funktion im virtuellen Netzwerk konfigurieren. Über den Befehl „ifconfig“ werden den Netzwerkschnittstellen der UML-Rechner die IP-Adressen zugeteilt.
VNUML geht auf die selbe Weise vor. Es liest aus den VNUML-Tags des Szenarios, innerhalb der Definition der virtuellen Maschinen, die gesetzten Einstellungen aus, generiert die entsprechenden Befehle für die Konfiguration und legt sie mit Hilfe des Befehls „echo“ in das Bash-Script „umlboot“ ab. Die Möglichkeiten die sich hier bieten um die UML-Rechner automatisch zu konfigurieren sind quasi unbegrenzt und mit jeder VNUML Version kommen mehr Einstellungsmöglichkeiten, die über die VNUML-Sprache konfiguriert werden können, hinzu.

```
echo "#!/bin/bash" >> mntuml1/umlboot
echo "hostname uml1" >> mntuml1/umlboot
echo "/sbin/ifconfig eth0 192.168.0.2 netmask 255.255.255.252 up" >> mntuml1/umlboot
echo "/sbin/ifconfig eth1 0.0.0.0 promisc up" >> mntuml1/umlboot
echo "/sbin/ifconfig eth1 10.0.0.1 netmask 255.255.255.252 up" >> mntuml1/umlboot
echo "/sbin/route add default gw 10.0.0.2" >> mntuml1/umlboot

chmod a+x mntuml1/umlboot
umount mntuml1
```

Befehl 5.11.6 : Erstellen der Script-Datei „umlboot“ für die Konfiguration des UML-Rechners „uml1“

Im Anschluß daran müssen noch die Zugriffsrechte des „umlboot“-Scriptes über den Befehl „chmod“ gesetzt werden, so dass die Datei ausgeführt werden darf. Während des Bootvorgangs des UML-Rechners wird das Script und damit die darin enthaltenen Befehle ausgeführt. Dafür muss der UML-Rechner für VNUML entsprechend konfiguriert sein. So muss in der Datei „/etc/fstab“, die die Dateisystemtabelle enthält, die Imagedatei als zweite Partition angegeben sein, wie dies im Listing 5.2 dargestellt ist, und es muss aus einem „Runlevel“ heraus auf die Script-Datei „umlboot“ referenziert werden, damit diese während des Bootens des UML-Rechners automatisch ausgeführt wird. Sämtliche Befehle, die innerhalb der Datei „umlboot“ abgelegt sind, könnten aber auch „von Hand“ innerhalb des UML-Rechners ausgeführt werden. Hier geht es aber darum den Mechanismus, den VNUML in der Version 1.5 verwendet, verständlich zu machen.

Zum Schluss muss die Imagedatei aus dem Host-System gelöst werden, sonst kann es beim Start des UML-Rechners zu Problemen kommen.
Wie bereits erwähnt erstellt VNUML für jeden an der Netzwerk-Simulation beteiligten UML-Rechner eine solche Imagedatei mit individuell konfiguriertem Bash-Script „umlboot“ für die Konfiguration.
Hier folgt die Konfiguration des UML-Rechners „uml2“.

```
dd if=/dev/zero of=uml2-opt-fs count=1 bs=1M
mke2fs -F uml2-opt-fs
mkdir mntuml2
mount -o loop uml2-opt-fs mntuml2
cat /root/.ssh/identity.pub >> mntuml2/authorized_keys
echo "#!/bin/bash" >> mntuml2/umlboot
echo "hostname uml2" >> mntuml2/umlboot
echo "/sbin/ifconfig eth0 192.168.0.6 netmask 255.255.255.252 up" >> mntuml2/umlboot
echo "/sbin/ifconfig eth1 0.0.0.0 promisc up" >> mntuml2/umlboot
echo "/sbin/ifconfig eth1 10.0.0.2 netmask 255.255.255.252 up" >> mntuml2/umlboot

chmod a+x mntuml2/umlboot
umount mntuml2
```

Befehl 5.11.7: Erstellen der Image- und Script-Datei für die Konfiguration des UML-Rechners „uml2“

Nun ist die Konfiguration beendet und die UML-Rechner können gestartet werden.

Ist beim Start einer Simulation mit VNUML die Befehlsoption für den „Blocking Mode“ (-B) angegeben, so wartet auch VNUML diese obige Konfiguration für alle beteiligten UML-Rechner ab, bevor sie dann parallel gestartet werden. Ist der „Blocking Mode“ nicht aktiviert, so wird jeder UML-Rechner einzeln konfiguriert und direkt im Anschluß daran gestartet.
Der Befehl zum Starten eines UML-Rechners wurde zuvor schon einmal im Befehl 5.9 gezeigt. Hier werden die beiden UML-Rechner mit den Imagedateien, die jeweils als erste und zweite Partition eingebunden werden, gestartet. Auch die Netzwerkschnittstellen, über die ein UML-Rechner verfügen soll, müssen beim Start des UML-Rechners angegeben werden. Dabei wird der verwendete Mechanismus (hier tuntap), die zugehörige Netzwerkschnittstelle im Host sowie die MAC-Adresse angegeben.
Der UML-Kernel heißt in diesem Beispiel einfach nur „linux“ und wird wie ein gewöhnliches Anwendungsprogramm unter Linux gestartet. Das Root-Dateisystem ist hier in der Imagedatei „root_fs“ gespeichert. Dieses wird mit dem bereits beschriebenen COW-Mechanismus angesteuert, so dass nur lesend auf die Basis-Imagedatei zugegriffen wird und alle Veränderungen in die jeweils angegebene COW-Imagedatei geschrieben werden. Dies ermöglicht auch, dass die beiden UML-Rechner parallel auf das gleiche Root-Dateisystem zugreifen können und durch Löschen eines COW-Images der Ursprungszustand des UML-Rechners wieder hergestellt ist.
Die optionale Angabe „root=/dev/ubd/0“ zeigt an, in welcher Imagedatei sich das Root-Dateisystem mit dem Wurzelverzeichnis befindet. Die Befehlsoption „umid“ konfiguriert die Schnittstelle des UML-Rechners für die Management-Konsole über die User Mode Linux zum ansteuern eines laufenden UML-Rechners verfügt. Mit dem Programm „uml_mconsole“ aus den „Usermode-Utilities“ können die UML-Rechner dann während der Laufzeit angesteuert und konfiguriert werden. VNUML legt die Konsolen-Schnittstelle für gewöhnlich im Backup Verzeichnisses der Simulation ab, das bei der Version 1.5 das Verzeichnis „/var/vnuml/SimpleNet/“ wäre. Da diese Netzwerk-Simulation aber manuell und ohne VNUML aufgebaut wird, werden die Schnittstellen für die Management-Konsole von UML im Verzeichnis „/root/.uml/“ gespeichert.

```
host # uml_mconsole /root/.uml/uml1/mconsole
(/root/uml/) help
OK Commands:
   version - Get kernel version
   help - Print this message
   halt - Halt UML
   reboot - Reboot UML
[ . . . ]
```

Befehl 5.12 : Verwendung der UML-Management-Konsole „uml_mconsole“

In die Konsole kann eventuell auch einfach über den Befehl „uml_mconsole uml1“ eingeloggt werden.
Die UML-Management-Konsole kann zum Beispiel dazu verwendet werden die UML-Rechner zu stoppen und herunterzufahren oder auch neue Imagedateien im laufenden Betrieb einzubinden.
Die Optionen „con“ und „con0“ konfigurieren die „Login-Connection“ zu den UML-Rechnern. Die Option „con=null“ schaltet den direkten Login über die Konsole aus. Die Option „con0=xterm“ lässt ein X-Terminal

während des Startvorgangs aufpoppen, über das in den UML-Rechner eingeloggt werden kann. In den VNUML-Szenarios wird dies im <boot> Tag angegeben. Statt des X-Terminals „xterm“ kann auch ein anderes Terminal-Programm angegeben werden. Diese Option kann auch gänzlich weggelassen werden, wenn lediglich via SSH in den UML-Rechner eingeloggt werden soll. *(Der Backslash am Ende der Zeile gibt nur den Zeilensprung an)*

```
./linux ubd0=uml1_cow_fs,root_fs ubd1=uml1-opt-fs root=/dev/ubd/0 eth1=tuntap,uml1-eth1,fe:fd:0:0:1:2\
  eth0=tuntap,uml1-eth0,fe:fd:0:0:1:1 umid=uml1 con=null con0=xterm &

./linux ubd0=uml2_cow_fs,root_fs ubd1=uml2-opt-fs root=/dev/ubd/0 eth1=tuntap,uml2-eth1,fe:fd:0:0:2:2\
  eth0=tuntap,uml2-eth0,fe:fd:0:0:2:1 umid=uml2 con=null con0=xterm &
```

Befehl 5.11.8: Manuelles Starten der beiden UML-Rechner „uml1“ und „uml2“

Die obige Eingabe startet die beiden UML-Rechner auf dem Host-System. Während des Startvorgangs werden sie von der Script-Datei „umlboot“ konfiguriert, so dass das virtuelle Netzwerk nach dem vollständigen Hochfahren aller UML-Rechner direkt funktionsfähig aufgebaut ist.
Sämtliche Befehle, die hier für die manuelle Erstellung des virtuellen Netzwerks aus den UML-Rechnern „uml1“ und „uml2“ dargestellt sind, müssen auf dem Host ausgeführt werden. Es ist auch möglich all diese Befehle in eine Datei abzulegen und mit dem ausführen der Datei das virtuelle Netzwerk zu starten.

Die folgende Abbildung 5.8 zeigt die Verteilung der Prozesse auf dem Host, nachdem das virtuelle Netzwerk aufgebaut wurde. Der Netzwerkverkehr, der zwischen den beiden UML-Rechnern entsteht, wird durch die virtuellen Netzwerkschnittstellen und die Software-Bridge auf dem Host-System gereicht. Die Abbildung soll noch einmal verdeutlichen, dass aus Sicht des Host-Systems alles lediglich Prozesse sind. Die „sonstigen Anwendungsprozesse“ in der Abbildung stehen für beliebige Programme, wie zum Beispiel einen Web-Server oder den SSH-Dienst. Diese Prozesse laufen in den UML-Rechnern isoliert vom Host-System und können dieses nicht direkt beeinflussen.

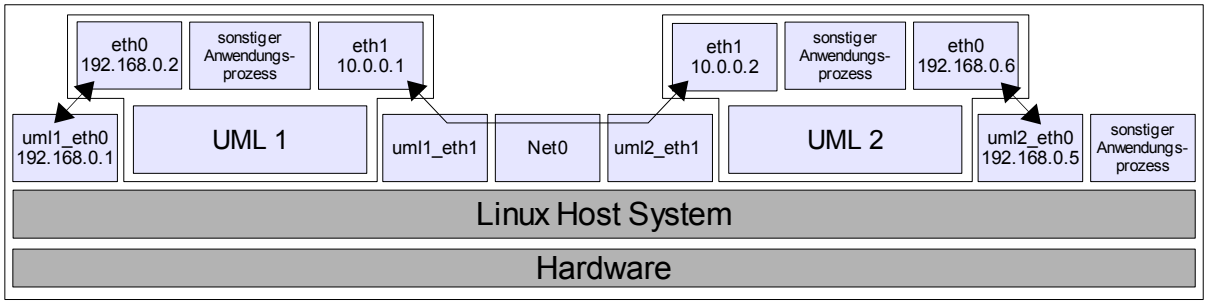


Abb. 5.8: Verteilung der Prozesse des virtuellen Netzwerks auf dem Host

Traditionell eröffnet jeder UML-Prozess einen Prozess im Host-System, weswegen dort über die Eingabe des Linux-Befehls „ps -aux“ viele UML-Prozessinstanzen während einer VNUML-Simulation zu sehen sein sollten. Bei UML gibt es hier einen speziellen „Thread“, genannt „tracing thread“, der die System-Aufrufe (system calls) der UML-Prozesse überwacht und zum UML-Kernel umleitet. Das hat den Nachteil, dass der UML-Kernel im Speicheradressbereich all seiner Prozesse zu finden ist. So kann innerhalb eines UML-Rechners über die Prozesse festgestellt werden, dass es sich um ein User Mode Linux System handelt. Auch könnte ein UML-Prozess auf das Host-System ausbrechen. UML benutzt außerdem Signal-Nachrichten um dem UML-Kernel während eines System-Aufrufs die Kontrolle zu ermöglichen. Die Nachrichten-Übertragung ist aber recht langsam und schwächt die Leistung des UML-Systems spürbar.
Um die Performance und Sicherheit, zum Beispiel für den Einsatz von UML für Honeypot-Systeme, zu verbessern wurde der SKAS-Patch (Separate Kernel Address Space) entwickelt. Der SKAS-Patch ist ein Kernel-Patch für den Host-Kernel. Er trennt den UML-Kernel von seinen Prozessen in dem er ihn in einem gänzlich anderen Speicheradressbereich laufen lässt. Damit können die UML-Prozesse nicht mehr direkt auf den UML-Kernel zugreifen. In deren Speicheradressbereichen findet sich nun kein UML-Kernel mehr und sie sind nicht mehr von gewöhnlichen Host-Prozessen zu unterscheiden. Durch die Separierung werden nun auch keine Nachrichten-Signale mehr für jeden System-Aufruf benötigt, was die Leistung des UML-Systems verbessert. Hier werden Leistungssteigerungswerte um die 30 % genannt, gerade in Verbindung mit virtuellen Netzwerken auf Basis von User Mode Linux.
Der Modus ohne SKAS wird tt-mode, für „Tracing Thread“, genannt. Der Modus mit SKAS wird skas-mode, für „Separate Kernel Address Space“, genannt. Der verwendete Modus wird beim Start des UML-Rechners angezeigt und ist zum Beispiel auch über den Befehl „cat /proc/cpuinfo“ im laufenden UML-Rechner zu sehen.
Mit dem SKAS-Patch laufen dann nur noch vier Prozessinstanzen für jeden UML-Rechner im Host-System. Dies ist der „UML-Kernel thread“, der den UML-Kernel ausführt und die System-Aufrufe der UML-Prozesse abfängt, der „UML userspace thread“, der die UML-Prozesse ausführt, der „ubd Treiber“ und der „write SIGIO emulation thread“. Der SKAS-Host-Kernel Patch ist auf der Internetseite des UML-Projektes zu finden.

5.6 Hinweise zur Installation von Software in den UML-Rechner

Da sich die SNMP-Software Net-SNMP, die in dieser Studienarbeit verwendet wird, nicht so ohne weiteres in das Root-Dateisystem der VNUML-Autoren installieren lässt, wird im folgenden etwas ausführlicher auf den allgemeinen Vorgang der Installation neuer Software eingegangen.
Es gibt generell zwei verschiedene Möglichkeiten neue Anwendungsprogramme in das Root-Dateisystem zu installieren. Zum einen in das Root-Dateisystem innerhalb der Basis-Imagedatei, so dass allen UML-Rechnern, die darauf aufbauen, das Anwendungsprogramm zur Verfügung steht. Zum Anderen in einen laufenden UML-Rechner, wobei das neu installierte Anwendungsprogramm dann nur diesem einen UML-Rechner zur Verfügung steht und bei Verlust der COW-Imagedatei, sofern verwendet, auch das Anwendungsprogramm gelöscht ist. Der Vorgang der Installation der Software selbst unterscheidet sich dabei nicht. Während im zweiten Verfahren nur der UML-Rechner gestartet und als Benutzer „root“ eingeloggt werden muss, müssen bei der Installation in die Basis-Imagedatei zuvor noch die folgende Schritte unternommen werden.

```
host filesystems # mkdir mntpoint
host filesystems # mount -o loop root_fs_tutorial-0.2.3 mntpoint/
host filesystems # dir mntpoint/
bin  cdrom etc   home  lib      mnt preferences root sys usr
boot dev   floppy initrd lost+found opt proc      sbin tmp var
host  filesystems # mount -t proc proc mntpoint/proc
host  filesystems # chroot mntpoint/ /bin/bash
/ # source /etc/profile
/ # dir /
bin  cdrom etc   home  lib      mnt preferences root sys usr
boot dev   floppy initrd lost+found opt proc      sbin tmp var
```

Befehl 5.13: Vorbereitung der Imagedatei für die Software Installation

Zuerst muss ein Verzeichnis erstellt werden, hier mit dem Namen „mntpoint“, über das die Basis-Imagedatei in das Host-System mit dem Befehl „mount“ eingebunden wird. Der Befehl „dir mntpoint/“ zeigt nun den Inhalt der Basis-Imagedatei an, hier das Root-Dateisystem der VNUML-Autoren. Man sieht das die Basis-Imagedatei einen gewöhnlichen Linux-Verzeichnisbaum enthält. Da so manche Software, zum Beispiel die Routing Software Quagga, während der Installation auf Einstellungen des Kernels zugreifen muss, die sich im Verzeichnis „proc“ befinden, muss auch dieses in das Root-Dateisystem „gemountet“ werden. Anschließend kann dann mit dem Befehl „chroot“ in das Root-Dateisystem hinein gewechselt werden. Für den Benutzer ist nun das UML-Root-Dateisystem und nicht mehr das Root-Dateisystem des Hosts gültig. Mit dem Befehl „source“ werden die Umgebungsvariablen geladen. Nun kann beliebige Software in das Root-Dateisystem innerhalb der Basis-Imagedatei installiert werden.

Um eine vollständige Internetverbindung innerhalb der „chroot“ Umgebung zu erhalten, von der aus auch Internetadressnamen aufgelöst werden können, muss vor dem Wechsel des Root-Dateisystems mit „chroot“ die Datei „resolv.conf“, die sich im Verzeichnis „/etc/“ auf dem Host befindet, ebenfalls in das Verzeichnis „/etc/“ des UML-Root-Dateisystem kopiert werden. Bei der Datei „resolv.conf“ handelt es sich um die Konfigurationsdatei des Resolvers. Sie enthält den zuständigen DNS-Server (Domain Name System) des Host-Systems und ermöglicht den Zugriff auf das Namensystem des Internet.
Mit dem Befehl „exit“ wird die „chroot“-Umgebung wieder verlassen und in das Host-System zurückgekehrt.

```
/ # exit
host filesystems # umount mntpoint/proc
host filesystems # umount mntpoint
```

Befehl 5.14: Verlassen der „chroot“-Umgebung und lösen der Imagedatei

Danach sollte unbedingt auch wieder die Einbindung des „proc“-Verzeichnisses und der Basis-Imagedatei aus dem Host-System über den Befehl „umount“ gelöst werden, sonst kann es während des Startens des UML-Rechners zu Problemen kommen.

Die zweite Möglichkeit, Anwendungsprogramme in einen laufenden UML-Rechner zu installieren, zum Beispiel während einer VNUML-Simulation, wird im folgenden dargestellt.

Um einen einzelnen UML-Rechner im laufenden Betrieb an das Internet anzubinden, können die folgenden Schritte durchgeführt werden. Dabei wird auf dem Host-Rechner ein NAT-Router (Network Address Translation) aktiviert, der den ein- und ausgehenden Datenverkehr des UML-Rechners durch das Host-System schleust. Wegen des NAT-Routers befinden sich dann hinter der einen Internet-Adresse des Host-Systems zwei Rechner, zum einen der Host-Rechner selbst und zum anderen der UML-Rechner.

```
host # modprobe tun
host # tunctl -t tap0 -f /dev/net/tun
host # ifconfig tap0 192.168.0.254 up
host # modprobe iptable_nat
host # iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
host # sysctl net.ipv4.conf.eth0.forwarding=1
host # sysctl net.ipv4.conf.tap0.forwarding=1
host # ./linux-2.6.10-1m ubd0=uml_cow_fs,root_fs_tutorial-0.2.3 eth0=tuntap,tap0,fe:fd:0:0:1:1 umid=lmu

( einloggen in den UML-Rechner mit Login=root und passwd=xxxx )
uml # ifconfig eth0 192.168.0.1 up
uml # route add default gw 192.168.0.254

host # scp /etc/resolv.conf root@192.168.0.1:/etc/
```

Befehl 5.15: Einrichten eines NAT-Routers auf dem Host für einen UML-Rechner

UML kann auch hier das „Virtual Point-to-Point Network Device“ (TUN) nutzen um in einem UML-Rechner eine virtuelle Netzwerkschnittstelle zu erstellen. Dafür muss im Linux Kernel des Host-Systems der „TUN/TAP Kernel Support“ als Modul aktiviert sein, was für die Funktion von VNUML allerdings ohnehin der Fall sein muss. Mit dem Befehl „modprobe“ wird das entsprechende Kernel Modul im Host aktiviert. Mit dem Befehl „lsmod“ können alle aktivierten Kernel Module des Hosts zur Überprüfung angezeigt werden. Über das Programm „tunctl“ wird die virtuelle Netzwerkschnittstelle generiert und über „ifconfig“ erhält sie die Host-seitige IP-Adresse. Nachdem auch das Kernel Modul „iptable_nat“ aktiviert wurde kann mit dem Firewall-Programm „iptables“ der NAT-Router über der Netzwerkschnittstelle „eth0“, die hier im Beispiel den Host-Rechner mit dem Internet verbindet, konfiguriert werden. Hierfür muss im Kernel des Host-Rechners auch der „MASQUERADE target support“ als Modul aktiviert sein. Damit der Host-Rechner die Netzwerk-Pakete des UML-Rechners auch ins Internet weiterleitet muss die IP-Weiterleitung mit dem Befehl „sysctl“ aktiviert werden.

Beim Start des UML-Rechners wird dann auf dem Host eine „TAP“-Netzwerkschnittstelle (Virtual Ethernet Network Device) mit der IP-Adresse 192.168.0.254 erstellt. Unter dieser IP-Adresse ist der Host vom UML-Rechner aus zu erreichen. Nachdem in den UML-Rechner eingeloggt wurde muss auch dessen Netzwerkschnittstelle, hier mit der IP-Adresse 192.168.0.1, aktiviert werden. Nun besteht auch schon die virtuelle Netzwerkverbindung und die IP-Adresse der „TAP“-Netzwerkschnittstelle des Hosts sollte im UML-Rechner als „Default-Gateway“ eingestellt werden.
Auf die gleiche Weise wurde im Kapitel 5.5 das virtuelle Netzwerk manuell aufgebaut.

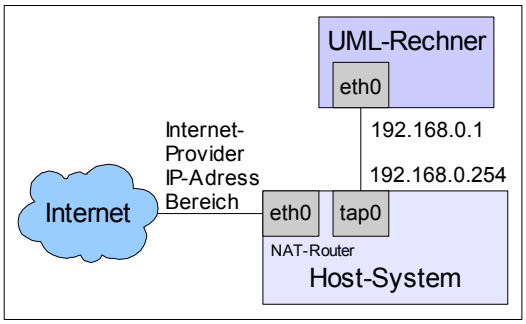


Abb. 5.9: Netz-Topologie der UML-Internet-Verbindung

Damit der UML-Rechner auch Adressnamen auflösen kann, benötigt er noch die Datei „resolv.conf“.
Der UML-Rechner kann mit dem Befehl „system -h now“, eingegeben innerhalb des UML-Rechners, wieder heruntergefahren werden oder über die Management-Konsole mit „uml_mconsole lmu halt“, eingegeben im Host, abgeschaltet werden. Mit dem Befehl „ifconfig tap0 down“ und „tunctl -d tap0“ wird auch die virtuelle Netzwerkschnittstelle im Host wieder deaktiviert.

Um einem beliebigen UML-Rechner, der zum Beispiel Bestandteil einer VNUML-Simulation ist, einen Zugang zum Internet zu ermöglichen genügt also einfach das Konfigurieren eines NAT-Routers auf dem Host, sofern der UML-Rechner, zum Beispiel über das Management-Netzwerk, mit dem Host verbunden ist und dieser wiederum mit dem Internet. Im folgenden wird der UML-Rechner „uml1“ des SimpleNet-Szenarios, mit dem Internet über einen Nat-Router auf dem Host verbunden.

```
host # modprobe iptable_nat
host # iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
host # sysctl net.ipv4.conf.eth0.forwarding=1
host # sysctl net.ipv4.conf.uml1-eth0.forwarding=1
host # ssh root@uml1 'route add default gw 192.168.0.1'
host # scp /etc/resolv.conf root@uml1:/etc/
```

Befehl 5.16: Einrichten einer Internetverbindung für den UML-Rechner „uml1“

Das Debian Linux-System, das VNUML in seinem Root-Dateisystemen verwendet, verfügt über eine Online-Installationsroutine, genannt „APT“. Um Anwendungsprogramme automatisch Mithilfe der APT-Tools zu installieren muss, wie zuvor beschrieben, eine Internetverbindung aus dem gestarteten UML-Rechner oder der „chroot“-Umgebung bestehen. Debian Linux stellt unter anderem die beiden Programme „apt-get“ und „apt-cache“ zur Verfügung. Diese Programme greifen auf eine interne Software-Distributionsliste zu, in der die verfügbaren Softwarepakete, die die verschiedenen Anwendungsprogramme enthalten, aufgelistet sind. In der Datei „/etc/apt/sources.list“ sind die Adressen der Internet-Server gelistet, von denen die Softwarepakete heruntergeladen werden können.

Die Debian APT-Tools verfügen, unter anderem, über die folgenden Optionen.

- **apt-get update**
  Synchronisiert die lokale Software-Distributionsliste der verfügbaren Softwarepakete mit einer globalen Distributionsliste im Internet. Der Befehl muss auch jedesmal ausgeführt werden, nachdem ein neuer Internet-Server in die Datei „sources.list“ eingetragen wurde, bevor von diesem Softwarepakete heruntergeladen und installiert werden können.
- **apt-cache search „**Suchwort“
  Findet alle Pakete auf der Software-Distributionsliste, deren Namen dem Suchwort ähneln oder mit dem Suchwort in einer Beziehung stehen.
- **apt-get install** „Softwarepaket“
  Lädt das angegebene Softwarepaket von einem Internet-Server herunter und installiert die enthaltene Software in das lokale System. Dabei wird auch überprüft, ob die Funktion der enthaltenen Software von weiteren Softwarepaketen abhängt, die dann ebenfalls heruntergeladen und installiert werden.
- **apt-get remove** „Softwarepaket“
  Deinstalliert die Software des angegebenen Paketes aus dem lokalen System.
- **apt-get source** „Softwarepaket“
  Lädt lediglich die Quelltexte des angegebenen Softwarepaketes aus dem Internet und entpackt diese im lokalen System ohne Installation. Hierfür wird eine Softwarequelltext-Quelle benötigt, die mit der Anweisung „deb-src“ in der Datei „sources.list“ angegeben werden muss. Meist genügt es eine dort bereits vorhandene „deb“ Anweisung einfach in „deb-src“ zu ändern, da dann unter der angegebenen Software-Quelle auch Quelltexte heruntergeladen werden können.
- **apt-get build-dep** „Softwarepaket“
  Lädt die zu dem angegebenen Softwarepaket abhängigen Pakete aus dem Internet und installiert deren Software. Das ist zum Beispiel notwendig, wenn ein Programm aus dem Quellcode lokal kompiliert werden soll, dafür aber eine bestimmte Softwareumgebung für seine Funktionen benötigt.

Die Debian Installationsroutine installiert standardmäßig fertig kompilierte Binärpakete und überprüft die Abhängigkeiten der zu installierenden Softwarepakete zu anderen Paketen und installiert diese gegebenenfalls mit. Der Vorteil dabei ist, dass die gewünschte Software ohne großen Aufwand meist recht schnell und funktionsbereit zur Verfügung steht.
Ein Nachteil ist aber, dass die Software eben bereits konfiguriert und kompiliert ist und womöglich nicht gewollte Einstellungen übernommen werden müssen oder auf gewünschte Einstellungen verzichtet werden muss. Da das UML-Root-Dateisystem auch weitestgehend klein gehalten wird, kann es gerade hier passieren, dass die Installation eines Softwarepaketes über die Debian APT-Tools, die Installation einer großen Menge an zusätzlicher Software nach sich zieht.
Bei Debian werden zu jedem Zeitpunkt drei Versionen eines Softwarepaketes parallel angeboten, genannt stable (stabil), testing (prüfen) und unstable (instabil), die einem unterschiedlich fortgeschrittenen Software-Entwicklungszweig zugeordnet sind. Je nachdem aus welchem Zweig ein Softwarepaket installiert wird, müssen eventuell davon abhängige, bereits installierte Softwarepakete ebenfalls neu installiert werden.
Es können aber nicht beliebig viele Anwendungsprogramme in das Root-Dateisystem installiert werden, da die Größe der Imagedatei den vorhandene Speicherplatz beschränkt. Da manche Anwendungsprogramme jedoch, wie zum Beispiel die SNMP Software Net-SNMP, viel Speicherplatz benötigen, wird im folgenden nun kurz erläutert, wie die Imagedatei vergrößert werden kann.

```
host filesystems # e2fsck -f root_fs_tutorial-0.2.3
host filesystems # dd if=/dev/zero of=root_fs_tutorial-0.2.3 bs=1M count=1 seek=1000 conv=notrunc
host filesystems # resize2fs -p root_fs_tutorial-0.2.3
host filesystems # ls -lsh root_fs_tutorial-0.2.3
552M -rwxr-xr-x  1 root root 1001M May 10 00:50 root_fs_tutorial-0.2.3
host filesystems # e2fsck -f root_fs_tutorial-0.2.3
```

Befehl 5.17: Vergrößern einer vorhanden Imagedatei

Da die VNUML-Autoren für ihr UML-Root-Dateisystem das „second extended filesystem“ (ext2) verwenden wird hier zuerst mit dem Programm „e2fsck“ das Dateisystem auf seine Konsistenz überprüft. Anschließend wird mit dem Befehl „dd“ die vorhandene Imagedatei „root_fs_tutorial-0.2.3“, hier auf 1000 Megabyte, vergrößert. Dabei wird Aufgrund der „seek“ Option des „dd“ Befehls der neu gewonnene Speicherplatz mit Nullwerten vollgeschrieben und nicht gleich belegt, sondern vorerst nur reserviert. Schließlich wird das Dateisystem der Imagedatei über den neuen Speicherplatz, mit dem Befehl „resize2fs“ ausgedehnt. Der Befehl „ls -lhs“ zeigt, dass die Imagedatei „root_fs_tutorial-0.2.3“ nun gut 1000 Megabyte groß ist aber nach wie vor, wegen der bereits vorhandenen Daten des Root-Dateisystems, nur etwa 550 Megabyte Speicherplatz belegt. Anschließend folgt noch einmal eine Überprüfung der Konsistenz des Dateisystems.
Da dieses Vorgehen auch fehlschlagen kann wird im folgenden noch gezeigt, wie eine neue Imagedatei erstellt und das Root-Dateisystem dort hinein kopiert werden kann. Hierbei wird ebenfalls mit dem Befehl „dd“ eine komplett neue Imagedatei erstellt, die anschließend mit dem Linux-Dateisystem „ext2“ formatiert wird. Die Imagedatei hat eine Größe von gut 1000 Megabyte, belegt aber Anfangs nur 2 Megabyte Speicherplatz. Zwei Verzeichnisse müssen erstellt werden, in die die neu erstellte Imagedatei und die vorhandene Imagedatei mit dem Root-Dateisystem für den Kopiervorgang eingebunden werden. Mit dem Kopierbefehl „cp -Ripv“ wird dann das Root-Dateisystem aus der kleineren Imagedatei in die neue, größere Imagedatei kopiert.

```
host filesystems # dd if=/dev/zero of=root_fs count=1 seek=1000 bs=1M
host filesystems # ls -lsh new_root_fs
2.0M -rw-r--r--  1 root root 1001M May 10 00:56 new_root_fs
host filesystems # mke2fs -F new_root_fs
host filesystems # mkdir mntoldimage
host filesystems # mkdir mntnewimage
host filesystems # mount -o loop root_fs_tutorial-0.2.3 mntoldimage
host filesystems # mount -o loop root_fs mntnewimage
host filesystems # cp -Rdipv mntoldimage/* mntnewimage/
[...]
```

Befehl 5.18: Erstellen einer neuen Imagedatei für UML-Rechner

Anschließend sollten die beiden Imagedateien wieder mit dem Befehl „unmount“ aus dem Host-System gelöst werden. Nachdem eine Basis-Imagedatei verändert wurde, können zugehörige COW-Imagedateien nicht mehr damit verwendet werden.

5.7 Die Routing-Software Quagga

Quagga ist eine Routing-Software die TCP/IP basierte Routing-Dienste bereitstellt und Routing-Protokolle, wie RIP, OSPF und BGP in verschiedenen Versionen, unterstützt. Die Quagga Routing-Dienste sind bereits im UML-Root-Dateisystem, das die VNUML-Autoren für ihre Beispiel Netzwerk-Szenarios zur Verfügung stellen, installiert und können für VNUML Netzwerk-Simulationen eingesetzt werden.
Quagga ist eine Weiterführung des von Kunihiro Ishiguro im Jahre 1996 gestarteten Zebra Projektes und steht unter der GNU General Public Licence. Die letzte Version von Zebra, mit der Versionsnummer 0.94, wurde Ende 2003 veröffentlicht. Unter der Bezeichnung Quagga streben die neuen Autoren demnächst den Sprung auf die Version 1.0 an. Quagga ist ein eigenständiges Open-Source Projekt und unabhängig von der VNUML-Entwicklung. Es gibt keine Unterschiede beim Einsatz, der Funktion und der Konfiguration von Quagga zwischen einem reellen Rechnernetz und einer VNUML Netzwerk-Simulation.
Quagga hat für jedes unterstützte Routing Protokoll ein eigenes Routing-Dienstprogramm. So behandelt das Programm „ospfd“ OSPF-Routing, „ripd“ RIP-Routing und „bgpd“ BGP-Routing. Der Kernel-Routing-Table-Manager „zebra“ schließlich beeinflusst die Routing Tabelle des Kernels und vermittelt Routen zwischen den einzelnen Quagga Routing-Diensten.

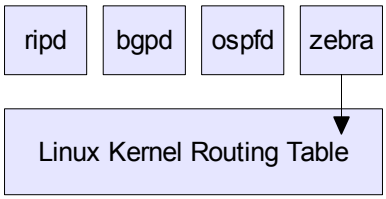


Abb. 5.10: Einige Quagga Routing Dienste

Diese Multi-Prozess Architektur bringt Quagga eine gewisse Erweiterbarkeit, Modularität und ermöglicht eine einfache Wartung. Alle Quagga Routing-Dienste können über eine eigene Konfigurationsdatei, die beim Start

angegeben werden muss, und während der Laufzeit, über eine „Telnet“-Schnittstelle, konfiguriert werden.
Werden mehrere Quagga Routing-Dienste in einem Rechner benötigt, kann die Vielzahl der Konfigurationsdateien und „Telnet“-Schnittstellen schnell unübersichtlich werden. Die Quagga Autoren haben dieses Problem mit einer integrierten Benutzerschnittstelle, genannt „vtysh“, gelöst. Der Benutzer loggt sich in das „vtysh“-Terminal ein, das als eine Art Proxy funktioniert und sich mit jedem lokal laufenden Routing Dienst verbindet. So können alle Quagga Routing Dienste während der Laufzeit über eine einzige Schnittstelle konfiguriert werden.
Die Konfiguration der einzelnen Quagga Routing-Dienste kann per SNMP ausgelesen werden. Quagga hat keinen eigenen SNMP Agenten implementiert, sondern verbindet sich über das SMUX-Protokoll mit einem Master SNMP-Agenten. Für alle Routing Protokolle liefert Quagga in seinem Quelltext-Verzeichnis die benötigten MIB-Dateien für die „Management Information Base“ mit.

5.7.1 Installation

Im Root-Dateisystem der VNUML-Autoren ist Quagga zwar bereits installiert, allerdings ohne SNMP, bzw. SMUX Unterstützung. Damit Quagga auch via SNMP angesteuert werden kann muss vor der Installation von Quagga die SNMP-Software bereits installiert worden sein. Für diese Studienarbeit wurde die SNMP-Software Net-SNMP gewählt, die derzeit wohl den umfangreichsten SNMP-Agenten für Linux bietet und auch von den Quagga-Autoren empfohlen und in deren Dokumentation über die SNMP/SMUX-Schnittstelle verwendet wird. Hinweise zur Installation, Konfiguration und Verwendung von Net-SNMP sind im Kapitel 6 zu finden.
Die Software Quagga steht auf der Internetseite „http://www.quagga.net“ zum download bereit. Zwar kann sie im Root-Dateisystem der VNUML-Autoren auch als Binärpaket über die Online-Installationsroutine des Debian Linux-Systems installiert werden, allerdings unterstützt diese fertig kompilierte Version kein SNMP. Soll SNMP unterstützt werden muss die Installation manuell durchgeführt werden, was im folgenden gezeigt wird.
Allgemeine Hinweise zur Installation neuer Software innerhalb des UML-Root-Dateisystems der VNUML-Autoren sind im Kapitel 5.6 zu finden.
Der Quagga Quellcode kann auch mit Hilfe der bereits erwähnten Debian Bordmittel aus dem Internet automatisch heruntergeladen werden. Dafür muss in der Datei „/etc/apt/sources.list“ über das Schlüsselwort „deb-src“ eine Softwarequelltext-Quelle hinzugefügt wurde. Der folgende Befehl lädt dann die Quagga-Software automatisch aus dem Internet und entpackt sie im selben Verzeichnis.

```
/ # apt-get source quagga
```

Befehl 5.19: Herunterladen und Entpacken des Quagga-Quellcodes

Damit Quagga SNMP unterstützt und mit einem SNMP Agenten Daten über SMUX austauscht muss SNMP für das Kompilieren aktiviert werden. Um die Vorzüge des „vtysh“-Terminals zu nutzen und die laufenden Quagga Routing Dienste nicht einzeln via „Telnet“ ansteuern zu müssen, muss auch „vtysh“ aktiviert werden.
Der Befehl „./configure --help“, eingegeben innerhalb des Quagga Quelltext-Verzeichnisses, listet alle Konfigurationsmöglichkeiten auf. Mit den folgenden Befehlen wird Quagga dann mit den benötigten Routing-Diensten, SNMP Unterstützung und „vtysh“ konfiguriert, kompiliert und im System installiert.

```
/ # ./configure --enable-snmp --enable-vtysh
/ # make && make install
```

Befehl 5.20: Manuelle Installation von Quagga

Für die Installation innerhalb der Basis-Imagedatei muss unbedingt auch das „proc“-Verzeichnis in das Root-Dateisystem eingebunden sein. Sollte die Installation fehlschlagen, weil Software, die Quagga für seine Funktion benötigt, nicht im System installiert ist, so kann mit dem Befehl „apt-get build-dep quagga“ diese installiert werden. Je nach Ausstattung des Debian Linux-Systems kann dieser Befehl allerdings die Installation einer sehr, sehr großen Menge an zusätzlicher Software verursachen.
Nach der erfolgreichen Installation sind die einzelnen Quagga Routing-Dienste global über die Konsole verfügbar. Mit dem Befehl „which“ kann festgestellt werden, in welchem Verzeichnis sie abgelegt wurden. Normalerweise sind die Quagge Routing-Dienstprogramme im Verzeichnis „/usr/local/sbin/“ zu finden.

```
/ # which zebra
   /usr/local/sbin/zebra
/ # which ripd
   /usr/local/sbin/ripd
```

Befehl 5.21: Lokalisierung der Quagga Routing-Dienstprogramme

Während der praktischen Ausarbeitung dieser Studienarbeit gab es einige Probleme dabei die Quagga Routing-Dienstprogramme mit SNMP-Unterstützung zu installieren. So kann mit dem Befehl „ldd“ nach der erfolgreichen Installation von Quagga überprüft werden, ob die SNMP-Bibliothek, die die Quagga Dienste für den Einsatz von SMUX benötigen, auch richtig in die einzelnen Routing-Dienstprogramme eingebunden wurden. Für SNMP ist dabei eine gültige Referenz der „libnetsnmp.so.x“ Bibliothek wichtig, wobei das „x“ für die Versionsnummer steht, die Abhängig von der installierten Net-SNMP Version ist. Im Umgang mit Net-SNMP in der Version 5.3 konnte diese „Library Dependence“ zwischen den einzelnen Quagga Diensten und der „libnetsnmp“ Bibliothek nicht hergestellt werden, weswegen die Version 5.2.2 von Net-SNMP für die praktische Ausarbeitung dieser Studienarbeit verwendet wurde.

```
/ # ldd /usr/local/sbin/zebra
  [...]
  libreadline.so.5 => /lib/libreadline.so.5 (0x400b3000)
  libncurses.so.5 => /lib/libncurses.so.5 (0x400da000)
  libnetsnmp.so.9 => /usr/lib/libnetsnmp.so.9 (0x40111000)      ←
  libcrypto.so.0.9.7 => /usr/lib/libcrypto.so.0.9.7 (0x40195000)
  [...]
```

Befehl 5.22: Überprüfen der Referenz auf die Net-SNMP-Bibliothek

Ist diese Referenz gesetzt, sollte die Verbindung via SMUX zwischen dem Quagga Routing-Dienstprogramm und Net-SNMP grundsätzlich möglich sein. Ist diese Referenz nicht gesetzt, so ist den einzelnen Routing Dienstprogrammen die Anweisung „smux peer“, welche die SMUX Verbindung aufsetzt, unbekannt.
Damit die Datentypen und die Definitionen der Managementinformationen, die die Quagga Routing-Dienste bereitstellen, auch bekannt sind und auch komfortable über die Bezeichner der zugehörigen Objekte zugegriffen werden kann, müssen die MIB-Dateien der einzelnen Routing-Dienstprogramme in das Standard-MIB-Verzeichnis des SNMP-Managers und eventuell auch des Agenten kopiert werden. Die MIB-Dateien befinden sich in den einzelnen Unterverzeichnissen der Routing-Dienste innerhalb des Quagga Quelltext-Verzeichnisses. Quagga liefert die folgenden MIB-Dateien, die hier mit dem Programm „find“ innerhalb des Quelltext-Verzeichnisses gefunden und gleich kopiert werden.

```
/ quagga # find \( -name '*MIB*' -or -name '*SMI*' \) -exec cp -vi '{}' /usr/local/share/snmp/mibs/ \;
  ./zebra/GNOME-SMI                  // beinhaltet enterprises Objekte „gnome“ und „gnomeProducts“
  ./zebra/GNOME-PRODUCT-ZEBRA-MIB    // beinhaltet enterprises Objekte für die SMUX-Kommunikation
  ./bgpd/BGP4-MIB.txt                // beinhaltet die BGP4 Objekte                { mib-2 15 }
  ./ripd/RIPv2-MIB.txt               // beinhaltet die RIPv2 Objekte               { mib-2 23 }
  ./ospfd/OSPF-MIB.txt               // beinhaltet die OSPF Objekte                { mib-2 14 }
  ./ospfd/OSPF-TRAP-MIB.txt          // beinhaltet die Objekte für Traps mit OSPF  { ospf 16 }
```

Befehl 5.23: Aufspüren der MIB-Dateien der Quagga Routing-Dienste

Diese MIB-Dateien müssen in das Standard-MIB-Verzeichnis der SNMP-Software kopiert werden. Bei Net-SNMP ist das für gewöhnlich „/usr/local/share/snmp/mibs/“ oder auch „/usr/share/snmp/mibs/“. Den Net-SNMP Befehlen muss noch der Zugriff auf die neuen MIB-Dateien erlaubt werden, was am einfachsten mit dem setzen der Umgebungsvariable MIBS="ALL" oder der stets angegebenen Befehlsoption „-m ALL“ erledigt werden kann. Ohne diese Einstellung kann nur über die nummerische OID auf die Managementinformation zugegriffen werden.

Der Quagga Kernel-Routing-Table-Manager „zebra“ kann nach der erfolgreichen Installation wie folgt gestartet werden, wobei die Konfigurationsdatei „zebra.conf“, die mit der Befehlsoption -f beim Start angegeben wird, im Verzeichnis „/etc/quagga/“ abgelegt ist. Mit der Befehlsoption -d wird „zebra“ als Dienst gestartet.

```
uml2 # zebra -f /etc/quagga/zebra.conf -d
```

Befehl 5.24: Start des Quagga Kernel-Routing-Table-Managers „zebra“

Jedes Quagga Routing-Dienstprogramm benötigt eine eigene Konfigurationsdatei, die beim Start des Dienstes angegeben werden muss. So wird der RIP-Routing Dienst „ripd“ wie folgt gestartet.

```
uml2 # ripd -f /etc/quagga/ripd.conf -d
```

Befehl 5.25: Start des Quagga RIP-Routers „ripd“

Mit den anderen Routing-Dienstprogrammen, die Quagga enthält, wird ebenso verfahren, wenn sie eingesetzt werden sollen. Alle Routing Dienste von Quagga können parallel ausgeführt werden und über „zebra“ auch Routen untereinander austauschen, so dass verschiedene Routing Protokolle gleichzeitig in einem System unterstützt werden.

5.7.2 Beispiele für Konfiguration und Umgang mit Quagga

Auf den Internetseiten der Quagga Autoren finden sich alle Konfigurationsmöglichkeiten für die einzelnen Quagga Routing-Dienste. Im folgenden werden Beispiele für die Konfiguration des Kernel-Routing-Table-Managers „zebra“ und des RIP-Routers „ripd“ aufgezeigt.
Über die Konfigurationsdatei von „zebra“ können Netzwerkschnittstellen und statische Routen konfiguriert werden. Im Zusammenhang mit VNUML, das Netzwerkschnittstellen und statische Routen bereits in der Szenario-Beschreibung festlegen lässt und diese dann auch selbst setzt, ist eine ausführliche Konfiguration von „zebra“ nicht unbedingt notwendig. Trotzdem muss für den Start eine Konfigurationsdatei vorhanden sein, in der zumindest ein Hostname gesetzt ist.
Die folgende Konfigurationsdatei für „zebra“ mit dem Namen „zebra.conf“, beinhaltet den Hostnamen, die Zugangspasswörter der „Telnet“-Schnittstelle zum Durchführen von Konfigurationen während der Laufzeit, die Angabe der Log-Datei und die Konfiguration der SMUX-Verbindung. Zeilen die mit einem „!“ (Ausrufezeichen) beginnen gelten als Kommentarzeilen. Der Eintrag „line vty“ aktiviert die Möglichkeit, den Routing Dienst auch über die „vtysh“-Terminal-Shell zu konfigurieren, was wesentlich komfortabler ist wenn mehrere Quagga Routing-Dienste auf einem Rechner konfiguriert werden müssen.
Die Konfigurationsanweisung „smux peer“ aktiviert die SMUX-Verbindung zu einem lokal laufenden Master SNMP-Agenten. Die angegebene OID dient dabei lediglich dem Aufbau der Kommunikationsverbindung. Sie enthält keine verwertbaren Managementinformationen. Der OID kann ein Passwort folgen, hier „quagga_zebra“, das die Funktion eines SNMPv1/v2c Community-Namens hat und auch in der Konfiguration des Master SNMP-Agenten angegeben werden muss. Während der praktischen Ausarbeitung dieser Studienarbeit führte die Angabe eines solchen Passwortes allerdings häufig zu Verbindungsproblemen, weswegen empfohlen wird diese Stelle leer zu lassen und kein Passwort zu verwenden.
Die Konfiguration des Net-SNMP Agenten als SMUX-Gegenstelle und weitere Hinweise zum Abfragen der Managementinformationen sind im Kapitel 6.5.1.4 zu finden.

```
hostname uml2zebra
password zebra
enable password enzebra
!
debug zebra events
line vty
log file /var/log/quagga/zebra.log
smux  peer  .1.3.6.1.4.1.3317.1.2.1 quagga_zebra
```

Listing 5.3: Inhalt der „zebra“-Konfigurationsdatei „zebra.conf“

Über die Log-Datei kann festgestellt werden, ob während des Startens Probleme aufgetreten sind. Außerdem erlauben alle Quagga Routing Dienste verschiedene Arbeitsprozesse zu „Debuggen“, so dass kontinuierlich Statusbeschreibungen des Routing-Dienstes in die Log-Datei geschrieben werden, mit deren Hilfe der Routing Prozess analysiert werden kann. Die Quagga Routing-Dienste verfügen für die Konfiguration während der Laufzeit über unterschiedliche Sicherheitsmodi, die jeweils über ein eigenes Passwort aktiviert werden müssen. Der unprivilegierte Modus (Normal Mode) erlaubt lediglich das Anzeigen von Einstellungen und Routing-Informationen. Erst im privilegierten Modus (Enable Mode), in den über die Anweisung „enable“ und unter Angabe des in der Konfigurationsdatei definierten Passwortes gewechselt werden kann, ist das Konfigurieren des Routing-Dienstes erlaubt.
Wird der Routing-Dienst „zebra“ mit der oben aufgeführten Konfigurationsdatei gestartet, so kann während der Laufzeit wie folgt via „Telnet“ in das Terminal des Dienstes eingeloggt werden.

```
uml2 # telnet localhost 2601
Trying 127.0.0.1...
Connected to localhost
Hello, this is Quagga (version 0.99.4).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification
Password: zebra
uml2zebra > show ip forwarding
IP forwarding is on
```

Befehl 5.26: Einloggen in die „zebra“ -Terminal-Shell via „Telnet“

Statt beim Einloggen mit „Telnet“ die Portnummer anzugeben, hinter der die Terminal-Schnittstelle zu finden ist, wie hier im Beispiel dargestellt, kann auch der Name des Routing Dienstes angegeben werden, sofern dieser in der Datei „/etc/services“ mit der Portnummer verknüpft ist, was normalerweise der Fall sein sollte.

Soll die Routing-Konfiguration verändert werden, so muss in den „Enable Mode“ gewechselt werden.

```
uml2zebra > enable
Password: enzebra
uml2zebra#
```

Befehl 5.27: Wechseln in den „Enable Mode“ von „zebra“

In diesem Modus kann nach Eingabe der Anweisung „configure terminal“ die Konfiguration verändert werden, die dann mit der Anweisung „write“ auch bleibend in die beim Start angegebene Konfigurationsdatei übernommen werden kann. Alle gültigen Anweisungen, die innerhalb eines Terminals gelten, können über die Eingabe des Fragezeichens („?“) angezeigt werden.

```
uml2zebra > show ?
debugging       Zebra configuration
history         Display the session command history
interface       Interface status and configuration
ip              IP information
[...]
```

Befehl 5.28: Auflistung aller gültigen Anweisungen innerhalb „zebra“

Mit dem Drücken der „Tab-Taste“ werden, wie auch in Linux für Befehle üblich, angefangene Anweisungen vollendet oder alle alternativen Anweisungsendungen angezeigt. Alle möglichen Anweisungen werden mittels der Anweisung „list“ aufgelistet. Ausführlichere Beschreibungen aller Anweisungen der einzelnen Routing-Dienste sind auf der Internetseite des Quagga Projektes dokumentiert.
Das Beispiel einer Konfigurationsdatei für den RIP-Router „ripd“ wird im folgenden gezeigt. Der RIP-Router ermöglicht dynamisches Routing mittels des „Routing Information Protocols“ das auf dem Distanzvektor-Algorithmus aufbaut. Der RIP-Router verfügt teilweise über eigene Konfigurationsanweisungen. Die Anweisung „router rip“ aktiviert den RIP-Router und die Anweisung „network“ aktiviert die lokalen Netzwerkschnittstellen für RIP-Routing, deren IP-Adressen in den angegebenen IP-Adressbereichen liegen. Die Anweisung „line vty“ erlaubt, wie zuvor bei „zebra“, die Konfiguration des RIP-Routers über die „vtysh“-Terminal-Shell. Die Konfigurationsanweisung „smux peer“ aktiviert die SMUX Verbindung zu einem laufenden Master SNMP-Agenten. Jeder Quagga Routing Dienst verfügt über eine eigene OID für die Kommunikation via SMUX mit dem Master SNMP-Agenten.

```
hostname uml2ripd
password ripd
enable password enripd
!
line vty
!
router rip
   network 10.0.0./24
   network 164.1.2.0/24
!
smux  peer  .1.3.6.1.4.1.3317.1.2.3 quagga_ripd
!
log file /var/log/quagga/ripd.log
```

Listing 5.4: Inhalt der „ripd“-Konfigurationsdatei ripd.conf

Nach dem Start des Routing-Dienstprogramms „ripd“ kann in die Terminal-Shell, wie bereits bei „zebra“ vorgestellt, via „Telnet“ eingeloggt werden. Hier muss allerdings die Portnummer 2602 angegeben werden. Statt der Portnummer kann auch „ripd“ angegeben werden.

```
uml2  # telnet localhost 2602
Trying 127.0.0.1...
Connected to localhost
Hello, this is Quagga (version 0.99.4).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification
Password:  ripd
uml2ripd >
```

Befehl 5.29: Einloggen in die „ripd“-Terminal-Shell

Innerhalb des Telnet-Terminals kann nur der RIP-Router eingesehen und konfiguriert werden. Um einen gleichzeitig laufenden „zebra“ Dienst zu konfigurieren muss wieder in dessen Telnet-Terminal eingeloggt werden. Abhilfe schafft hier das bereits erwähnte „vtysh“-Terminal, das sich automatisch mit allen laufenden Quagga Routing-Diensten verbindet. Das Terminal „vtysh“ besitzt auch eine eigene Konfigurationsdatei, in der zum Beispiel ein Benutzer und ein Passwort angegeben werden kann. Die Konfigurationsdatei muss allerdings nicht, wie bei den Quagga Routing-Diensten, vorhanden sein.
Über den Befehl „vtysh“ kann in die Terminal-Shell eingeloggt werden.

```
uml2 # vtysh

Hello, this is Quagga (version 0.99.4).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

uml2# configure terminal
uml2(config)#  router ?
  bgp      BGP information
  isis     ISO IS-IS
  ospf     Start OSPF configuration
  rip      RIP
  zebra    Make connection to zebra daemon
```

Befehl 5.30: Einloggen in die „vtysh“-Terminal-Shell

Einige Anweisungen der einzelnen Quagga Dienste scheint „vtysh“ allerdings nicht zu übernehmen. So konnte zum Beispiel die „smux peer“ Anweisung nicht innerhalb dieser Terminal-Shell verwendet werden, sondern nur in den Terminals der einzelnen Quagga Routing-Dienste bzw. deren Konfigurationsdateien.
Die „vtysh“-Terminal-Shell bietet noch einen Vorteil, der auch im Zusammenhang mit Net-SNMP benutzt werden kann, um Routing-Informationen aus den Quagga Routern auszulesen, ohne vorher eine SMUX-Verbindung aufbauen zu müssen. Über die Befehlsoption -c (vormals -e) erlaubt „vtysh“ das Anzeigen von Information des Routers in der Konsole außerhalb der Terminal-Shell.

```
uml2 # vtysh -c ″show ip rip″
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
    (n) - normal, (s) - static, (d) - default, (r) - redistribute, (i) - interface

    Network         Next Hop   Metric From    Tag Time
C(i) 10.0.0.0/30     0.0.0.0         1 self          0
C(i) 164.1.2.0/30    0.0.0.0         1 self          0
```

Befehl 5.31: Anzeigen von RIP Routing-Informationen über „vtysh“ in der Konsole

Während der praktischen Ausarbeitung dieser Studienarbeit wurde in diesem Zusammenhang das kleine GNU Perl-Programm „quagga-snmp“ der Firma Net-Track entdeckt, das eben diese Funktion von „vtysh“ nutzt, um Informationen aus dem Quagga BGP-Router „bgpd“ auszulesen und über SNMP verfügbar zu machen. Dabei nutzt dieses Perl-Programm die Möglichkeit von Net-SNMP, über die dortige Konfigurationsanweisung „passpersist“ SNMP-Anfragen an externe Programme weiterzuleiten. Das Perl-Programm „quagga-snmp“ führt den Befehl „vtysh -c "show ip bgp summary"“ in der Konsole aus, ordnet die erhaltenen Informationen einzelnen Management-Objekten zu, die innerhalb des Scripts definiert sind, und macht diese für SNMP-Anfragen eines Managers über Net-SNMP erreichbar. Hierfür kann auf eine SMUX-Verbindung zwischen dem Net-SNMP-Agenten und dem Quagga Routing-Dienst verzichtet werden. Weitere Hinweise zur Erweiterung des Net-SNMP-Agenten mittels der Konfigurationsanweisung „passpersist“ sind im Kapitel 6.5.1.4 zu finden. Das Perl-Programm „quagga-snmp“ ist im Internet auf der Seite „http://net-track.ch/opensource/“ zu finden.

Um die SMUX Verbindung für das Auslesen von Informationen via SNMP zu nutzen muss der SNMP-Dienst vor den Quagga Routing-Diensten gestartet werden und wie im Kapitel 6.5.1.4 beschrieben konfiguriert sein.
In der Log-Datei des Net-SNMP Agenten „snmpd“ sollten sich dann, nach erfolgreichem zustandekommen der SMUX Verbindung mit „zebra“ und „ripd“, die folgenden Einträge finden.

```
[smux_accept] accepted fd 13 from 127.0.0.1:3189
accepted smux peer: oid GNOME-PRODUCT-ZEBRA-MIB::zserv, descr Quagga-0.99.4
[smux_accept] accepted fd 14 from 127.0.0.1:3190
accepted smux peer: oid GNOME-PRODUCT-ZEBRA-MIB::ripd, descr Quagga-0.99.4
```

Listing 5.5: Inhalt der Datei „snmpd.log“ nach Zustandekommen einer SMUX-Verbindung

Schlägt die Verbindung fehl, so wird dies in der Log-Datei des Net-SNMP Agenten und des entsprechenden Quagga Routing-Dienstes angezeigt.

Ist die Verbindung zwischen dem Net-SNMP-Agenten und einem Quagga Routing-Dienst, zum Beispiel dem RIP-Router „ripd“, erfolgreich aufgebaut, so können die Managementinformationen des Routing-Dienstes fortan über einen SNMP Manager abgefragt werden.

```
uml1 # snmptable -v1 -c public -Cw 100 -m ALL  10.0.0.2  .1.3.6.1.2.1.23.3
SNMP table: RIPv2-MIB::rip2IfConfTable

 rip2IfConfAddress  rip2IfConfDomain  rip2IfConfAuthType  rip2IfConfAuthKey  rip2IfConfSend
    10.0.0.2          "00 00 "          0                    ""                 ripVersion2
    127.0.0.1         "00 00 "          0                    ""                 doNotSend
    164.1.2.1         "00 00 "          0                    ""                 ripVersion2
    192.168.0.6       "00 00 "          0                    ""                 doNotSend

SNMP table RIPv2-MIB::rip2IfConfTable, part 2

 rip2IfConfReceive  rip2IfConfDefaultMetric  rip2IfConfStatus  rip2IfConfSrcAddress
    rip2               1                       notInService      10.0.0.2
    doNotRecieve       1                       notInService      127.0.0.1
    rip2               1                       notInService      164.1.2.1
    doNotRecieve       1                       notInService      192.168.0.6
```

Befehl 5.32: SNMP-Abfrage von Managementinformationen des RIP-Routers

Die Objekte, die die RIPv2 Managementinformationen definieren, sind mit dem RFC 1724 „RIP Version 2 MIB Extensions“ veröffentlicht worden.
Ihre Funktion kann aber auch mittels des Net-SNMP Programms „snmptranslate“ abgefragt werden, das die SMI-Definition aus der zugehörigen MIB-Datei ausliest. Das Programm wird im Kapitel 6.4.2 vorgestellt.

An dieser Stelle sei auch angemerkt, dass während der praktischen Ausarbeitung dieser Studienarbeit der Quagga OSPF Routing-Dienst „ospfd“ mit aktivem SMUX sehr instabil lief und nach SNMP-Anfragen von Managementinformationen hin und wieder abstürzte.

Im Quellcode-Verzeichnis von Quagga, sowie in der offiziellen Dokumentation finden sich weitere Hinweise zu Konfigurationen im Zusammenhang mit SNMP. Dort ist auch ein kleines Programm-Script zu finden, das im Net-SNMP Trap-Server eingebunden werden kann und SNMP-Meldungen des Quagga BGP-Routers „bgpd“ verarbeitet.

5.8 Zusammenfassung

In diesem Kapitel wurde das Netzwerk-Simulationsprogramm Virtual Network User Mode Linux (VNUML) vorgestellt. Am beispielhaften Aufbau einer einfachen Netzwerk-Simulation wurde die Funktionsweise von VNUML und ein Ausschnitt der Möglichkeiten, die dieses Programm bietet, gezeigt. Da VNUML hauptsächlich auf den Funktionen von User Mode Linux (UML) aufbaut, wurde auch darauf näher eingegangen. Ein kleines virtuelles Netzwerk wurde ohne die Hilfe von VNUML, aber mit der ebenfalls von VNUML verwendeten Software, aufgebaut, um die Zusammenhänge zwischen VNUML und UML zu erläutern und um ein besseres Verständnis für die Möglichkeiten von VNUML Netzwerk-Simulationen aufzuzeigen. Auf die UML-Rechner wurde näher eingegangen, um allgemeine Hinweise für ihre Erweiterung mit neuen Programmen zu geben. Es wurde gezeigt, dass VNUML zwar nichts neu erfindet aber trotzdem ein sehr mächtiges Linux-Rechner-Netzwerk-Simulationsprogramm ist, da es auf bewährter Linux Software aufbaut und diese auf intelligente Weise miteinander verknüpft, um so den komplexen Vorgang des Aufbaus einer Netzwerk-Simulation mit Hilfe von UML-Rechnern hinter einer vergleichsweise einfachen XML-Sprache zu verbergen.
Außerdem wurde in diesem Kapitel die Routing Software Quagga vorgestellt, die unterschiedliche Routing-Protokolle unterstützt und in einer VNUML Netzwerk-Simulation dynamisches Routing ermöglicht. Die Möglichkeit die einzelnen Routing-Dienste von Quagga über das Erweiterungsprotokoll SMUX mit einem Net-SNMP Agenten kommunizieren zu lassen wurde ebenfalls dargestellt.

6 Net-SNMP in VNUML Simulationen

Um SNMP in einer VNUML-Netzwerk-Simulation verfügbar zu machen bedarf es der Installation einer SNMP-Software in das Root-Dateisystem, das in einem VNUML-Szenario für die UML-Rechner verwendet wird. In den gegenwärtigen Root-Dateisystemen, die die VNUML-Autoren auf ihrere Internetseite zum Herunterladen bereitstellen, ist keine SNMP-Software installiert.
In diesem Kapitel wird die SNMP Implementierung Net-SNMP vorgestellt, installiert und Möglichkeiten der Konfiguration behandelt. Einige der Manager- und Agentenfunktionen, die Net-SNMP bietet, werden vorgestellt und in Beispielen angewendet.

6.1 Einleitung

Es existieren zahlreiche SNMP Implementierungen in verschiedenen Programmiersprachen. Unter den kostenlos, frei Verfügbaren hebt sich Net-SNMP hervor, weswegen es für den praktischen Teil der Studienarbeit verwendet wurde. Net-SNMP beinhaltet derzeit wohl die umfangreichste Implementierung eines SNMP-Agenten für das Linux-Betriebssystem, auf dem ja auch die UML-Rechner der VNUML Simulationen basieren. Die Erweiterungsprotokolle AgentX und SMUX werden ebenfalls durch Net-SNMP unterstützt.
Net-SNMP ist in der Programmiersprache C implementiert, ermöglicht aber zur Funktionserweiterung eine einfache Anbindung von Perl- und Bash-Script-Programmen. Net-SNMP ist in IPv4- und in IPv6-Netzen einsetzbar und könnte somit auch in VNUML IPv6-Simulationen verwendet werden.

Bei Net-SNMP handelt es sich um ein Softwarepaket, das aus mehreren Anwendungsprogrammen besteht und eine vollständige SNMP-Umgebung, also Manager und Agenten, aufbauen kann. Unterstützt werden alle derzeitigen SNMP Versionen, also SNMPv1, -v2c und -v3.

Insgesamt beinhaltet das Net-SNMP Softwarepaket folgende Programme:

- Eine Anzahl von Kommandozeilen-Programmen mit denen SNMP-Befehle und -Meldungen gesendet werden können.
- Darauf aufbauende „Second-Level“-Kommandozeilen-Programme, die den Umgang mit SNMP für spezielle Aufgaben erleichtern.
- Einen Agenten- sowie einen Trap-Server mit denen SNMP-Befehle, bzw. -Meldungen, empfangen, verarbeitet und beantwortet werden können.
- Eine grafische Benutzeroberfläche für den Manager
- Eine Programm-Bibliothek sowie verschiedene „Helfertools“, die zur Erweiterung und Konfiguration des Net-SNMP Agenten eingesetzt werden können.

Net-SNMP wird im gleichnamigen Open-Source Projekt gepflegt. Die offizielle Internetseite, sowie die Software selbst, ist unter „http://www.net-snmp.org“ zu finden. Bei zahlreichen Linux-Distributionen kann Net-SNMP einfach über deren Installationsroutine installiert werden.

6.2 Entwicklung und geschichtlicher Hintergrund zu Net-SNMP

Die Anfänge des Net-SNMP Projektes gehen in das Jahr 1992 an die Carnegie Mellon University (CMU) in Pennsylvania, USA, zurück. Die Netzwerk Gruppe der CMU, geleitet von Steve Waldbusser, entwickelte eine Softwareimplementierung des damals noch relativ neuen Netzwerkmanagement-Protokolls mit Namen SNMP. Diese Implementierung enthielt eine Programm-Bibliothek, eine Auswahl von Management-Befehlen und einen SNMP Agenten, welcher die meisten Standard-Managementinformationen, die im RFC 1213 definiert sind, verarbeiten konnte. Der Quellcode dieses Softwarepaketes wurde unter der Bezeichnung CMU-SNMP veröffentlicht und fand rasch Verbreitung.
Wes Hardaker, System Administrator an der University of California Davis (UCD), installierte das CMU-SNMP Paket und erweiterte den enthaltenen Agenten um mehr Informationen über die von ihm verwalteten Systeme zu erhalten und um Fehlerzustände besser erkennen zu können. Dabei vereinfachte er die Möglichkeit neue Managementinformationen in den Agenten einzubinden. Das Ergebnis war ein einfach zu erweiternder

SNMP Agent. 1995 veröffentlichte er den Quellcode seiner Arbeit als UCD-SNMP und begründete damit gleichfalls das UCD-Projekt. Schon kurz darauf schlossen sich einige weitere System-Administratoren dem Projekt an und portierten UCD-SNMP auf die Systeme, die sie verwalteten.
So folgte schon bald die Unterstützung für weitere Plattformen wie Solaris und HP-UX. Eine Perl-Schnittstelle, sowie eine „Thread-Sichere API“ und zahlreiche andere Erweiterungen wurden von immer mehr freiwilligen Programmierern beigesteuert. UCD-SNMP entwickelte sich zu einem vollwertigen Open-Source Projekt.
Im Jahr 1998 beteiligte die Internet Engineering Task Force (IETF) Wes Hardaker bei der Entwicklung einer Referenz-Implementierung der neu entworfenen SNMPv3-Spezifikation. Diese Arbeit verlief zuerst getrennt von der UCD-SNMP Entwicklung. Die Ergebnisse wurden aber schließlich in das UCD-Projekt übernommen.
Im August 1999 wurde UCD-SNMP in der Version 4.0 veröffentlicht. Neben der SNMPv3 Unterstützung, war auch schon eine erste Implementierung der AgentX Spezifikation enthalten. Das als überholt geltende SNMPv2p wurde zugunsten von SNMPv2c aus dem Softwarepaket entfernt.
Das UCD-SNMP Projekt verband nun, bis auf seinen Namen, kaum noch etwas mit der University of California, an der es einst entstand. So wurde der Name des Projektes im November 2000 in Net-SNMP geändert und auf „sourceforge.net“ als Open-Source Projekt angemeldet.
Dies und die Einstellung des bis dahin noch parallel laufenden CMU-SNMP Projektes der Carniege Mellon University beschleunigten die Weiterentwicklung von Net-SNMP erheblich.
Der Quellcode des Net-SNMP Agenten wurde restrukturiert und mit einer wesentlich flexibleren MIB-Modul API ausgestattet, die das Einbinden neuer Managementinformationen noch mehr erleichterte. Der Quellcode wurde nun zunehmend modularer aufgebaut, was zum Beispiel den Austausch der Netzwerk-Transportschicht, extern zuschaltbare SNMPv3 Zugriffskontrollen und Sicherheitsmodelle, sowie einen flexibleren Umgang mit der Verarbeitung von SNMP-Meldungen ermöglichte. Das Ergebnis wurde im April 2002 als Net-SNMP v5.0 veröffentlicht.
Durch die immer größer werdende Zahl freiwilliger Programmierer verbesserte sich nun auch die Unterstützung für Microsoft Windows Plattformen.
Auch in Eingebetteten Systemen wie zum Beispiel in Mikrokontrollern, speziell solche die µCLinux verwenden, findet der Net-SNMP Agent mittlerweile seine Anwendung. Zahlreiche auf Linux und BSD basierende Distributionen haben Net-SNMP in ihre Software-Distributionsliste aufgenommen.
Die enorme Popularität verdankt Net-SNMP auch zahlreichen zusätzlichen Programmen, die auf den Anwendungen des Softwarepaketes aufbauen, aber von anderen Projektgruppen entwickelt und gepflegt werden. So besitzt der Net-SNMP Agent zum Beispiel mit der Software LM-Sensors für das Linux-Betriebssystem sogar die Möglichkeit, Rechner-Hardware zu überwachen und Alarm zu schlagen, falls CPU-Temperatur oder Lüfterdrehzahl kritische Werte erreichen. Auch für den Net-SNMP Manager gibt es zahlreiche, vor allem für das Leistungsmanagement ausgelegte, professionelle Programme, die in Open-Source Projekten gepflegt werden.
Laut Net-SNMP Projektgruppe liegt das Projekt in der Sourceforge Statistik der „Most Active Projects“, seit seiner dortigen Anmeldung, in den oberen 2%, was auch andeutet wie belebt dieses Software-Projekt ist.

6.3 Installation von Net-SNMP

Es gibt kaum einen Unterschied zwischen der Installation von Net-SNMP in ein gewöhnliches Linux-System auf einem reellen Rechner und der Installation in ein UML-Root-Dateisystem für den Einsatz in VNUML-Simulationen. Es wird hier jedoch empfohlen Net-SNMP nicht während einer VNUML-Simulation in einen einzigen UML-Rechner zu installieren, da SNMP dann ja nur diesem einen UML-Rechner der Simulation zur Verfügung steht, sondern wie im Kapitel 5.6 beschrieben, direkt in die Imagedatei in der sich das UML-Root-Dateisystem befindet mit dem die UML-Rechner für die VNUML-Simulation generiert werden. Da mit dem Net-SNMP Softwarepaket mehrere Programme, die die Funktionen des SNMP-Agenten und des SNMP-Managers abdecken, installiert werden, ist es so letztlich nur eine Frage der Konfiguration, welcher UML-Rechner die Aufgaben eines SNMP-Managers und welcher die eines Agenten übernimmt.
Zwar gibt es über die Debian-Installationsroutine die Möglichkeit den Net-SNMP Agenten getrennt vom Net-SNMP Manager zu installieren, bei der manuellen Installation der Software werden aber standardmäßig alle notwendigen Anwendungsprogramme des Net-SNMP Softwarepaketes installiert.

Wird Net-SNMP auch in das Host-System installiert könnte dieses als Manager konfiguriert werden, während die UML-Rechner der VNUML-Simulation als Agenten funktionieren. Diese Möglichkeit des Einsatzes von Net-SNMP könnte dann in Ergänzung zum VNUML-Management arbeiten, mit Verbesserungen vor allem im Leistungs- und Fehlermanagement. Auf dem Host-System können dann, sehr viel einfacher und komfortabler als in den UML-Rechnern, professionelle Management-Applikationen, die auf dem Net-SNMP Manager aufbauen, verwendet werden. Ein um SNMP erweitertes VNUML-Management kann so auch vom Mehrwert, den grafische Ausgaben und Benutzeroberflächen mit sich bringen, profitieren.

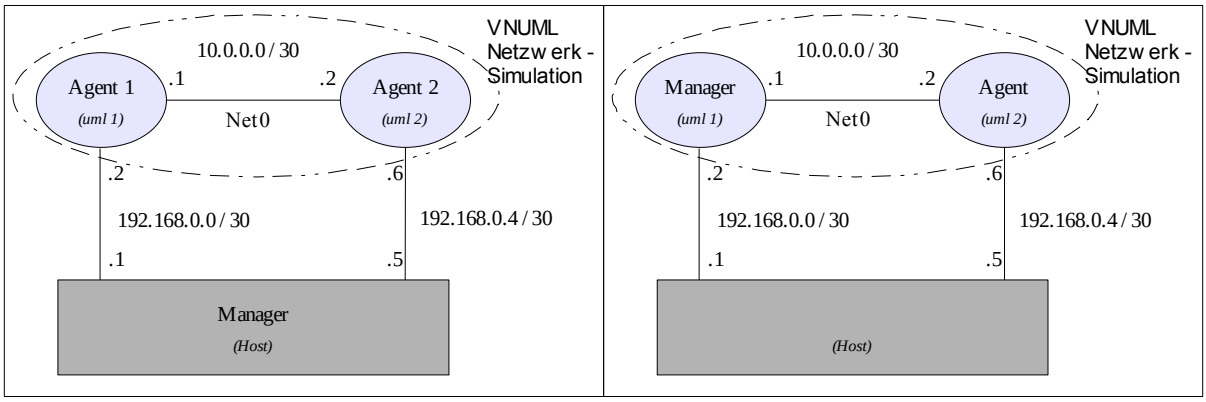


Abb. 6.1: Möglichkeiten des Einsatzes: SNMP als Erweiterung des VNUML-Managements oder in isolierter Simulations-Umgebung

Wie bereits erwähnt ist das Linux-System des UML-Root-Dateisystems, das von den VNUML-Autoren bereitgestellt wird, ein Debian Linux-System. Net-SNMP kann normalerweise direkt über die Online-Installationsroutine von Debian installiert werden, was hier aber gerade für das UML-Root-Dateisystem nicht empfohlen wird. Debian installiert neue Software in Form von Binärpaketen, also bereits fertig konfiguriert und kompiliert. Die Software Net-SNMP ist bei Debian auf verschiedene Softwarepakete verteilt. So enthält das Softwarepaket „snmp“ die Net-SNMP Manager-Applikationen und das Paket „snmpd“ den Net-SNMP Agenten. Dabei wird über das Abhängigkeitssystem geprüft, ob weitere Software für den Betrieb notwendig ist, die dann ebenfalls installiert wird. Derzeit verfügt zum Beispiel das Softwarepaket des Net-SNMP-Agenten über eine Abhängigkeit zum Hardwaremonitoring Softwarepaket „lm-sensors“ und verlangt neben diesem auch noch die Installation eines Linux-Kernels und weiterer Software.
Hier wird also die manuelle Installation des Net-SNMP Softwarepaketes empfohlen, von der für diese Studienarbeit die Version 5.2.2 gewählt wurde. Die derzeit schon verfügbare, neuere Version 5.3.1 funktionierte zwar ebenfalls einwandfrei, allerdings war es Quagga während der praktischen Ausarbeitung dieser Studienarbeit nicht möglich auf die SNMP-Bibliotheken dieser Version für den erfolgreichen Aufbau einer SMUX Verbindung zuzugreifen.

Allgemeine Hinweise zur Installation von Programmen in die Imagedatei des VNUML-Root-Dateisystems werden in Kapitel 5.6 aufgeführt.

Da Net-SNMP für die manuelle Installation gut und gerne 90 MB Speicherplatz benötigt und die Imagedatei mit dem UML-Root-Dateisystem eventuell nicht über so viel freien Speicherplatz verfügt, muss wie im Kapitel 5.6 beschrieben eine größere Imagedatei erstellt werden und das UML-Root-Dateisystem in diese umziehen. Abhängig von der Aktualität des UML-Root-Dateisystems und der gewählten Version von Net-SNMP muss eventuell auch andere Software von der Net-SNMP abhängt, wie zum Beispiel Perl, aktualisiert werden da ansonsten das Kompilieren und die Installation des Net-SNMP Softwarepaketes fehlschlägt.
Aufgrund der Schwierigkeiten, die sich vor allem während des Kompilierens von Net-SNMP innerhalb des von den VNUML-Autoren bereitgestellten UML-Root-Dateisystems ergaben, wurde während der praktischen Ausarbeitung dieser Studienarbeit eine eigenes UML-Root-Dateisystem auf Basis von Gentoo-Linux erstellt und mit diesem gearbeitet. Tatsächlich ist es wohl Abhängig von der Aktualität der Software und der Ausstattung des UML-Root-Dateisystems, ob das Kompilieren und die Installation des Net-SNMP Softwarepaketes insgesamt einfach oder schwierig vonstatten geht.

Im folgenden wird nun noch erläutert, wie die Installation von Net-SNMP weitestgehend mit Debian Bordmitteln vorbereitet werden kann. Dazu muss die Imagedatei, mit dem UML-Root-Dateisystem, sowie das darin befindliche „proc“-Verzeichnis, wie in Kapitel 5.6 erläutert, gemountet werden. Für die Internetverbindung muss die Datei „/etc/resolv.conf“ aus dem Host in das UML-Root-Dateisystem kopiert werden. Schließlich muss, für die Installation notwendig, mittels dem Befehl „chroot“ in das UML-Root-Dateisystem gewechselt werden. Um Net-SNMP nun über die Debian APT-Tools aus dem Internet herunterzuladen muss in der Datei „sources.list“, die sich im Verzeichnis „/etc/apt/“ befindet, über die Anweisung „deb-src“, eine Softwarequelltext-Quelle angegeben werden. Es genügt hier meist eine dort bereits vorhandene „deb“ Anweisung in eine „deb-src“ Anweisung zu verändern. Dabei sollte auf den verwendeten Software-Entwicklungszweig geachtet werden. Hier werden „stable“ Software-Quellcodepakete freigeschaltet.

```
deb-src ftp://ftp.debian.org/debian stable main contrib non-free
```

Listing 6.1: Eintrag einer Quelltext-Downloadquelle für die Debian APT-Tools in die Datei „sources.list“

Wegen der neu eingetragenen Downloadquelle muss die Software-Distributionsliste nun noch aktualisiert werden. Anschließend kann Net-SNMP automatisch aus dem Internet heruntergeladen und direkt im gegenwärtige Verzeichnis entpackt werden. Bei bestehender Internetverbindung wird dies wie folgt durchgeführt.

```
/ # apt-get update
/ # apt-get source net-snmp
```

Befehl 6.1: Download des Net-SNMP Quellcodes mittels Debians „apt-get“

Nun kann mit dem Kompilieren und der Installation des Net-SNMP Softwarepaketes fortgefahren werden. Lässt sich Net-SNMP nicht kompilieren, so kann versucht werden, die Software von der Net-SNMP abhängt zu aktualisieren und zu vervollständigen.
Mit dem folgenden Befehl aktualisiert und installiert Debian automatisch die Software, die Net-SNMP zu seiner Funktion benötigt.

```
/ # apt-get build-dep net-snmp
```

Befehl 6.2: Installation der Software, die Net-SNMP für seine Funktion benötigt

Nachteil dieses Befehls ist, wie bereits angemerkt, dass Debian Linux eventuell sehr viel zusätzliche Software installiert, die nicht unbedingt benötigt wird und der Speicherplatz bedarf des UML-Root-Dateisystems erheblich ansteigt. Abhängig davon, wie weit das Debian Linux des UML-Root-Dateisystems aktualisiert werden muss und welchen Umfang es besitzt, kann die Ausführung des Befehls auch fehlschlagen.
Der Net-SNMP Quelltext steht auch auf der Internetseite des Projektes zum Download bereit.

Nachdem entpacken des Quellcodes kann Net-SNMP beispielsweise wie folgt für das Kompilieren und die anschließende Installation vorbereitet werden. (*Die Backslashs am Ende der Zeilen zeigen lediglich den Zeilensprung an.*)

```
/ net-snmp # ./configure –-with-openssl=/usr/bin/openssl –-enable-ucd-snmp-compatibility \
  --enable-embedded-perl --with-perl-modules --enable-shared \
  --with-mib-modules=“smux disman/event-mib host ucd-snmp/diskio“ \
  --enable-mfd-rewrites
```

Befehl 6.3: Konfiguration für das Kompilieren von Net-SNMP

Alle möglichen Konfigurationsoptionen werden über den Befehl „./configure --help“ angezeigt.
Über die im Befehl 6.3 dargestellte Konfiguration werden alle Agenten- und Manager-Programme, die das Net-SNMP Softwarepaket beinhaltet, für das Kompilieren vorbereitet.
Mit der Konfigurationsoption in der ersten Zeile wird für die SNMPv3-Nachrichtenverschlüsselung „Openssl“ verwendet, das unter „/usr/bin/openssl“ im UML-Root-Dateisystem zu finden sein sollte. Die Einstellung danach aktiviert die Abwärtskompatibilität zum Net-SNMP Vorgänger UCD-SNMP.
Die drei Konfigurationsoptionen in der zweiten Zeile, bzgl. Perl und „enable-shared“, müssen angegeben werden, wenn es den Net-SNMP-Anwendungen möglich sein soll externe Programme zu starten. Sind diese Konfigurationsoptionen aktiviert, so kann zum Beispiel der Net-SNMP Trap-Server aufgrund einer eingehenden SNMP-Meldung ein externes Perl- oder Bash-Programm auf dem Rechner starten, das unabhängig von Net-SNMP den Inhalt der Meldung weiterverarbeitet.
Die Konfigurationsoption in der dritten Zeile aktiviert die an dieser Stelle aufgelisteten MIB-Module. MIB-Module sind Implementierungen von Objekten und deren Managementinformationen. Bei den hier angegebenen MIB-Modulen handelt es sich um solche, die nicht standardmäßig mit Net-SNMP kompiliert, installiert und in das Programm eingebunden werden. Einige Standard-MIB-Module, zum Beispiel die MIB-Module, die die in Kapitel 4.3.3.3 aufgelisteten Gruppen der MIB-II und deren Management-Objekte implementieren, werden automatisch von Net-SNMP unterstützt und müssen hier nicht angegeben werden. Auch das AgentX MIB-Modul wird standardmäßig unterstützt, während das ältere SMUX hier angegeben werden muss damit der Net-SNMP Agent auch SMUX verwenden kann. Der Bezeichner „disman/event-mib“ bindet das DISMAN-EVENT-MIB-Modul ein und aktiviert damit das Überwachungsmanagement, welches dann über die Konfigurationsdatei des Net-SNMP Agenten konfiguriert werden kann. Der Bezeichner „host“ steht für die Einbindung des MIB-Moduls welches die HOST-RESOURCES-MIB implementiert, die im RFC 2790 spezifiziert ist und eine Fülle von Management-Objekten über die Ressourcen eines Agenten definiert, unter Linux zum Beispiel die Anzahl der angeschlossenen Festplatten und deren Speicherkapazität sowie auch laufende Anwendungsprozesse und deren CPU-Auslastung. Der Bezeichner „ucd-snmp/diskio“ implementiert das „Disk Input/Output“-Statistik MIB-Modul der UCD-DISKIO-MIB. Über das Objekt „diskIOTable“ dieser MIB kann dann die gelesene und geschriebene Datenmenge der I/O-Schnittstellen des UML-Rechners eingesehen werden. Darüber hinaus sind im Softwarepaket von Net-SNMP noch weitere MIB-Module enthalten, die hier aber nicht verwendet werden. Auch eigene, selbstverfasste MIB-Module können an dieser Stelle angegeben werden, auf deren Objekte, bzw. Managementinformationen, der Net-SNMP Agent dann direkt zugreifen kann.

Die Konfigurationsoption in der vierten und letzten Zeile gibt Net-SNMP C-Quellcode, der mittels der MIB-Parser-Vorlage „Mfd“ (MIBs for Dummies) entstanden ist, stets den Vorzug gegenüber traditionellem Net-SNMP C-Code. Bei „Mfd“ handelt es sich um eine Art Schablone, die auf Basis einer vorhandenen SMI-Objekt-Definitionen eine übersichtliche C-Quellcode Vorlage, bestehend aus Funktionen der Net-SNMP C-API, erstellt und in die nur noch die eigentliche C-Code Schnittstelle zur Objekt-Ressource implementiert werden muss, so dass auf einfachem Weg, ohne größeres Wissen über die Net-SNMP C-API, neue Objekte und deren Managementinformationen in den Agenten eingebettet werden können. „Mfd“ erleichtert vor allem die Implementierung von Managementinformationen die in Tabellen strukturiert sind.
Während des Konfigurationsvorgangs werden sämtliche MIB-Module, die in den Net-SNMP Agenten direkt implementiert werden, angezeigt. Die Konfiguration schließt mit einem kurzen Frage/Antwort-Dialog ab, über den noch einige Einstellungen, wie zum Beispiel „Systemname“(oid: sysName), „Kontaktperson“ (oid: sysContact) und die zu verwendende Standard SNMP-Version angegeben werden können. Eine abschließende Übersicht zeigt die konfigurierten Einstellungen, die die Net-SNMP-Programme mit dem Kompilieren übernehmen werden.

Mit dem folgenden Befehl wird Net-SNMP schließlich kompiliert und installiert.

```
/ net-snmp # make && make install
```

Befehl 6.4: Installation von Net-SNMP

Nach der Installation sind sämtliche Net-SNMP Programme global in der Konsole ausführbar.
Zusätzliche Dateien, wie die MIB-Dateien in denen die SMI-Objekt-Definitionen der Management-informationen zu finden sind, befinden sich standardmäßig im Verzeichnis „/usr/local/share/snmp/“ oder auch im Verzeichnis „/usr/share/snmp/“.

An dieser Stelle sei noch angemerkt, dass während der praktischen Ausarbeitung der Studienarbeit festgestellt wurde, dass der Net-SNMP Agent die Managementinformationen seines zu verwaltenden Systems in einem Cache zwischenspeichert und diesen stellenweise nur minütlich aktualisiert. Dies hat im normalen Einsatz den Vorteil, dass der Agent schnell und direkt auf Managementinformationen zugreifen kann ohne das zu verwaltende System aufgrund der Anfrage eines oder mehrere Manager übermäßig zu belasten.
Für eine Simulations-Umgebung ist es jedoch nicht unbedingt wünschenswert, wenn zum Beispiel das Zähler-Objekt des ein- und ausgehende Datenverkehrs der Netzwerkschnittstellen (if[In/Out]Octets) nur einmal pro Minute aktualisiert wird und der Manager sich so auch nur minütlich über den Verkehrsfluss der Netzwerkschnittstelle informieren kann.
Der Versuch den Cache ganz abzuschalten ließ den Agenten überhaupt nicht mehr auf SNMP-Anfragen reagieren. Letztlich konnte zumindest für das Beispiel der Netzwerkschnittstellen-Aktualisierung Abhilfe geschafft werden in dem, vor dem Kompilieren des Quellcodes, die Variable „IFTABLE_CACHE_TIMEOUT“ aus der Datei „ifTable_data_access.h“, die sich im Verzeichnis „agent/mibgroup/if-mib/ifTable“ innerhalb des Net-SNMP Quelltext-Verzeichnisses befindet, auf einen kleineren Wert, zum Beispiel 1, gesetzt wurde. Somit aktualisiert der Agent die Managementinformationen der Objekte in der Interface-Tabelle, die die Verkehrsdaten der Netzwerkschnittstellen verwaltet, jede Sekunde.

6.4 Der Net-SNMP Manager

Net-SNMP besitzt mehrere Kommandozeilen-Programme die zusammen die Funktionen des SNMP-Managers ausüben und die Management-Applikationen darstellen. Sie können SNMP-Befehle senden, die Antworten der Agenten darauf empfangen und auf der Konsole ausgeben. Die Kommandozeilen-Programme entsprechen den SNMP-Befehlen, die in Kapitel 4.3.4 vorgestellt wurden. Darüberhinaus verfügt Net-SNMP über eine Anzahl von „Second-Level“-Kommandozeilen-Programmen die das Erledigen spezieller Aufgaben mittels SNMP erleichtern. Zusätzlich enthält Net-SNMP noch eine, auf die Kommandozeilen-Programme aufbauende, Benutzeroberfläche auf Tk/Perl-Basis, genannt „tkmib“. Zum Umfang des Net-SNMP Managers gehört auch der Trap-Server „snmptrapd“, der SNMP-Meldungen empfangen, verarbeiten und beantworten kann.

6.4.1 Konfiguration des Net-SNMP Managers

Die Konfiguration des Net-SNMP Managers ist auf zwei Wegen möglich. Der erste Weg ist eine allgemeingültige Konfiguration über eine Konfigurationsdatei, der zweite Weg eine spezifische Konfiguration, die bei jeder Ausführung eines Kommandozeilen-Programms über entsprechende Befehlsoptionen des Programms mit angegeben werden kann. Die grundlegende Konfiguration zur Funktion des Managers, wie zum

Beispiel der zu verwendende Netzwerk-Port, sind standardmäßig gesetzt und benötigen keine explizite Einstellung. Andere Konfigurationen des Managers, wie zum Beispiel die zu verwendende SNMP-Version und die Zugriffsnamen, Community- oder SNMPv3-Benutzernamen, müssen angegeben werden.
Der Großteil der Konfigurationen kann einfach über Befehlsoptionen bei jeder Ausführung eines Kommandozeilen-Programms angegeben werden. Zusätzlich, bzw. alternativ dazu, kann die Konfiguration des Managers über eine Datei mit Namen „snmp.conf“ beeinflusst werden. Diese Konfigurationsdatei kann für nur einen Benutzer oder systemweit für alle Benutzer geltend eingestellt werden. Um für nur einen Benutzer zu gelten muss sich die Datei im Home-SNMP-Verzeichnis „~/.snmp/“ (/root/.snmp/) des entsprechenden Benutzers befinden. Um systemweit für alle Benutzer zu gelten muss die Datei im Verzeichnis „/etc/snmp/“ abgelegt sein.
Die vielen verschiedenen Konfigurationsoptionen die es gibt werden hier nicht behandelt. Sie sind über die Manual-Page der Datei einsehbar die über den Befehl „man snmp.conf“ aufgerufen werden kann.

Mit dem Perl-Programm „snmpconf“ bietet Net-SNMP auch die Möglichkeit die „snmp.conf“ Datei über ein übersichtliches Menüsystem zu erstellen.

Die Konfigurationsmöglichkeiten des Net-SNMP Managers lassen sich in vier Kategorien einteilen:

- **Textuelles MIB-Parsen**
  Diese Kategorie bietet Konfigurationsmöglichkeiten die den Umgang mit der MIB des Managers definieren, zum Beispiel in welchem Verzeichnis die MIB-Dateien abgelegt sind, welche Management-Objekte der MIB eingebunden sein sollen, usw.
- **Standard-Authentifizierungs Optionen**
  Hierunter finden sich Konfigurationen die den Zugriff auf die SNMP-Agenten des Managers festlegen. Neben des zu verwendenden Netzwerk-Ports, auch die SNMP Version, Zugriffsnamen und Passwörter.
- **Debbug Ausgabe**
  Hier wird konfiguriert wie ausführlich die Ausgabe sein soll, um zum Beispiel Konfigurationsfehler besser finden zu können.
- **Ausgabeformat Optionen**
  Bietet Konfigurationsmöglichkeiten bezüglich des Ausgabeformats der eintreffenden SNMP-Nachrichten. Zum Beispiel kann hier eingestellt werden, dass eintreffende Managementinformationen so ausgegeben werden, dass sie einfach in Variablen anderer Programme gespeichert und direkt weiterverarbeitet werden können.

Die Konfiguration, die in der Datei „snmp.conf“ eingestellt wird, gilt als allgemeine Standardeinstellung des Managers, die bei jeder Ausführung von Kommandozeilen-Programmen automatisch mitgegeben wird. Die Datei „snmp.conf“ erleichtert in erster Linie den Umgang mit den Programmen des Net-SNMP Managers, da zum Beispiel die Passwörter eines SNMPv3-Benutzers nicht jedesmal bei einer SNMP-Anfrage explizit mit angegeben werden müssen. Das Vorhandensein dieser Datei ist jedoch nicht Voraussetzung für die Funktion der Net-SNMP Manager Programme.

6.4.2 Anwendungsprogramme des Net-SNMP Managers

Der Manager von Net-SNMP verfügt über mehrere Kommandozeilen-Programme, welche die verschiedenen SNMP-Befehle generieren, an den Agenten senden, dessen Antwort empfangen und diese auf der Konsole ausgeben.
Net-SNMP baut auf einem einheitlichen Kommandosatz auf, so dass sich alle Kommandozeilen-Programme in ihrer Verwendung ähneln. Die Kommandozeilen-Programme sind im allgemeinen wie folgt aufgebaut.

```
snmpcmd [Optionen] Agent [Parameter]
```

Befehl 6.5: Allgemeines Verwendungs-Schema der Kommandozeilen-Programme des Managers

An der Stelle des Platzhalters „snmpcmd“ wird der Name des eigentlichen Kommandozeilen-Programms angegeben. Zu den darauffolgenden „Optionen“ gehört zum Beispiel die verwendete SNMP Version und, dem Sicherheitsmodell entsprechend, Community- oder Benutzername sowie Authentifizierungs- und Verschlüsselungspasswort. Darüber hinaus können hier die zahlreichen Einstellungen angegeben werden die auch über die Konfigurationsdatei „snmp.conf“ gesetzt werden können. Den „Optionen“ folgt schließlich die IP-Adresse des Agenten, der mit dem SNMP-Befehl angesprochen werden soll. Die darauffolgenden „Parameter“ sind die Identifikationsnummern (OID) der Objekte des Agenten, die über den SNMP-Befehl angesteuert werden sollen, bzw. bei Set-Befehlen auch die zu übertragenden Managementinformationen.

Für die anzusteuernden Objekte muss nicht unbedingt deren nummerische OID im Parameterfeld angegeben werden. Die Objekte können auch über ihre Namen angesteuert werden, sofern der Net-SNMP Manager Zugriff auf die SMI-Definition des Objektes hat, aus der die Manager Programme die nummerische OID für das versenden letztlich ablesen. So kann beispielsweise die Systembeschreibung eines Agenten, die über das Management-Objekt „sysDescr“ aus der Gruppe „system“ verwaltet wird, wie folgt angesteuert werden.

```
.1.3.6.1.2.1.1.1.0
.iso.org.dod.internet.mgmt.mib-2.system.sysDescr.0
SNMPv2-MIB::sysDescr.0
system.sysDescr.0
sysDescr.0
```

Listing 6.2: Möglichkeiten der Objekt-Bezeichnung des Managers

Es können auch Mischformen zwischen nummerischen und textuellen OIDs verwendet werden. Damit die textuellen Objektnamen verwendet werden können, muss der Net-SNMP Manager Zugriff auf die textuelle „Management Information Base“ haben. Dafür muss den Net-SNMP Programmen das Verzeichnis bekannt sein, in dem die MIB-Dateien abgelegt sind. Standardmäßig benutzen die Net-SNMP Kommandozeilen-Programme das Verzeichnis „/usr/local/share/snmp/mibs/“. Neue MIB-Dateien, deren MIB-Module nicht mit Net-SNMP kompiliert und installiert wurden, müssen in diesem Verzeichnis abgelegt werden und sie müssen meist auch über die Befehlsoption (-m) oder über die Umgebungsvariable MIBS eingebunden sein. Mit dem Befehl „export MIBS="ALL"“ wird die Umgebungsvariable MIBS so gesetzt, dass alle MIB-Dateien, die sich im Standard-MIB-Verzeichnis befinden, von Net-SNMP verwendet werden. Erlaubt aber ein Agent grundsätzlich den Zugriff auf eine Managementinformation, so sollte diese vom Manager aus immer mit der nummerischen OID angesteuert werden können und über die textuelle OID immer nur dann, wenn der Manager zusätzlich auf die MIB-Datei, in der sich die SMI-Definition des Objektes befindet, zugreifen kann. Ob dies der Fall ist kann mit dem Kommandozeilen-Programm „snmptranslate“ festgestellt werden.

Im folgenden werden nun die wichtigsten Net-SNMP Kommandozeilen-Programme des Managers mit je einem Beispiel für ihre Benutzung erläutert. In den Beispielen wird immer die niedrigste SNMP Version verwendet, die für das Kommandozeilen-Programm funktioniert. Dies schließt die Funktion für die darüber liegenden Versionen mit ein. Das Beispiel basiert auf der Netztopologie der isolierten Simulations-Umgebung aus Abbildung 6.1. Der Manager besitzt die IP-Adresse 10.0.0.1 und der Agent die IP-Adresse 10.0.0.2 .
Bei den Kommandozeilen-Programmen wird die verwendete SNMP Version (1, 2c, 3) mit der Option -v angegeben. Für die ersten beiden Versionen muss ein Community-Name über die Option -c angegeben werden, hier ist dies „public“ für eine Community mit Lese-Rechten und „private“ für eine Community die zusätzlich über Schreib-Rechte verfügt.
Für die dritte SNMP-Version muss statt des Community-Namens ein Benutzername über die Befehlsoption -u, der Sicherheitslevel über die Befehlsoption -l (noAuthNoPriv, authNoPriv, authPriv) und, dem Sicherheitslevel entsprechend, die Passwörter (-A, -X) und die zugrundeliegenden Authentifizierungs- (-a) und Verschlüsselungsalgorithmen (-x) mit angegeben werden. Die verwendeten Zugriffsnamen müssen natürlich auf dem Agenten eingerichtet sein.
Standardmäßig verwenden die Net-SNMP Kommandozeilen-Programme des Managers den Netzwerkport 161 für das versenden der SNMP-Befehle.
Für jedes der hier aufgeführten Net-SNMP Kommandozeilen-Programme existiert auf dem Manager eine Manual-Page (man 'Programmname'), die detaillierte Angaben über die Verwendung und die vorhandenen Befehlsoptionen macht.

Der Net-SNMP Manager besitzt die folgenden Programme für die Ausführung von SNMP-Befehlen:

- **snmpget:**
  Entspricht dem Get-Befehl der SNMP-Spezifikation. Die Managementinformationen der angegebenen Objekte werden vom Agenten abgefragt. Im Beispiel wird die Systembeschreibung sowie die Kontaktinformation des Systemoperators vom Agenten abgefragt.

```
Manager # snmpget -v 1 -c public 10.0.0.2 SNMPv2-MIB::sysDescr.0 SNMPv2-MIB::sysContact.0

SNMPv2-MIB::sysDescr.0 = STRING: Linux Agent 2.6.12.6-1m #5 Mon Jul 3 23:32:43 UTC 2006 i686
SNMPv2-MIB::sysContact.0 = STRING: root@Agent
```

Befehl 6.6: Get-Anfrage mit Net-SNMPs „snmpget“

Das gleiche Beispiel nochmal ausgeführt, diesmal mit SNMPv3:

```
Manager # snmpget -v 3 -u myuser -l authPriv -a MD5 -A auth_password      \
                  -x DES  -X  priv_password  10.0.0.2  SNMPv2-MIB::sysDescr.0  SNMPv2-MIB::sysContact.0

SNMPv2-MIB::sysDescr.0 = STRING: Linux Agent 2.6.12.6-1m #5 Mon Jul 3 23:32:43 UTC 2006 i686
SNMPv2-MIB::sysContact.0 = STRING: root@Agent
```

Befehl 6.7: SNMPv3 Get-Anfrage mit Net-SNMPs „snmpget“

- **snmpgetnext:**
  Entspricht dem GetNext-Befehl der SNMP-Spezifikation. Die Managementinformation des Objektes, dessen OID die nächst Größere gegenüber der des angegebenen Objektes ist, wird ausgelesen. Im Beispiel wird die Systembeschreibung, deren OID wegen fehlender Angabe der Instanz folgt, und der Systemname, dessen OID auf die der Kontaktinformation folgt, ausgelesen.

```
Manager # snmpgetnext -v 1 -c public 10.0.0.2  SNMPv2-MIB::sysDescr  SNMPv2-MIB::sysContact.0

SNMPv2-MIB::sysDescr.0 = STRING: Linux Agent 2.6.12.6-1m #5 Mon Jul 3 23:32:43 UTC 2006 i686
SNMPv2-MIB::sysName.0 = STRING: Agent
```

Befehl 6.8: GetNext-Anfrage mit Net-SNMPs „snmpgetnext“

- **snmpbulkget:**
  Entspricht dem GetBulk-Befehl der SNMPv2-Spezifikation. Über die zusätzlich angegebenen Parameter „non-repeaters“ (-Cn) und „max-repetitions“ (-Cr) werden die auszulesenden Objekte eingegrenzt. Der erste Wert gibt die Anzahl der ersten OIDs auf der abzufragenden OID-Liste an, auf die nur eine einzige GetNext-Operation angewendet werden soll, der zweite Wert gibt die Anzahl der GetNext-Operationen an, die auf die übrigen OIDs auf der abzufragenden OID-Liste angewendet werden sollen. Im Beispiel wird, wegen fehlender Angaben der Instanz, der Systemname ausgelesen und die jeweils nächsten drei Werte unterhalb der OID der Netzwerkschnittstellen-Beschreibung „ifDescr“ und unterhalb der OID des Objektes „ifInOctets“, das die Anzahl der eingehenden Bytes der Netzwerkschnittstelle definiert.

```
Manager # snmpbulkget -v 2c -c public  -Cn1 -Cr3 10.0.0.2  SNMPv2-MIB::sysName  IF-MIB::ifDescr      \
                                                                                  IF-MIB::ifInOctets

SNMPv2-MIB::sysName.0 = STRING: Agent
IF-MIB::ifDescr.1 = STRING: eth1
IF-MIB::ifInOctets.1 = Counter32: 13443
IF-MIB::ifDescr.2 = STRING: eth0
IF-MIB::ifInOctets.2 = Counter32: 17709
IF-MIB::ifDescr.3 = STRING: lo
IF-MIB::ifInOctets.3 = Counter32: 0
```

Befehl 6.9: GetBulk Anfrage mit Net-SNMPs „snmpbulkget“

- **snmpset:**
  Entspricht dem Set-Befehl der SNMP-Spezifikation. Die Managementinformation eines Objektes des Agenten wird vom Manager aus neu gesetzt. Im Beispiel wird die Kontaktmöglichkeit des Systemoperators neu gesetzt. Bei der Übergabe der neuen Managementinformation muss auch der Datentyp des Objektes, dessen Wert neu gesetzt werden soll, angegeben werden. Der Buchstabe „s“ zwischen Objektangabe und neuer Managementinformation steht für den Datentyp „String“. Die Angabe für andere Datentypen kann über „snmpset --help“ eingesehen werden.
  An dieser Stelle sei auch angemerkt, dass Net-SNMP standardmäßig auf nur sehr wenige, hauptsächlich der aus dem eigenen Projekt entstandenen, Objekte den Schreibzugriff erlaubt, auch wenn dieser laut SMI-Definition des Objektes erlaubt sein müsste.

```
Manager # snmpset -v1 -c private 10.0.0.2 SNMPv2-MIB::sysContact.0 s "admin@agent"

SNMPv2-MIB::sysContact.0 = STRING: admin@agent
```

Befehl 6.10: Set-Aufforderung mit Net-SNMPs „snmpset“

- **snmpwalk, snmpbulkwalk:**
  Diese beiden Kommandozeilen-Programme sind so nicht in der SNMP-Spezifikation zu finden. Es handelt sich dabei um Net-SNMP eigene Befehlssätze, die aus mehreren SNMP Get- und GetNext- bzw. GetBulk-Befehlen zusammengesetzt sind. Das Programm „snmpwalk“ sendet mehrere GetNext- und Get-Befehle an den Agenten und liest die Managementinformationen sämtlicher Objekte aus, deren OID sich im Teilbaum unterhalb der angegebenen OID, befindet. Dieses Programm ist unter anderem auch dafür geeignet, die wirklich implementierten und verfügbaren Managementinformationen eines Agenten auf dem Manager aufzulisten. Wird keine OID angegeben so liest „snmpwalk“ die verfügbare Managementinformation aller Objekte aus, die Bestandteil der MIB-II Gruppen sind. Das Programm „snmpbulkwalk“ ist „snmpwalk“ ähnlich, baut allerdings auf SNMP GetBulk-Befehlen auf und ist deswegen wesentlich effizienter. Es hat dieselbe Ausgabe, erzeugt jedoch wesentlich weniger Datenverkehr zwischen Manager und Agent. Hier wird dann allerdings ein SNMPv2-fähiger Agent benötigt.

```
Manager # snmpwalk -v1 -c public 10.0.0.2 udp

UDP-MIB::udpInDatagrams.0 = Counter32: 146
UDP-MIB::udpNoPorts.0 = Counter32: 12
UDP-MIB::udpInErrors.0 = Counter32: 0
UDP-MIB::udpOutDatagrams.0 = Counter32: 133
UDP-MIB::udpLocalAddress.0.0.0.0.161 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalPort.0.0.0.0.161 = INTEGER: 161
```

Befehl 6.11: Auflistung aller verfügbaren Managementinformationen der UDP-MIB mit „snmpwalk“

- **snmptranslate:**
  Dieses Kommandozeilen-Programm entspricht keiner SNMP Spezifikation. Mit diesem Programm kann der komplett verfügbare ISO-Registrierungsbaum und damit die „Mangement Information Base“ des Managers nach einem Objekt durchsucht und die Definition des Objektes ausgelesen werden. Auch die Baumstruktur, die unter dem angegebenen Objekt aufgebaut ist, kann mit diesem Programm angezeigt werden. Im Beispiel links wird die Baumstruktur der UDP-MIB angezeigt und im Beispiel rechts die SMI-Definition des Objektes „udpLocalAddress“ aus der UDP-MIB.

```
Manager # snmptranslate -Tp -IR udp
+--udp(7)
  |
  +-- -R-- Counter   udpInDatagrams(1)
  +-- -R-- Counter   udpNoPorts(2)
  +-- -R-- Counter   udpInErrors(3)
  +-- -R-- Counter   udpOutDatagrams(4)
  |
  +--udpTable(5)
    |
    +--udpEntry(1)
      |  Index: udpLocalAddress, udpLocalPort
      |
      +-- -R-- IpAddr    udpLocalAddress(1)
      +-- -R-- INTEGER   udpLocalPort(2)
            Range: 0..65535
```

```
Manager # snmptranslate -Td -IR UDP-MIB::udpLocalAddress
UDP-MIB::udpLocalAddress
udpLocalAddress OBJECT-TYPE
  -- FROM       UDP-MIB, RFC1213-MIB
  SYNTAX       IpAddress
  MAX-ACCESS    read-only
  STATUS       current
  DESCRIPTION  "
    The local IP address for this UDP listener.  In the case of a UDP
    listener which is willing to accept datagrams for any IP interface
    associated with the node, the value 0.0.0.0 is used.
    "
::= { iso(1) org(3) dod(6) internet(1) mgmt(2) mib-2(1) udp(7)
udpTable(5) udpEntry(1) 1 }
```

Befehl 6.12: Abfrage von Objekt-Definitionen mittels „snmptranslate“

Damit „snmptranslate“ auch alle MIB-Dateien, die sich im Standard-MIB-Verzeichnis befinden, nach Objekten durchsucht, darf nicht vergessen werden, die Umgebungsvariable „MIBS“ oder die Befehlsoption -m auf „ALL“ zu setzen. Die Eingabe „snmptranslate -Tp -IR iso > isobaum.txt“ schreibt den komplett verfügbaren ASN.1-Registrierungsbaum der ISO, mit all seinen im Manager vorhandenen und zumindest theoretisch verwaltbaren Objekten, in die Datei „isobaum.txt“. Die Ausgabe wird hier deswegen in eine Datei geschrieben, da der lokal verfügbare ISO-Registrierungsbaum für gewöhnlich so groß ist, dass er nicht in den Hintergrundspeicher der Konsole passt und so nur teilweise einzusehen wäre. Über so genannte MIB-Browser, den auch das Net-SNMP Programm „tkmib“ enthält, können die Objekte aus der lokalen MIB, aber auch der komplette Registrierungsbaum, über eine meist übersichtliche Ordner-Struktur angesteuert werden.

Dies sind die wichtigsten Net-SNMP Kommandozeilen-Programme des Managers.

Der Vorteil der Kommandozeilen-Programme des Net-SNMP Managers ist die Möglichkeit, sie in größere Script-Programme, zum Beispiel Bash- oder Perl-Programme, einzubetten. So können auf schnellem und einfachem Weg umfangreichere und komplexere SNMP-Anfragen mit beliebigem Ausgabeformat erstellt werden.

Darüber hinaus verfügt der Net-SNMP Manager noch über einige weitere „Second-Level"-Kommandozeilen-Programme, die den Umgang mit SNMP-Befehlen für bestimmte Aufgaben vereinfachen.
So eignet sich das Kommandozeilen-Programm „snmptable" sehr gut für die Ausgabe von Tabellen, da es die Managementinformation in einer übersichtlicheren Tabellenstruktur auf der Konsole ausgibt und nicht einfach nur untereinander auflistet. Das Programm „snmptable" verwendet unter SNMPv1 das Programm „snmpwalk" und ab SNMPv2c das Programm „snmpbulkwalk".
Net-SNMP verfügt auch über eine SNMP-Variante des bekannten Netzwerkstatus-Programms „netstat", mit dem Namen „snmpnetstat", das den Netzwerkstatus und einige Konfigurationensinformationen des Netzwerks via SNMP vom Agenten ausliest und dem Manager anzeigt.
Mit dem Programm „snmpdf" werden dem Manager via SNMP alle eingehängten Partitionen des Agenten-Systems angezeigt, sowie die gesamte, belegte und freie Speicherkapazität der Partitionen.
Darüber hinaus verfügt der Net-SNMP Manager noch über Programme, die die Konfiguration und Analyse des SNMP-Agenten via SNMP ermöglichen. Hier ist zum Beispiel „snmpusm" und „snmpvacm", zum konfigurieren von Zugriffsrechten und -sichten, sowie „snmptest" und „snmpcheck", letzteres mit grafischer Ausgabe, zum Testen eines SNMP Agenten, zu erwähnen.

Das folgende Bash-Programm ist ein Beispiel für die Einbettung von Net-SNMP Kommandozeilen-Programmen des Managers. Es bekommt als Übergabeparameter die IP-Adresse des Agenten sowie den dort gültigen Community-Namen und fragt über mehrere SNMPv1-Befehle die Identifikation und den bisherigen Ein- und Ausgangsverkehr des Agenten ab. Im ersten Abschnitt des Programms wird der Name, die Beschreibung, der Kontakt und die Anzahl der Netzwerkschnittstellen vom Agenten jeweils mit einem SNMPv1-Befehl abgefragt. Im zweiten Teil des Programms läuft dann eine While-Schleife über die zuvor ermittelte Anzahl der Netzwerkschnittstellen und fragt von diesen die Menge der bisher ein- und ausgegangen Bytes ab.

```
#!/bin/bash

AGENT_IP=$1
COM=$2
AGENT_NAME=`snmpget  -Oqv  -v1 -c $COM  $AGENT_IP  system.sysName.0`
AGENT_DESCRIPTION=`snmpget  -Oqv  -v1  -c $COM $AGENT_IP  system.sysDescr.0`
AGENT_CONTACT=`snmpget  -Oqv  -v1  -c $COM  $AGENT_IP  system.sysContact.0`
IFNUMBER=`snmpget  -Oqv -v1  -c $COM  $AGENT_IP  interfaces.ifNumber.0`

echo ""
echo "Agent: $AGENT_NAME   Contact: $AGENT_CONTACT"
echo "$AGENT_DESCRIPTION"
echo ""

set -- $IFNUMBER
ifnum=$1
count=0
echo "Interfaces: IN             OUT"

while [ "$count" -lt "$ifnum"  ]
do
  count=$((count+1))
  IFDESCR=`snmpget -Oqv -v1 -c $COM $AGENT_IP ifDescr.$count`
  IFINOCTS=`snmpget -Oqv -v1 -c $COM $AGENT_IP ifInOctets.$count`
  IFOUTOCTS=`snmpget -Oqv -v1 -c $COM $AGENT_IP ifOutOctets.$count`

  echo "$count: $IFDESCR : $IFINOCTS Byte, $IFOUTOCTS Byte"
done

exit 0
```

Listing 6.3: Bash-Script „snmpQuery.sh" mit eingebundenen Net-SNMP Anwendungen des Managers

Wird dieses Bash-Programm in eine Datei mit Namen „snmpQuery.sh“ abgelegt und werden die Ausführungsrechte entsprechend gesetzt (chmod 755 snmpQuery.sh), so kann es wie folgt ausgeführt werden.

```
Manager # ./snmpQuery.sh  10.0.0.2  public

AGENT: Agent   Contact: admin@agent
Linux Agent 2.6.12.6-1m #5 Mon Jul 3 23:32:43 UTC 2006 i686

Interfaces: IN          OUT
1: eth1 : 55138 Byte, 69862 Byte
2: eth0 : 102991 Byte, 103895 Byte
3: lo : 2016 Byte, 2016 Byte
```

Befehl 6.13: Ausführung und Ausgabe des Bash-Programms „snmpQuery.sh“

Die Abfrage ist an den SNMP-Agenten mit der IP-Adresse 10.0.0.2 und dem Community-Namen public gerichtet und gibt in dieser Reihenfolge Name, Kontaktinformation, System-Beschreibung und die bisher ein- und ausgegangenen Bytes der einzelnen Netzwerkschnittstellen des Agenten aus.

Da hier jedes Management-Objekt über einen eigenen SNMP-Befehl abgefragt wird, ist der Datenverkehr, der von diesem Programm erzeugt wird, relativ hoch. Das Programm kann bezüglich der Menge der zu übertragenen Daten erheblich optimiert werden, wenn über einen SNMP-Befehl mehrere Objekte abgefragt werden.

6.4.3 Der Net-SNMP Trap-Server

Net-SNMP beinhaltet eine Serveranwendung die einkommende Meldungen von Agenten verarbeiten kann. Diese Serveranwendung, mit Namen „snmptrapd“, protokolliert eingehende Trap- oder Inform-Meldungen von Agenten und antwortet auf letztere. Zusätzlich können externe Bash- oder Perl-Programme in einer Konfigurationsdatei des Trap-Servers angegeben werden, die beim Empfang einer bestimmten Meldung ausgeführt werden und den Inhalt der Meldung dann weiterverarbeiten können.
Mit dem Befehl „snmptrapd --help“ wird eine Liste von möglichen Startkonfigurationen angezeigt.

Der folgende Befehl startet den Trap-Server und gibt die eingehenden Meldungen direkt auf der Konsole aus.

```
Manager # snmptrapd -f -Lo
```

Befehl 6.14: Starten des Trap-Servers „snmptrapd“ mit Konsolenausgabe

Standardmäßig lauscht der Net-SNMP Trap-Server auf dem Netzwerkport 162 auf eingehende SNMP-Meldungen.

Für den Trap-Server kann beim Start eine Konfigurationsdatei angegeben werden. In dieser Datei können zum Beispiel SNMP-Trap Zugriffsnamen definiert werden. Es können auch externe Programme mit zu erwartenden Meldungen verknüpft werden, die dann bei Eingang der Meldung ausgeführt werden. Dem Trap-Server kann beim Start auch eine explizite Log-Datei mit angegeben werden, in die alle eingehenden Meldungen protokolliert werden. Die Konfigurationsdatei, hier im Beispiel „snmptrapd.conf“, wird mit der Option -c angegeben. Die Log-Datei, hier snmptrapd.log, mit der Option -L für „Logging“ und f für „Filelogging“.

```
Manager # snmptrapd -c /etc/snmp/snmptrapd.conf -Lf /var/log/snmptrapd.log
```

Befehl 6.15: Starten des Trap-Servers „snmptrapd“ mit Konfigurations- und Log-Datei

Alle Anweisungen für die Konfigurationsdatei „snmptrapd.conf“ können über den Befehl „snmptrapd -H“ aufgelistet werden. Eine ausführliche Beschreibung aller möglichen Anweisungen für die Konfigurationsdatei findet sich in der Manual-Page der Datei, die mit dem Befehl „man snmptrapd.conf“ aufgerufen wird.

Mit dem Perl-Programm „snmpconf“, das im Net-SNMP Softwarepaket enthalten ist, kann über eine übersichtliche Menüsteuerung ebenfalls die Konfigurationsdatei für den Trap-Server erstellt werden.
In der Konfigurationsdatei können Zugriffsnamen, Community- oder SNMPv3-Benutzernamen, eingestellt werden, so dass nur die Meldungen vom Trap-Server verarbeitet werden, die einen passenden Zugriffsnamen besitzen. Die Net-SNMP Version 5.2.2 ist bei der Behandlung von SNMPv1 und -v2c Meldungen etwas abweichend zur Version 5.3.1. Bei der muss eine „authCommunity“ mit einer Zuteilung der Zugriffsrechte „log“ (Protokollieren), „execute“ (Ausführen) oder „net“ (Weiterleiten) angegeben werden und nur die

Meldungen der definierten „authCommunities“ werden angenommen. Bei der hier verwendeten Version 5.2.2 ist das aber noch nicht der Fall. Hier wird jede Meldung zumindest protokolliert. Die Zugriffsrechte unter dieser Version können wie folgt in der Konfigurationsdatei angegeben werden.

```
## ACCESS CONTROL ##
#SNMPv3
createUser mytrapuser MD5 "md5_password" DES "des_password"
createUser trUser MD5 "trap_password" DES "other_password"

#SNMPv1, -v2c
trapcommunity trapCom
trapcommunity trapatonie
trapcommunity publictrap
```

Listing 6.4.1: Konfiguration der Zugriffsrechte in der Datei „snmptrapd.conf“ des Trap-Server

Um externe Programme nach dem Erhalt einer Meldung starten zu können muss die Anweisung „traphandle“ und die OID des Objektes, das die Meldung verursacht, zusammen mit dem auszuführenden Programm verknüpft werden. Im angegebenen Beispiel in Listing 6.4.2 werden durch empfangene Meldungen, ausgelöst durch unterschiedliche Objekte, verschiedene Programme oder die gleichen Programme mit unterschiedlichen Parametern gestartet.

```
## TRAP HANDLE ##
#            OID            ausf. Anwendung(opt)  Scriptname      Argumente
traphandle SNMPv2-MIB::coldStart                 /tmp/trapProg.sh start
traphandle NET-SNMP-AGENT-MIB::nsNotifyShutdown /tmp/trapProg.sh shutdown
traphandle IF-MIB::linkDown                      /tmp/ifTrapProg.sh down
traphandle IF-MIB::linkUp                        /tmp/ifTrapProg.sh up
traphandle .1.3.6.1.4.1.2021.2.1.100             /tmp/progieControl.sh Error

traphandle default          /bin/bash            /tmp/trapReceiver
```

Listing 6.4.2: Konfiguration des Trap-Servers für das Verarbeiten von Meldungen

Im Beispiel startet der Trap-Server das Bash-Programm „trapProg.sh“, das im Verzeichnis „/tmp/“ abgelegt ist, wenn er eine SNMP-Meldung empfängt, deren Inhalt besagt, dass der Agent gerade gestartet ist oder kurz davor ist herunterzufahren. Je nachdem welche SNMP-Meldung hier eintrifft wird das Argument „start“ oder „shutdown“ dem Programm übergeben. Die nächsten zwei Einträge starten das Bash-Programm „ifTrapProg.sh“ mit den entsprechenden Parametern, falls über eine SNMP-Meldung des Agenten das ein- oder ausschalten einer Netzwerkverbindung gemeldet wird. Der Eintrag in der vorletzten Zeile wartet auf SNMP-Meldungen die durch die Net-SNMP eigene Prozessbeobachtung des Agenten zustande gekommen sind. Der Net-SNMP Agent kann Prozessinstanzen beobachten und eine Meldung senden falls zu viele oder zu wenige Instanzen eines Prozesses existieren. Bei Eingang einer solchen Meldung veranlasst der Trap-Server mit dieser Konfiguration das Starten des Bash-Programms „progieControl.sh“ mit dem Argument „Error“. Der Eintrag in der letzten Zeile führt für alle eingehenden Meldungen, die einen hier registrierten Zugriffsnamen besitzen, das Programm „trapReceiver“ aus.

Die praktische Ausarbeitung der Studienarbeit ergab, dass bei den externen Programmen, die interpretiert werden müssen, eventuell auch die ausführende Anwendung angegeben werden muss, also zum Beispiel „/bin/perl“ für Perl-Programme oder „/bin/bash“ für Bash-Programme, wie in der letzten Zeile des obigen Beispiels dargestellt. Ob die ausführende Anwendung angegeben werden muss oder nicht ist wohl vom Namen, bzw. genauer von der Bezeichnung des Anhangs der Datei, in der das Programm abgelegt ist, abhängig. Bei Bash-Programmen, deren Dateien die Anhangsbezeichnung „.sh“ tragen, weiß der Trap-Server mit welcher Anwendung das Bash-Programm interpretiert werden muss. Fehlt der Dateianhang, kann der Trap-Server das Programm möglicherweise nicht ausführen. Unter Linux müssen die Programme natürlich auch die entsprechenden Ausführungsrechte besitzen (z.B.: chmod 755 'Programmname').

Es gibt auch professionelle Zusatzprogramme die an dieser Stelle angegeben werden können, zum Beispiel das Programm „SNMP Trap Translator“ (snmptt). Das Programm „snmptt“ wird in einem von Net-SNMP unabhängigen Open-Source Projekt gepflegt. Es übersetzt die eingehenden, stellenweise eher kryptisch wirkenden SNMP-Meldungen und ermöglicht auch die Einbindung von SQL-Datenbanken für die Protokollierung. Das Programm „snmptt“ wird in dieser Studienarbeit nicht weiter aufgeführt da es ein größeres Programm mit zusätzlichem Einrichtungs- und Konfigurationsaufwand ist.
Das Open-Source Projekt „snmptt“ ist unter der Adresse „http://snmptt.sourceforge.net“ im Internet zu finden.

6.5 Der Net-SNMP Agent

Die im Net-SNMP Softwarepaket enthaltene Serveranwendung „snmpd“ ist der SNMP-Agent. Diese Serveranwendung verwaltet die Objekte und macht deren Managementinformationen für SNMP-Management-Applikationen zugänglich. Sie kann auch die Werte der Management-Objekte überwachen und beim überschreiten von Grenzwerten SNMP-Meldungen an einen Manager absenden. Zum Umfang des Agenten gehört auch das Kommandozeilen-Programm „snmptrap“, mit dem Trap- und Inform-Meldungen gesendet werden können.

6.5.1 Konfiguration des Net-SNMP Agenten

In diesem Kapitel werden Beispiele für die grundlegende Konfiguration eines Net-SNMP-Agenten aufgezeigt. Die Konfiguration des Net-SNMP Agenten findet über die Datei „snmpd.conf“ statt. Darüber wo diese Konfigurationsdatei abgelegt werden soll gibt sich Net-SNMP unschlüssig. Hier wird sie im Verzeichnis „/etc/snmp/“ abgelegt. Sie kann aber auch beim Start des Agenten explizit mit angegeben. Im Quelltext-Verzeichnis von Net-SNMP befindet sich eine Konfigurationsdatei mit Namen „EXAMPLE.conf“, in der viele Konfigurationsbeispiele aufgeführt sind. Zusätzlich gibt es auch noch das Perl-Programm „snmpconf“, mit dem mittels eines übersichtlichen Menüsystems ebenfalls die Konfigurationsdatei für den Net-SNMP Agenten erstellt werden kann. Über dieses Programm kann auch mit dem folgenden Befehl ein Frage/Antwort-Dialog gestartet werden, über den dann die Konfigurationsdatei für den Agenten einfach erstellt werden kann.

```
Agent # snmpconf -g basic_setup
```

Befehl 6.16: Erstellen einer Konfigurationsdatei für den Agenten mittels „snmpconf“

Der Net-SNMP Agent wird mit dem folgenden Befehl gestartet. Mit der Befehlsoption -c wird die Konfigurationsdatei des Agenten, hier „snmpd.conf“, angegeben. Mit der Befehlsoption -Lf wird das verhalten des Agenten in die Datei „snmpd.log“ protokolliert.

```
Agent # snmpd -c /etc/snmp/snmpd.conf -Lf /var/log/snmpd.log
```

Befehl 6.17: Starten des Net-SNMP Agenten

Sämtliche Startoptionen des Net-SNMP Agenten werden mit dem Befehl „snmpd --help“ aufgezeigt

Es gibt eine sehr große Anzahl an Konfigurationsanweisungen für den Net-SNMP Agenten, die sich in die folgenden vier Kategorien einteilen lassen.

Es gibt Konfigurationsanweisungen die:

- SNMPv1/-v2c Community-Namen und SNMPv3-Benutzernamen sowie deren Zugriffsrechte konfigurieren.
- bestimmte Objekte, bzw. deren Managementinformationen, des Agenten konfigurieren können.
- die Überwachung von Management-Objekten konfigurieren und festlegen, wann und wegen welchem Objekt an welchen Manager Meldungen gesendet werden.
- Erweiterungen des Agenten konfigurieren, zum Beispiel den Anschluß an Sub-Agenten.

Eine Liste aller Konfigurationsanweisungen des Net-SNMP Agenten wird über den Befehl „snmpd -H“ aufgelistet. Eine ausführliche Beschreibung aller Konfigurationsanweisungen des Net-SNMP Agenten ist über die Manual-Page zu erhalten, die mit dem Befehl „man snmpd.conf“ aufgerufen werden kann.

In den folgenden Kapiteln werden nun einige grundlegende Konfigurationsanweisungen über Beispiele dargestellt.

6.5.1.1 Konfiguration von Zugriffsrechten

Die schnellste und einfachste Möglichkeit Zugriff auf die Objekte und deren Managementinformationen des Net-SNMP Agenten zu bekommen, ist die Konfiguration von SNMPv1/-v2c Zugriffsmöglichkeiten durch die Erstellung von „read-write“ (RW) oder „read-only“ (RO) Communities. Im folgenden Beispiel wird die RW-Community „public“ und die RO-Community „private“ erstellt, die keinerlei weitere Einschränkungen besitzen,

sowie die RO-Community „publicSys“, die lediglich Lesezugriff auf die Objekte der Gruppe „system“ der MIB-II besitzt und deren Anfragen nur aus dem IP-Bereich des 10.0.0.0/24er Netzes eingehen dürfen.

```
#              Community      NET             MIB-Part
rocommunity    public
rocommunity    publicSys      10.0.0.0/24     system
rwcommunity    private
```

Listing 6.5.1: Konfiguration für den SNMPv1 / -v2c Zugriff in der Datei „snmpd.conf“

Um einen SNMPv3 Benutzerzugriff zu konfigurieren müssen zwei Einträge in die Konfigurationsdatei gemacht werden. Ein Eintrag der den Benutzer erstellt und ihm ein Authentifizierungs- und ein Verschlüsselungspasswort zuteilt sowie ein weiterer Eintrag der den Benutzer als RO- oder RW-Benutzer definiert. Im folgenden Beispiel werden die zwei SNMPv3-Benutzer „myuser“ und „mySysUser“ mit Authentifizierungs- und Verschlüsselungspasswörtern erstellt. Der Benutzer „myuser“ ist als schreibberechtigter Benutzer definiert und muss sich immer mit beiden Passwörtern am Agenten anmelden. Der Benutzer „mySysUser“ darf auf dem Agenten nur die Managementinformationen der Objekte der Gruppe „system“ lesen und benötigt zur Anmeldung lediglich sein Authentifizierungspasswort.

```
#              Username       authentification-passw.     privacy-passw.
createUser     myuser MD5 "auth_password" DES "priv_password"
createUser     mySysUser MD5 "my_password" DES "other_password"

#              Username       access         MIB-Part
rwuser         myuser         authPriv
rouser         mySysUser      authNoPriv     system
```

Listing 6.5.2: Konfiguration für den SNMPv3 Benutzerzugriff in der Datei snmpd.conf

Mit dem Programm „net-snmp-config“ und dessen Befehlsoption „--create-snmpv3-user“ kann ein SNMPv3-Benutzer über einen Frage/Antwort-Dialog angelegt werden. Das Programm schreibt die für die Erstellung notwendigen Konfigurationsanweisungen in zwei verschiedene Konfigurationsdateien, die an unterschiedlichen Stellen im Agenten-System zu finden sind. Hier im Beispiel wird der oben bereits angelegte „myuser“ durch das Programm „net-snmp-config“ angelegt. Das Programm schreibt den Benutzernamen in die Datei „snmpd.conf“, die sich im Verzeichnis „/usr/local/share/snmp/“ befindet und die Passwörter in die Datei „snmpd.conf“, die sich im „persistent“-Verzeichnis von Net-SNMP, hier „/var/net-snmp/“, befindet. Der Übersicht wegen können die dortigen Einträge auch gelöscht und in die Konfigurationsdatei „snmpd.conf“ im Verzeichnis „/etc/snmp/” eingetragen werden.

```
Agent # net-snmp-config --create-snmpv3-user
Enter a SNMPv3 user name to create:
myuser
Enter authentication pass-phrase:
auth_password
Enter encryption pass-phrase:
priv_password
adding the following line to /var/net-snmp/snmpd.conf:
  createUser myuser MD5 "auth_password" DES "priv_password"
adding the following line to /usr/local/share/snmp/snmpd.conf:
  rwuser myuser
```

Befehl 6.18: Einrichten eines SNMPv3-Benutzers mit dem Programm „net-snmp-config“

Benutzt der SNMPv3-Benutzer sein Verschlüsselungspasswort, so ist die komplette SNMP-Nachricht verschlüsselt und von Protocol-Analyzern wie „Ethereal“ nicht einzusehen. Ohne Verschlüsselung kann die komplette SNMP-Nachricht von Dritten, die sie zwischen dem Agenten und dem Manager abfangen, gelesen werden. Bei der Installation von Net-SNMP wurde hier „Openssl“ für die Verschlüsselung angegeben. Das Softwarepaket Net-SNMP beinhaltet selbst keinen eigenen Verschlüsselungsalgorithmus.

Diese grundlegenden Zugriffskonfigurationen von Net-SNMP sind schnell eingerichtet, bieten allerdings nur eine sehr grobe Zugriffsverwaltung auf die einzelnen Objekte des Agenten und deren Informationen.
Dank des „View-Based Access Control Models“ (VACM) der SNMP-Spezifikation ist aber eine wesentlich detailliertere Zugriffsverwaltung auch mit Net-SNMP möglich. Die Konfiguration von VACM ist allerdings ein wenig umfangreicher.

Die folgende Konfiguration erstellt vier Communities mit unterschiedlichen Zugriffsrechten auf verschiedene Management-Objekte. Die Communities „myprivate“ und „admin“ dürfen alles, aber „myprivate“ nur vom „localhost“ und die Community „admin“ nur von der IP-Adresse 10.0.0.1 aus. Die Community „mynetwork“ darf nur lesend auf die Management-Objekte der Gruppen der MIB-II zugreifen und die Community „allpublic“ darf nur die Systembeschreibung (SNMPv2-MIB::sysDescr.0) via SNMPv1 abfragen. Außerdem wird im weiteren Verlauf noch der SNMPv3 Benutzer „InterfaceAdmin“ erstellt, der vollen Zugriff auf die Management-Objekte der Gruppe „interfaces“ besitzt.

Um VACM zu konfigurieren nutzt Net-SNMP vier Schlüsslewörter,

- com2sec ordnet Community-Namen bestimmte Security-Namen zu (gilt nur für IPv4-Netze)
- group Fasst die Security-Namen mit Sicherheitsmodellen zu Gruppen zusammen
- view Erstellt Sichten über den ISO-Registrierungsbaum, bzw. die MIB
- access Regelt den Zugriff und verknüpft die erstellten Gruppen mit den Zugriffsrechten und -sichten

Im ersten Abschnitt werden die Community-Namen und sogenannte Security-Namen für Zugriffe aus verschiedenen IPv4-Netzbereichen festgelegt. Für Zugriffe über Verbindungen anderer Art gibt es andere Anweisungen, zum Beispiel für Zugriffe über Unix-Sockets die Anweisung com2secunix.

```
# Access Control, define security names
######################################################################
#             sec.-name     source        community
com2sec       manager       localhost     myprivate
com2sec       manager       10.0.0.1      admin
com2sec       network       10.0.0.0/24   mynetwork
com2sec       world         default       allpublic
#
createUser InterfaceAdmin MD5 "ifmd5_password" DES "ifdes_password"
```

Listing 6.5.3: Konfiguration von VACM Security-Namen in der Datei „snmpd.conf“

Hier wird der Vollständigkeit wegen auch gleich der SNMPv3-Benutzer „InterfaceAdmin“ erstellt.

Im zweiten Abschnitt werden die zuvor definierten Security-Namen mit Sicherheitsmodellen zu Gruppen zusammengefasst. Die Security-Namen entsprechen übrigens auch SNMPv3-Benutzernamen, das bedeutet, dass ab hier auch SNMPv3-Benutzer in das VACM-Modell eingebunden werden können.

```
# Map security names to groups
######################################################################
#      groupname     sec.-model    sec.-name
group  untrusted     v1            world
group  ROgroup       v1            network
group  ROgroup       v2c           network
group  RWgroup       v1            manager
group  RWgroup       v2c           manager
group  IFgroup       usm           InterfaceAdmin
```

Listing 6.5.4: Konfiguration von VACM Gruppen in der Datei „snmpd.conf“

Im dritten Abschnitt der VACM Konfiguration werden verschiedene Sichten über den ISO-Registrierungsbaum erstellt. Diese Sichten bilden fest definierte Bereiche über der „Management Information Base“ des Agenten. Nur auf die Objekte innerhalb der zugewiesenen Sicht ist einem Benutzer dann der Zugriff erlaubt.

```
# create a view to let the groups have the rights to
######################################################################
#      name          incl/excl     subtree
view   all           included      .1
view   descr         included      system.sysDescr
view   mib-2         included      iso.org.dod.internet.mgmt.mib-2
view   ifview        included      interfaces
```

Listing 6.5.5: Konfiguration von VACM-Sichten in der Datei „snmpd.conf“

Im vierten und letzten Abschnitt schließlich wird festgelegt welche Gruppen mit welchen Zugriffsrechten auf welche Sichten zugreifen dürfen.

```
# grant the groups access to the views with different write permissions
##################################################################
#       name       context    model    level      match    read     write    notif
access  untrusted     ""       any      noauth     exact    descr    none     none
access  ROgroup       ""       any      noauth     exact    mib-2    none     none
access  RWgroup       ""       any      noauth     exact    all      all      all
access  IFgroup       ""       usm      authPriv   exact    ifview   ifview   ifview
```

Listing 6.5.6: Konfiguration von VACM Access Policies in der Datei „snmpd.conf“

Besitzt ein Agent diese Konfiguration, so bekommt zum Beispiel ein Mitglied der „allpublic“-Community auf die folgenden Anfragen an den Agenten die folgende Ausgaben auf dem Manager.

```
Manager # snmpwalk -v2c -c allpublic 10.0.0.2
Timeout: No Response from 10.0.0.2

Manager # snmpwalk -v1 -c allpublic 10.0.0.2
SNMPv2-MIB::sysDescr.0 = STRING: Linux Agent 2.6.12.6-1m #5 Mon Jul 3 23:32:43 UTC 2006 i686
End of MIB

Manager # snmpwalk -v1 -c allpublic 10.0.0.2 interfaces
End of MIB
```

Befehl 6.19: „snmpwalk“ -usgabe der VACM-Community „allpublic“

Die „allpublic“-Community erhält vom Agenten keine Antwort auf SNMPv2-Befehle. Sie erhält nur auf SNMPv1-Befehle Antwort und hier auch nur die Systembeschreibung des Agenten. Das Kommandozeilen-Programm „snmpwalk“, das bei fehlender Angabe einer OID im Parameterfeld die „mib-2“ OID einsetzt, liefert lediglich die Managementinformation des Objektes „SNMPv2-MIB::sysDescr.0“. Auf die Management-information anderer Objekte hat die „allpublic“-Community überhaupt keinen Zugriff.

6.5.1.2 Konfiguration von Managementinformationen

Net-SNMP erlaubt das einfache setzen von Managementinformationen einiger Objekte über Anweisungen in der Konfigurationsdatei „snmpd.conf“. Mit den folgenden Konfigurationsanweisungen können die Management-Objekte der „system“ Gruppe der MIB-II, die den Agenten beschreiben, mit Management-informationen versehen werden. Die hier konfigurierte Managementinformation darf dann allerdings nicht mehr via dem SNMP SET-Befehl neu gesetzt werden. Net-SNMP erlaubt dann nur noch den lesenden Zugriff auf die Informationen dieser Objekte.

```
# System information
######################
syslocation "Down, in the Cyber World"
syscontact "The Agent's Admin admin@agent"
sysdescr "The virtual Agent"
sysname "Agent"
```

Listing 6.5.7: Konfiguration von Systeminformationen des Agenten in der Datei „snmpd.conf“

Es gibt auch noch einige andere Anweisungen, welche es erlauben neue Managementinformationen über die Konfigurationsdatei zu definieren. So ist es zum Beispiel auch möglich, Tabellen neue Zeilen mit Managementinformationen anzuhängen oder Netzwerkschnittstellen, die der Net-SNMP Agent bei der Installation nicht erkannt hat, nachträglich zu definieren.

6.5.1.3 Konfiguration der Objekt-Überwachung

Innerhalb der Konfigurationsdatei „snmpd.conf“ lässt sich auch die Überwachung von Management-Objekten konfigurieren. Für die Überwachungskonfiguration muss bei der Installation von Net-SNMP die Perl-Schnittstelle und das „DISMAN-EVENT-MIB-Modul“ eingebunden werden. Bei der neueren Net-SNMP Version 5.3.1 wird dies standardmäßig installiert und muss nicht mehr explizit angegeben werden.

Um SNMP-Meldungen beim Eintritt bestimmter Ereignisse zu senden, wird die IP-Adresse des Managers benötigt, auf dem ein Trap-Server die gesendeten Meldungen entgegen nehmen kann. Hat der Agent eine falsche Adresse oder läuft der Trap-Server des Managers nicht, so bemerkt dies der Agent für gewöhnlich nicht. Nur bei Inform-Meldungen merkt der Agent, wenn keine Gegenstelle erreicht wird, da er vom Manager eine Antwort erwartet. Die Einrichtung eines Trap-Servers wurde bereits in einem vorherigen Kapitel behandelt.

Über die Konfigurationsanweisungen „trapsink“, „trap2sink“ oder „informsink“ wird die IP-Adresse des Managers und ein Zugriffsname, Community-Name oder SNMPv3-Benutzername, angegeben. Die Anweisungen generieren, wie aus ihrer jeweiligen Bezeichnung herauszulesen, SNMPv1-, SNMPv2-Traps bzw. Informs. Die Meldungen werden an die angegebene IP-Adresse mit dem angegebenen Zugriffsnamen gesendet. Die Zugriffsnamen müssen auch dem Trap-Server des empfangenden Managers bekannt sein.

Das folgende Beispiel sendet alle ausgehenden SNMP-Meldungen an den Manager mit der IP-Adresse 10.0.0.1 und zwar dreifach, als SNMPv1- und SNMPv2-Trap mit dem Community-Namen „trapCom“, sowie als Inform mit dem Community-Namen „trapatonie“.

```
#Trap configuration
######################
#Version      destination   Community/User
trap2sink     10.0.0.1      trapCom
trapsink      10.0.0.1      trapCom
informsink    10.0.0.1      trapatonie
```

Listing 6.5.8: Konfiguration der Ziele für SNMP-Meldungen in der Datei „snmpd.conf“

Ohne eine solche Konfigurationsanweisungen verlässt keine SNMP-Meldung den Agenten.

Der Net-SNMP Agent bietet die Möglichkeit die Überwachung mancher Ressourcen des Agenten-Systems, wie zum Beispiel der vorhandenen Speicherkapazität oder der laufenden Anwendungsprozesse, mit wenigen Anweisungen einfach in der Konfigurationsdatei des Agenten zu konfigurieren.
So kann mittels der Anweisung „proc“ die Überwachung der Existenz und Anzahl von Instanzen von laufenden Anwendungsprozessen konfiguriert werden. Dazu wird der Name des Prozesses sowie die Anzahl der maximal und minimal zulässigen Prozessinstanzen angegeben. Der Net-SNMP Agent generiert dann automatisch unterhalb der OID „iso.org.dod.internet.private.enterprises.ucdavis.prTable“ (1.3.6.1.4.1.2021.2) die Objekte bezüglich der konfigurierten Prozesse. Mit der Konfigurationsanweisung „procfix“, die hier allerdings nicht weiter beachtet wird, kann auch angegeben werden was geschehen soll falls die Anzahl der Prozessinstanzen den zulässigen Bereich verlässt.

```
# Process checks
############################
#      proc.-name    max    min
proc   sshd          5      1
proc   ripd          1      1
proc   zebra         1      1
```

Listing 6.5.9: Konfiguration der Überwachung von Prozessinstanzen in der Datei „snmpd.conf“

Diese Beispielkonfiguration überwacht, dass mindestens ein ssh-Dienst vorhanden ist und maximal fünf ssh-Dienste gleichzeitig laufen dürfen. Hier muss dazu erwähnt werden, dass mit jeder eingehenden ssh-Verbindung auch ein ssh-Dienst gestartet wird. Diese Angabe überprüft also auch, dass generell eine ssh-Verbindung zum Agenten-System möglich ist und nicht mehr als vier weitere ssh-Verbindungen zur gleichen Zeit zum Agenten aufgebaut sind. Des weiteren wird überwacht, dass genau ein RIP-Routing-Dienst und ein Zebra-Dienst im Agenten-System laufen.

Werden die beiden Werte der minimal und maximal zulässigen Prozessinstanzen auf 0 gesetzt, so gilt für die Anzahl der mindestens vorhanden Prozessinstanzen 1 und für die der maximal vorhanden unendlich.

Unterhalb der OID des Objektes „prTable“ ist nun die folgende Managementinformation via SNMP zu lesen, wenn von den oben angegebenen Anwendungsprozessen, zwei „ssh“-Dienste, ein „zebra“- und kein „ripd“-Dienst im Agenten-System laufen.

```
Manager # snmptable -v 1 -c private localhost private.enterprises.ucdavis.prTable

SNMP table: UCD-SNMP-MIB::prTable
prIndex prNames prMin prMax prCount prErrorFlag prErrMessage                 prErrFix prErrFixCmd
  1     sshd    1     5     2       0                                         0
  2     ripd    1     1     0       1           Too few ripd running (# = 0)  0
  3     zebra   1     1     1       0                                         0
```

Befehl 6.20: Abfrage der Prozessüberwachungstabelle „ucdavis.prTable“

Es wird nun allerdings nicht automatisch eine SNMP-Meldung an den Manager generiert. Dies muss über weitere Anweisungen in der Konfigurationsdatei des Net-SNMP Agenten erst noch aktiviert werden.

Einen ähnlich einfachen Mechanismus bietet Net-SNMP für das einbinden von Partitionen und der Überwachung der frei verfügbaren Speicherkapazität (OID: ucdavis.dskTable, 1.3.6.1.4.1.2021.9), sowie Dateien und deren maximalen Größe (OID: ucdavis.fileTable, 1.3.6.1.4.1.2021.15).

```
# Disk/File checks
############################
#      path                 min free(KB | %)
disk   /opt/                10000
disk   /                    5%

#      filename             max size(KB)
file   /tmp/mainlog.log     500
```

Listing 6.5.10: Konfiguration der Überwachung der Speicherkapazität und Dateigröße

Darüber hinaus gibt es noch weitere Möglichkeiten auch andere Ressourcen des Agenten-Systems über Konfigurationsanweisungen durch den Net-SNMP Agenten überwachen zu lassen, wobei parallel dazu, durch die SNMP-Spezifikation ohnehin die Möglichkeit besteht über die Definition von Notification-Objekten, jede beliebige Ressource des Agenten-Systems zur Überwachung einzubinden.

Die obigen Konfigurationsanweisungen binden die entsprechenden Ressourcen des Agenten-Systems zur Überwachung in die MIB ein, erzeugen aber nicht automatisch beim überschreiten der Grenzwerte eine SNMP-Meldung. Dies kann über die folgende Konfigurationsanweisung allgemein aktiviert werden. Zuvor aber muss noch über die Anweisung „agentSecName“ ein entsprechender Benutzer generiert werden.

```
# Monitoring
############################
createUser      trUser MD5 "trap_password" DES "other_password"
agentSecName    trUser
rouser          trUser

defaultMonitors yes
```

Listing 6.5.11: Konfiguration für das Senden von Standard-Ereignismeldungen in der Datei„snmpd.conf“

Es können auch noch weitere Überwachungen, die SNMP-Meldungen erzeugen, durch solche Anweisungen aktiviert werden. Das sich zum Beispiel ein Net-SNMP Agent bei seinem Manager meldet, wenn ihn ein SNMP-Befehl mit einem ihm unbekannten Zugriffsnamen erreicht kann über die folgende Konfigurations-anweisung aktiviert werden.

```
authtrapenable 1
```

Listing 6.5.12: Konfiguration für das Senden einer Meldung bei fehlgeschlagener Authentifizierung

Eine flexiblere Einstellung der Überwachung von Management-Objekten bietet die Konfigurationsanweisung „monitor“. Damit lassen sich beliebige Management-Objekte und deren Werte-Abweichungen überwachen.

```
#       repeat   user    message-OID     message-OID    monitored OID  value
monitor -r 60 -u trUser -o sysUpTime.0 -o prErrMessage "process"   prErrorFlag != 0
```

Listing 6.5.13: Konfiguration für das Senden von Ereignismeldungen für fehlende Prozessinstanzen

Die Option -r gibt die Zeit in Sekunden an, in welcher der Agent das Senden der SNMP-Meldung im Zeitraum des Fehlerzustandes wiederholt. Standardmäßig sind hier 600 Sekunden eingestellt. Mit der nächste Option -u kann ein Zugriffsname angegeben werden. Hier wurde der SNMPv3-Benutzer „trUser“ erstellt unter dessen Namen die SNMP-Meldungen den Manager erreichen werden. Dort muss der SNMPv3-Benutzer „trUser“ ebenfalls im Trap-Server definiert sein. Die darauffolgenden OIDs, die mit der Option -o angegeben werden, sind zusätzliche, mit der Meldung gesendete Managementinformationen, die frei angegeben werden können. Am Ende steht das zu überprüfende Objekt und der Objekt-Wert.
Auch hierzu bietet die Manual-Page von „snmpd.conf“ eine wesentlich ausführlichere Beschreibung.

6.5.1.4 Konfiguration von Erweiterungen

Net-SNMP bietet die Möglichkeit den Agenten um Sub-Agenten zu erweitern oder externe Bash- oder Perl-Programme direkt einzubinden. Mit der Konfigurationsanweisung „exec“, die in der neueren Net-SNMP Version 5.3.1 durch die Anweisung „extend“ ersetzt wird, können solche externen Programme via SNMP-Befehl aufgerufen und deren Aus- und Rückgabewerte abgefragt werden.

```
# executables / scripts
############################
#             OID           name          scriptname           arguments
exec                        testprogie    /tmp/testprogie.sh
exec                        huhu          /bin/echo            how are you
exec   .1.3.6.1.4.1.5432.31 switchON      /tmp/switch.sh       ON
exec   .1.3.6.1.4.1.5432.32 switchOFF     /tmp/switch.sh       OFF
exec   .1.3.6.1.4.1.9876.51 cpuinfo       /bin/cat             /proc/cpuinfo
```

Listing 6.5.14: Konfiguration der Einbindung externer Programme über „exec“ in der Datei „snmpd.conf“

Wird hier keine OID angegeben, so wird die Managementinformation des eingebunden Programms standardmäßig von Net-SNMP unterhalb der OID „iso.org.dod.internet.private.enterprises.ucdavis.extTable“ (1.3.6.1.4.1.2021.8) in die „Management Information Base“ eingegliedert. Unterhalb der OID finden sich dann auch Rück- und Ausgabewerte, weswegen hier auch nur Programme in Frage kommen, die deterministisch und terminierend sind, da sich sonst der Net-SNMP Agent beim warten auf die Programmausgabe- bzw. Rückgabewerte aufhängt.

Als Beispiel ist hier nun ein kleines Bash-Programm mit dem Namen „testprogie.sh“ beschrieben.

```
#!/bin/bash
echo "Hello World"
exit 35
```

Listing 6.6: Beispiel eines Script-Programms „testprogie.sh“ zum Einbinden in den Net-SNMP Agenten

Das Programm gibt lediglich die Textzeile „Hello World“ auf der Konsole aus, wenn es zuvor mit den notwendigen Ausführungsrechten versehen wurde (chmod 755 testprogie.sh).

```
Agent # /tmp/testprogie.sh
Hello World
```

Befehl 6.21: Ausgabe des Bash-Scripts „testprogie.sh“

Das Programm „testprogi.sh“ kann über das Abrufen des Objektes „.ucdavis.extTable.extEntry.extCommand.1“ (1.3.6.1.4.1.2021.8.1.3.1) via SNMP-Befehl vom Manager aus auf dem Agenten ausgeführt werden. Beim aufrufen der kompletten Tabelle mit allen Managementinformationen des Programms, wie hier gezeigt, wird das Programm „testprogie.sh“ natürlich ebenfalls ausgeführt.

```
Manager # snmptable -v 1 -c private 10.0.0.2 .1.3.6.1.4.1.2021.extTable
SNMP table: UCD-SNMP-MIB::extTable

extIndex  extNames   extCommand                 extResult   extOutput    extErrFix   extErrFixCmd
    1     testprogi  /tmp/testprogi.sh             35       Hello World     0
    2     huhu       /bin/echo      how are you    0        how are you     0
[...]
```

Befehl 6.22: Anzeigen der Tabelle der durch „exec“ ausführbaren Programme auf dem Agenten

Darüber hinaus kann der Net-SNMP Agent noch über die Konfigurationsanweisungen „pass“ und „passpersist“ erweitert werden. Diese beiden Anweisungen erlauben das Einbinden weitaus komplexere Programme, die einen eigenen Teil der „Management Information Base“ mit eigenen Management-Objekten verwalten. Diese Programme verwalten externe „MIB-Bibliotheken“. Der Net-SNMP Agent leitet die betreffenden SNMP-Befehle an die Programme weiter, sofern die angesteuerte OID zu einem Objekt aus deren „MIB-Bibliothek“ gehört. Der Unterschied zwischen den beiden Konfigurationsanweisungen „pass“ und „passpersist“ ist, dass ein Programm hinter der „pass“ Anweisung nur einmal kurz läuft und dann endet (one shot), während ein Programm hinter der „passpersist“ Anweisung nach dem Aufruf dauerhaft weiter laufen kann (persistent).
Im Quellcode-Verzeichnis von Net-SNMP finden sich zwei Beispiel-Programme, das Bash-Programm „passtest“ und das Perl-Programm „pass_persisttest“, die jeweils Managementinformationen über eigene Objekte in einer eigenen „MIB-Bibliothek“ verwalten.

```
# executables / scripts
############################
#                    oid                    scriptname
pass          .1.3.6.1.4.1.2021.255.1     /tmp/passtest
pass_persist  .1.3.6.1.4.1.2021.255.2     /tmp/pass_persisttest
pass          .1.3.6.1.4.1.99999.1        /tmp/vtyshQuery.sh
```

Listing 6.5.15: Konfiguration von extern verwalteten MIB-Bibliotheken über „pass“ und „pass_persist“

Hier wird nun als Beispiel das kleine Bash-Script „vtyshQuery.sh“ gezeigt, das hier im Verzeichnis „/tmp/“ abgelegt ist. Das Script verwaltet zwei Objekte, die unter der OID .1.3.6.1.4.1.99999.1 mit SNMP abgefragt werden können. Die Information des ersten Objektes ist eine Auflistung aller Quagga Routing-Dienste, die auf dem Agenten laufen. Dafür führt das Script, nach eingehen einer entsprechenden SNMP-Anfrage, den Befehl „vtysh -c "show daemons"“ in der Konsole aus. Die Ausgabe wird an den Net-SNMP Agenten übergeben, der die Information dann an den SNMP-Manager weiterleitet. Die Information des zweiten Objektes ist ein einfacher, fester Integerwert.

```
#!/bin/sh -f
PATH=$path:/bin:/usr/bin:

PLACE=".1.3.6.1.4.1.99999"
REQ="$2"
if [ "$1" = "-n" ]; then
  case "$REQ" in
    $PLACE)      RET=$PLACE.1 ;;
    $PLACE.1)    RET=$PLACE.1.1 ;;
    $PLACE.1.1)  RET=$PLACE.1.2 ;;
    *)               exit 0 ;;
  esac
else
  case "$REQ" in
    $PLACE)    exit 0 ;;
    *)         RET=$REQ ;;
  esac
fi

echo "$RET"
case "$RET" in
  $PLACE.1.1) echo "string"; vtysh -c "show daemons"; exit 0 ;;
  $PLACE.1.2) echo "integer"; echo "123456"; exit 0 ;;
  *) echo "string"; echo "ack... $RET $REQ"; exit 0 ;;
esac
```

Listing 6.7: Beispiel für ein durch „pass“ angesteuertes Bash-Script „vtyshQuery.sh“

Der erste Teil des Programms „vtyshQuery.sh“ definiert die logische Struktur der externen MIB-Bibliothek. Im unteren Teil des Scripts werden dann die Managementinformationen der Objekte definiert.
Das Programm „vtyshQuery.sh“ muss mit den entsprechenden Ausfürungsrechten ausgestattet sein, um zu funktionieren (z.B.: chmod 755 vtyshQuery.sh).

Diese Objekte können vom Manager aus nur über ihre nummerische OID angesteuert werden, so lange keine MIB-Datei mit den SMI-Definitionen der Objekte vorliegt.

```
Manager # snmpwalk -v1 -c public 10.0.0.2 .1.3.6.1.4.1.99999.1
SNMPv2-SMI::enterprises.99999.1.1 = STRING: " zebra ripd"
SNMPv2-SMI::enterprises.99999.1.2 = INTEGER: 123456
```

Befehl 6.23: SNMP-Abfrage der durch das Script „vtyshQuery.sh“ verwalteten Objekte

Eine weitere Möglichkeit externe und völlig unabhängige Anwendungsprogramme an den Net-SNMP Agenten anzubinden, um deren Managementinformationen via SNMP zu verwalten, sind die Erweiterungsprotokolle SMUX und AgentX. Diese Anwendungsprogramme werden dann im allgemeinen Sub-Agenten genannt.
Ein SMUX-Agent, auch SMUX-Peer genannt, wird über die entsprechende Konfigurationsanweisung mit einer darauffolgenden OID an den Master SNMP-Agenten angekoppelt. Diese OID ist aber nicht die OID eines Objektes des SMUX-Agenten, welches durch SNMP-Befehle angesteuert werden kann, sondern sie dient lediglich der Kommunikationsverbindung zwischen Master SNMP-Agent und SMUX-Agent. Die Quagga Routing-Dienste können sich via SMUX an den Net-SNMP Agenten ankoppeln. Jeder Quagga Routing-Dienst verfügt über eine eigene OID für die Kommunikation, die in der Konfigurationsdatei des Dienstes und auch in der Konfigurationsdatei des Net-SNMP Agenten angegeben sein muss. Ein Passwort, ähnlich der Funktion des SNMPv1/-v2c Community-Namens, kann mit übergeben werden. Die Angabe eines Passwortes verursachte aber während der praktischen Ausarbeitung dieser Studienarbeit hin und wieder Verbindungsprobleme.
Der Master Net-SNMP-Agent lauscht standardmäßig auf Port 199 des „localhost“. Mit der Konfigurationsanweisung „smuxsocket“ können aber auch andere IP-Adressen und Ports eingebunden werden.

Hier im Beispiel werden nun alle SMUX-Peers des Quagga Routers angegeben. Jeder Quagga Routing-Dienst besitzt eine eigene SMUX Kommunikationsverbindung zum Master Net-SNMP Agenten, da die Dienste ja auch sonst unabhängig voneinander funktionieren. Aus den angegebenen Passwörtern kann abgelesen werden, welcher der Quagga Routing-Dienste über welche OID eingebunden wird.

```
# executables / SMUX PEERS
############################
#                   oid                   password
smuxpeer      .1.3.6.1.4.1.3317.1.2.1     quagga_zebra
smuxpeer      .1.3.6.1.4.1.3317.1.2.2     quagga_bgpd
smuxpeer      .1.3.6.1.4.1.3317.1.2.3     quagga_ripd
smuxpeer      .1.3.6.1.4.1.3317.1.2.5     quagga_ospfd
smuxpeer      .1.3.6.1.4.1.3317.1.2.6     quagga_ospf6d
# smuxsocket tcp:127.0.0.1:199
```

Listing 6.5.16: Konfiguration von Quagga SMUX-Peers in der Datei „snmpd.conf“

Diese Daten können der offiziellen Quagga Dokumentation entnommen werden.

In gleicher Reihenfolge wird hier nun noch aufgelistet unter welcher OID die Managementinformationen der einzelnen Quagga Routing-Dienste via SNMP-Befehl zu erreichen ist.

```
# Dienst      Abfrage-OID                OID ( klartext )
# zebra       .1.3.6.1.2.1.4.24    iso.org.dod.internet.mgmt.mib-2.ip.ipForward
# ospfd       .1.3.6.1.2.1.14      iso.org.dod.internet.mgmt.mib-2.ospf
# bgpd        .1.3.6.1.2.1.15      iso.org.dod.internet.mgmt.mib-2.bgp
# ripd        .1.3.6.1.2.1.23      iso.org.dod.internet.mgmt.mib-2.rip2
# ospf6d      .1.3.6.1.3.102       iso.org.dod.internet.experimental.ospfv3
```

Listing 6.8: OID der Quagga beiligenden MIBs (MIB-Module)

Für die hier aufgeführten Gruppen liefern die Quagga-Autoren MIB-Dateien im Quelltext-Verzeichnis von Quagga mit, so dass vom Net-SNMP Manager aus die Management-Objekte der Router auch über deren Namen angesteuert werden können. Die MIB-Dateien müssen in das entsprechende Standard-MIB Verzeichnis kopiert und über die Umgebungsvariable MIBS (export MIBS="ALL") den Net-SNMP Programmen bekannt gemacht werden. Natürlich muss SMUX auch in den Konfigurationsdateien der Routing-Dienste eingestellt werden, was in Kapitel 5.7.2 behandelt wurde.

Das zweite hier behandelte und von Net-SNMP unterstützte Erweiterungsprotokoll ist AgentX. Über die Konfigurationsanweisung „master agentx“ wird die AgentX-Schnittstelle des Net-SNMP-Agenten aktiviert. Standardmäßig lauscht der Net-SNMP Agent auf AgentX-Erweiterungen über eine „IPC“-Schnittstelle (Interprozesskommunikation), doch mit der Anweisung „agentXSocket“ können auch IP-Adressen angegeben werden.

Darüber hinaus gibt es noch weitere Konfigurationsanweisungen mit denen die Kommunikation mit dem AgentX Sub-Agenten detaillierter eingestellt werden kann.

```
# executables / AgentX
###########################
master agentx
#agentXSocket tcp:127.0.0.1:705
```

Listing 6.5.17: Konfiguration des AgentX Sub-Agenten in der Datei „snmpd.conf“

Auch der Net-SNMP Agent selbst kann als AgentX Sub-Agent gestartet werden, so dass er sich über das Erweiterungsprotokoll an einen Master SNMP-Agenten ankoppelt.

```
Agent # snmpd -X -x localhost:705 -c /etc/snmp/subxd.conf -Lf /var/log/subxd.log
```

Befehl 6.24: Starten des Net-SNMP Agenten als AgentX Sub-Agent

Der Net-SNMP AgentX Sub-Agent besitzt auch eigene Konfigurationsanweisungen die ebenfalls über die Manual-Page von „snmpd.conf“ bzw. über den Befehl „snmpd -H“ eingesehen werden können.

6.5.2 Die Anwendungsprogramme des Net-SNMP Agenten

Neben des zuvor vorgestellten Net-SNMP Agenten „snmpd“, beinhaltet das Softwarepaket Net-SNMP für den Agenten auch die Kommandozeilen-Programme „snmptrap“ und „snmpinform“, über die SNMP-Meldungen an den SNMP Trap-Server eines Managers gesendet werden können sowie das Kommandozeilen-Programm „net-snmp-config“, das die Konfiguration des Agenten und die Erstellung von AgentX Sub-Agenten erleichtert.

- **snmptrap, snmpinform:**
  Mit diesen beiden Kommandozeilen-Programmen ist es möglich Trap- und Inform-Meldungen an einen SNMP-Manager zu senden. Entsprechend den SNMP-Spezifikationen wird bei Trap-Meldungen nicht überprüft, ob diese auch beim Manager angekommen sind, im Gegensatz zu Inform-Meldungen. Das „snmpinform“ Net-SNMP Kommandozeilen-Programm ist vom „snmptrap“ Programm abgeleitet, weswegen mit dem Befehl „snmptrap -Ci“ ebenfalls eine Inform-Meldung gesendet wird.
  Die anzugebenden Befehlsoptionen unterscheiden sich bei den einzelnen SNMP-Versionen. Bei SNMPv1-Meldungen muss die OID des Objektes das die Meldung ausgelöst hat, die IP-Adresse des Agenten der dieses Objekt verwaltet, der Nachrichtentyp der Meldung und die Trap-Nummer sowie die Systemzeit angegeben werden. Werden Hochzeichen (" ") anstelle von Agenten-IP-Adresse, auslösendem Objekt und Systemzeit angegeben füllt das Programm die Werte automatisch aus. Bei SNMPv2- und Inform-Meldungen muss lediglich die Systemzeit und das auslösende Objekt angegeben werden. Beiden Meldungen können beliebige Managementinformationen angehängt werden, in ähnlicher Form wie bei den SET-Befehlen.
  Die folgenden beiden Beispiele senden eine Meldung, die den Start eines Agenten meldet. Die Meldung wird jeweils unter der Community „publicTrap“ an den Manager mit der IP-Adresse 10.0.0.1 gesendet. Bei SNMPv1-Meldungen wird das Objekt über den Zahlencode angegeben. Die erste 0 steht für eine Meldung vom Typ „coldStart“. Die zweite 0 für das erste Trap-Objekt der Meldungen diesen Typs.

```
Agent # snmptrap -v1 -c publicTrap 10.0.0.1 "" "" 0 0 " "
Agent # snmpinform -v2c -c publicTrap 10.0.0.1 ""  UCD-SNMP-MIB::ucdStart
```

Befehl 6.25: Versenden von SNMP-Meldungen

  Wie bei den Kommandozeilen-Programmen des Managers ist auch hier der Vorteil, dass die Programme in komplexere Scripte, wie Bash- oder Perl-Scripte, eingebunden werden können.

- **net-snmp-config:**
  Dieses Konfigurationsprogramm wurde zuvor schon im Zusammenhang mit der Einrichtung eines SNMPv3-Benutzers vorgestellt. Das Programm dient allgemein zur optionalen System-Einrichtung eines Net-SNMP Agenten. Es kann aber auch aus der Net-SNMP C-Implementierung eines Management-Objektes ein AgentX Sub-Agenten Programm kompilieren, welches sich dann standardmäßig über eine IPC-Schnittstelle an den Master Net-SNMP Agenten ankoppelt.

```
Agent # net-snmp-config --compile-subagent myagentx_name agentx.c
```

Befehl 6.26: Kompilieren eines AgentX Sub-Agenten für Net-SNMP

  Über die Eingabe „net-snmp-config –help“ werden alle Befehlsoptionen des Programms aufgelistet.

6.5.3 Erstellen und Einbinden eigener MIB-Module

Im vorherigen Kapitel 6.5.1 ging es bereits um die Konfiguration des Net-SNMP Agenten. Dabei wurde gezeigt wie über Anweisungen in der Konfigurationsdatei einige Managementinformationen für die Verwaltung via SNMP in den Agenten eingebunden werden können. Diese Möglichkeit der Einbindung von Managementinformationen ist zwar schnell durchgeführt und benötigt keine größeren Hintergrundkenntnisse, ist aber auch relativ stark auf die Vorgaben von Net-SNMP beschränkt.
Im folgenden wird an einem Beispiel gezeigt, wie Managementinformationen über die Net-SNMP C-API implementiert werden können. Über die C-API sind voll-funktionale Schnittstellen zu den Ressourcen des Agenten-Systems möglich. Mit den in Net-SNMP beiliegenden „Helfertools“ können Management-Objekte, deren SMI-Definition in einer MIB-Datei bereits vorliegen, auf recht einfachem Weg in C-Code implementiert werden und dann entweder in den Net-SNMP Agenten direkt oder als selbständiges Programm, das sich über das Erweiterungsprotokoll AgentX an den Net-SNMP Agenten ankoppelt, kompiliert werden.

Um Ressourcen des Agenten-Systems zu implementieren müssen die folgenden Dateien erstellt werden:

- eine MIB-Datei, die die SMI-Definition des Objektes der Ressource (Managementinformation) enthält
- eine C-Header-Datei (dateiname.h)
- eine C-Anwendungs-Datei (dateiname.c)

Zu den hier benötigten „Helfertools“ gehört das MIB-Parser-Programm „mib2c“. Dieses Programm generiert aus der vorhandenen SMI-Definition eines Management-Objektes eine C-Quellcode Vorlage, bestehend aus C-Funktionen der Net-SNMP C-API. Diese C-Funktionen realisieren die logische Schnittstelle zwischen dem Net-SNMP Agenten und den Management-Objekten. Lediglich die C-Funktionen, die die physikalischen Schnittstellen zwischen dem Net-SNMP Agenten und den Objekt-Ressourcen des verwalteten Systems realisieren, müssen individuell in diese C-Code Vorlage eingefügt werden, um einen vollständigen SNMP Zugriff zu ermöglichen.
Das Programm „mib2c“ generiert auf Basis der SMI-Objekt-Definitionen einer MIB-Datei eine C-Header-Datei und eine C-Anwendungs-Datei, deren Inhalt diese C-Quellcode Vorlage ist. Die erstellte C-Header-Datei muss nicht mehr verändert werden, wird aber aber später für den Kompiliervorgang benötigt. Die C-Anwendungs-Datei beinhaltet nach ihrer Erstellung bereits sämtliche notwendigen Funktionen der Net-SNMP C-API, um ein neues Management-Objekt einzubinden. Ihr fehlt allerdings der „funktionale“ C-Code, also der Teil des Quellcodes der den eigentlichen Zugriff auf die Objekt-Ressource des Agenten-Systems ermöglicht. Dieser Quellcode muss in die C-Anwendungs-Datei, an den entsprechenden Stellen, eingefügt werden.
Somit sind zwar Grundkenntnisse der Programmiersprache C notwendig, allerdings nur ein minimales Wissen über die Funktionen der Net-SNMP C-API.
Entsprechend der verschiedenen Arten von SNMP-Management-Objekten, enthält Net-SNMP auch verschiedene Stil-Vorlagen, die die C-Quellcode Vorlagen in den C-Dateien erstellen. So gibt es Stil-Vorlagen speziell für die Einbindung von Skalaren-Objekten, von Objekten in Tabellen und von Notification-Objekten. Bei diesen C-Stil-Vorlagen handelt sich um Dateien mit der Endung „conf“, die innerhalb des Net-SNMP Quelltext-Verzeichnisses unter „local/“ oder, nach der Installation, im Verzeichnis „/usr/local/share/snmp/“ zu finden sind. Darunter ist auch die C-Stil-Vorlage „Mfd“ (MIBs For Dummies), die eine einfache Implementierung von Tabellenstrukturen ermöglicht. Ein Beispiel für die Vorgehensweise bei der Implementierung von Tabellen ist in einem, sehr C-Quellcode bezogenen, „Tutorial“ auf der Net-SNMP Projekt-Internet-Seite zu finden. Das hier aufgeführte Beispiel besitzt der Einfachheit halber nur Skalare-Management-Objekte. Für das „mib2c“-Tool müssen die zu implementierenden Management-Objekte bereits vorher in die, von den Net-SNMP-Programmen lesbare, „Management Information Base“ eingegliedert sein. Dies wird am ehesten erreicht, indem die MIB-Datei, die die SMI-Definition der Management-Objekte enthält, im Standard-MIB-Verzeichnis abgelegt wird und die Umgebungsvariable „MIBS“ auf „ALL“ gesetzt wird.
Im nun folgenden Beispiel werden zwei neue Management-Objekte definiert. Die beiden Objekte werden unter der MODULE-IDENTITY „studienarbeit“ definiert und unterhalb des OBJECT-IDENTIFIER „enterprises“ in die Management Information Base eingegliedert. Das erste Objekt, mit Namen „superUserCommand“ (ID: 1), ist vom Typ „DisplayString“, also ein Objekt über eine beliebige Textzeile von maximal 255 Byte Länge. Über dieses Objekt sollen beliebige Linux-Konsolen-Befehle via SNMP zum Net-SNMP Agenten gesendet und dort ausgeführt werden können. Eine Absicherung darüber, ob es sich dabei um einen gültigen Linux-Befehl oder „Wortwirrwarr“ handelt, wird hier allerdings nicht implementiert.
Über das zweite Management-Objekt, mit Namen „ipv4forwarding“ (ID: 2), soll die IPv4-Weiterleitung eines Agenten-Systems und damit dessen Funktion als Durchgangs-Server, via SNMP von einem Manager aus, an- und abgeschaltet werden können. Ist die Funktion abgeschaltet so leitet das Agenten-System keine eingehenden Netzwerk-Nachrichten weiter, die an externe IPv4-Adressen gerichtet sind. Dieses Objekt ist eine Auswahlmöglichkeit über dem Datentyp „INTEGER“, wobei der Wert 0 mit der Auswahl „nonIpForwarding“, also IP-Weiterleitung aus, und der Wert 1 mit der Auswahlmöglichkeit „IpForwarding“, also IP-Weiterleitung

an, verknüpft ist. Die beiden Management-Objekte „superUserCommand“ und „IPv4Forwarding“ sind unterhalb des OBJECT-IDENTIFIER „server-mgmt“ (ID: 1) zusammengefasst, welches wiederum, zusammen mit dem OBJECT-IDENTIFIER „reserved“ (ID: 2), im Modul „studienarbeit“ (ID: 9876) zu finden ist. Dieses Modul ist mit der OID „iso.org.dod.internet.private.enterprises.studienarbeit“ (OID: .1.3.6.1.4.1.9876) in der „Management Information Base“ des Agenten eingegliedert.
Der Teilbaum der „Management Information Base“ sieht nach Eingliederung der Objekte wie folgt aus.

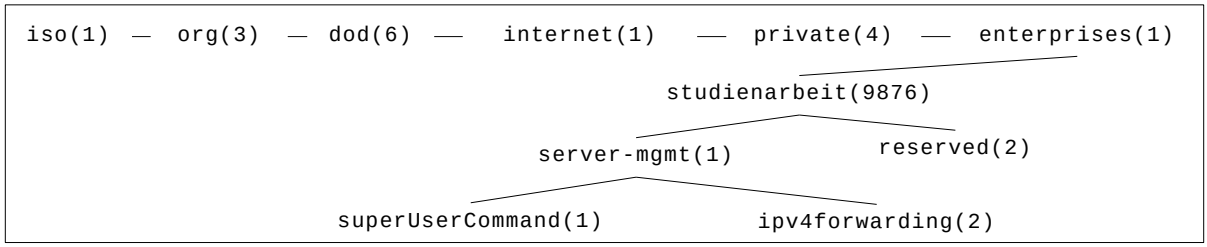


Abb. 6.2: MIB-Modul „studienarbeit“ im ISO-Registrierungsbaum

Die MIB-Datei, die die SMI-Definitionen dieser Objekte enthält und den obigen Management-Teilbaum „studienarbeit“ aufstellt, trägt den Namen „STUDIENARBEIT-MIB.txt“ und hat den folgenden Inhalt.

```
STUDIENARBEIT-MIB DEFINITIONS ::= BEGIN

IMPORTS
 enterprises, MODULE-IDENTITY, OBJECT-TYPE, INTEGER    FROM SNMPv2-SMI
 DisplayString                                         FROM SNMPv2-TC;

--
-- A brief description and update information about this mib.
--
studienarbeit MODULE-IDENTITY
 LAST-UPDATED "0411060000Z" -- 11 April 2006, midnight
 ORGANIZATION " - "
 CONTACT-INFO "
 Author: Frank Bohdanowicz, Studienarbeit: SNMP in VNUML Simulationen,
 AG Rechnernetze, University of Koblenz, 56070 Koblenz, Germany ,
 email: bohdan@uni-koblenz.de, phone: -
 "
 DESCRIPTION "Example MIB for remote control via SNMP"
::= { enterprises 9876 }

server-mgmt OBJECT IDENTIFIER::= { studienarbeit 1 }
reserved OBJECT IDENTIFIER::= { studienarbeit 2 }
--
-- a command of max 255 bytes length
--
superUserCommand OBJECT-TYPE
 SYNTAX DisplayString  (SIZE (0..255))
 MAX-ACCESS read-write
 STATUS current
 DESCRIPTION
 "A variable command, executed as root user"
::= { server-mgmt 1 }
--
-- an integer value 0 or 1 switches IP-Forwarding on/off
--
ipv4forwarding OBJECT-TYPE
 SYNTAX INTEGER {
       nonIpForwarding(0),  -- forwarding off
       ipForwarding(1)      -- forwarding on
       }
 MAX-ACCESS read-write
 STATUS current
 DESCRIPTION "Switches ipv4 Forwarding on/off"
::= { server-mgmt 2 }
END
```

Listing 6.9: Inhalt der MIB-Datei „STUDIENARBEIT-MIB.txt“

Wie bereits in Kapitel 4.3.3.2 erwähnt, beginnt eine solche MIB-Datei mit dem Namen der Datei und dem Schlüsselwort „BEGIN“. Anschließend folgt das Importieren der verwendeten Datentypen. Hier im Beispiel wird nun das MODUL „studienarbeit“ mit den dazugehörigen organisatorischen Details definiert. Darauf folgen die beiden OBJECT-IDENTIFIER „server-mgmt“ und „reserved“, die keine eigene Managementinformation tragen und im Registrierungsbaum als interne Knoten dargestellt sind. Zu Guter Letzt folgen die beiden eigentlichen Informationsträger, die Management-Objekte „superUserCommand“ und „ipv4forwarding“, mit der Definition ihrer zugrundeliegenden Syntax und der Zugriffsrechte, dem Status der Gültigkeit der Implementierung, einer Erläuterung der Funktion und schließlich der OID, unterhalb der es im Registrierungsbaum, bzw. der MIB, zu finden ist. Mit dem Schlüsselwort „END“ wird das Ende der MIB-Datei markiert.
Die MIB-Datei „STUDIENARBEIT-MIB.txt“ muss nun ins Standard-MIB-Verzeichnis von Net-SNMP kopiert werden. Anschließend müssen die neuen Objekte in die „Management Information Base“ der Net-SNMP Programme eingegliedert werden, was am Besten über das Setzen der Umgebungsvariable MIBS auf den Wert „ALL“ erreicht wird, womit alle im Standard-MIB-Verzeichnis enthaltenen Objekte eingegliedert werden.
Während der praktischen Ausarbeitung der Studienarbeit ergab sich, dass das „mib2c“ Parser Programm die MIB-Datei auch im Verzeichnis „/usr/share/snmp/mibs/“ sehen möchte.

Folglich sind also folgende Schritte zu erledigen.

```
Agent # cp STUDIENARBEIT-MIB.txt /usr/local/share/snmp/mibs/
Agent # cp STUDIENARBEIT-MIB.txt /usr/share/snmp/mibs/
Agent # export MIBS=“ALL“
```

Befehl 6.27: Eingliederung der „STUDIENARBEIT-MIB“ in die Net-SNMP MIB-Umgebung

Nun sollten die neuen Management-Objekte über den Befehl „snmptranslate“ einzusehen sein.

```
Agent # snmptranslate -Tp -IR studienarbeit
+--studienarbeit(9876)
  |
  +--server-mgmt(1)
  | |
  | +-- -RW- String   superUserCommand(1)
  | |       Textual Convention: DisplayString
  | |       Size: 0..255
  | +-- -RW- EnumVal  ipv4forwarding(2)
  |         Values: nonIpForwarding(0), ipForwarding(1)
  |
  +--reserved(2)
```

Befehl 6.28: Anzeigen der neuen Objekte mit „snmptranslate“

Ist dies der Fall, so kann nun die C-Anwendungs-Datei und die C-Header-Datei erstellt werden.
Da innerhalb der „STUDIENARBEIT-MIB“ lediglich Skalare-Objekte definiert sind, kann als Stil-Vorlage die Datei „mib2c.scalar.conf“ zum Erstellen der C-Quellcode-Vorlage verwendet werden.

Die beiden benötigten C-Dateien können dann wie folgt, in einem beliebigen Verzeichnis, erstellt werden.

```
Agent # mib2c -c /usr/local/share/snmp/mib2c.scalar.conf studienarbeit
writing to studienarbeit.h
writing to studienarbeit.c
running indent on studienarbeit.c
running indent on studienarbeit.h
```

Befehl 6.29: Erstellen der C-Dateien auf Basis der STUDIENARBEIT-MIB

Wie an der Ausgabe zu erkennen ist werden hier die beiden C-Dateien erzeugt und innerhalb des Verzeichnisses abgelegt. Der Befehl „mib2c studienarbeit“, ohne Angabe einer Stil-Vorlage, erstellt die C-Dateien nicht sofort wie hier, sondern öffnet einen Frage/Antwort-Dialog, über den die passende Stil-Vorlage gewählt werden kann.

Die erstellte C-Header-Datei „studienarbeit.h“ kann so wie sie ist belassen werden.
Die C-Anwendungs-Datei „studienarbeit.c“ beinhaltet nun bereits alle für die SNMP Kommunikation notwendigen C-Funktionen der Net-SNMP C-API und muss nun noch um die C-Funktionen, die auf die Objekt-Ressource des Agenten-Systems zugreifen, erweitert werden.

Im folgenden wird nun Stück für Stück die fertige C-Anwendungs-Datei erläutert. Die Code-Stücke die zusätzlich eingefügt wurden sind dabei in Fett-Schrift markiert, alle anderen Code-Stücke sind zuvor automatisch vom „mib2c-Parser“ eingefügt worden.

Zu Beginn sind die inkludierten Header-Dateien aufgeführt, denen für dieses Beispiel hier noch die Header Datei „stdio.h“, für Ein- und Ausgabefunktionen, fehlt. Außerdem sind hier noch die benötigten globalen Variablen für das Zwischenspeichern der Objekt-Werte definiert. Dazu gehört das Char-Array „superUserCommand“ für den Befehlstext, die Integer Variable „ipForwarding“ für den Weiterleitungswert und der C-Pointer „*stream“, der das Lesen und Schreiben von Dateien auf dem Agenten-System ermöglicht.

```
#include <net-snmp/net-snmp-config.h>
#include <net-snmp/net-snmp-includes.h>
#include <net-snmp/agent/net-snmp-agent-includes.h>
#include "studienarbeit.h"

#include <stdio.h>

static char superUserCommand[] = "";
static int ipForwarding = 0;
FILE *stream;
```

Listing 6.10.1: Inhalt der Datei „studienarbeit.c“, Teil 1

Als nächstes folgt mit der Initialisierung schon der Kern der C-Anwendungsdatei. In dieser „init“-Funktion werden sämtliche Management-Objekte beim Net-SNMP Agenten registriert.
Über die Funktion „netsnmp_register_scalar“ wird ein Skalar-Objekt eingebunden. Angegeben werden muss dabei, der Name des Objektes, die C-Funktion die bei Abruf aufgerufen werden soll, die OID des Objektes, die Bytelänge der OID und die Zugriffsrechte, die das Lesen oder Schreiben des Objektes festlegen.

```
void init_studienarbeit(void)
{
    static oid superUserCommand_oid[] = { 1, 3, 6, 1, 4, 1, 9876, 1, 1 };
    static oid ipv4forwarding_oid[] = { 1, 3, 6, 1, 4, 1, 9876, 1, 2 };

    netsnmp_register_scalar(netsnmp_create_handler_registration
                            ("superUserCommand", handle_superUserCommand,
                             superUserCommand_oid,
                             OID_LENGTH(superUserCommand_oid),
                             HANDLER_CAN_RWRITE));

    netsnmp_register_scalar(netsnmp_create_handler_registration
                            ("ipv4forwarding", handle_ipv4forwarding,
                             ipv4forwarding_oid, OID_LENGTH(ipv4forwarding_oid),
                             HANDLER_CAN_RWRITE));
}
```

Listing 6.10.2: Inhalt der Datei „studienarbeit.c“, Teil 2

Hier ist nichts mit Fett-Schrift markiert da das Programm „mib2c“ die komplette Funktion selbst erstellt. Allerdings ist darauf zu achten, dass das Programm die richtige OID einsetzt. Da die Objekte aus der globalen Net-SNMP „Management Information Base“ gelesen werden, kann es bei gleichen textuellen Objekt-Bezeichnungen zu Verwechslungen kommen.
Anschließend folgt auch schon die Definition der Funktionen, die bei Abruf des Objektes aufgerufen werden sollen.

Hier folgt der Kopf der ersten Funktion, die für das Objekt „superUserCommand“ den Zugriff implementiert.

```
int
handle_superUserCommand(netsnmp_mib_handler *handler,
                        netsnmp_handler_registration *reginfo,
                        netsnmp_agent_request_info *reqinfo,
                        netsnmp_request_info *requests)
{
```

Listing 6.10.3: Inhalt der Datei „studienarbeit.c“, Teil 3

Innerhalb dieser Funktion ist eine Switch-Case Anweisung implementiert. Diese Switch-Case Anweisung besteht aus insgesamt acht Fällen, die den „Get“- und „Set“-Zugriff verarbeiten.
Der erste Fall der Switch-Case Anweisung, „MODE_GET“, verarbeitet alle eingehende Get-Befehle, die das Objekt ansprechen. Hier muss C-Code eingefügt werden, der die Managementinformation des Objektes aus dem Agenten-System ausliest.

Die Verarbeitung des SNMP-Set-Befehls kann komplexer gestaltet werden. Da hier schreibend auf das Agenten-System zugegriffen wird, gibt es die Möglichkeit, über verschiedene Sicherheits-Stufen festzustellen, ob bestimmte Voraussetzungen erfüllt sind, die ein sicheres Schreiben der Managementinformation in das Agenten-System erlauben, ohne dessen Stabilität zu gefährden. Diese Abstufungen sind über die nächsten sechs Fälle der Switch-Case Anweisung realisiert. Wie in der Abbildung 6.3 dargestellt, teilt sich die Verarbeitung eines SNMP-Set-Befehls auf diese sechs Fälle auf. Sind alle Voraussetzungen zum schreiben der Managementinformation erfüllt, so werden die vier Fälle auf der linken Seite des Bildes durchlaufen. Treten im Vorfeld des Schreibens Probleme auf, so können diese über den Fall „MOD_SET_FREE“ behoben werden. Treten während des Schreibens Fehler auf, so kann die Set-Transaktion komplett unterbrochen und rückgängig gemacht. Dieses Vorgehen bei SNMP-Set-Befehlen gewährleistet dem Net-SNMP Agenten eine höhere Sicherheit und Kontinuität.

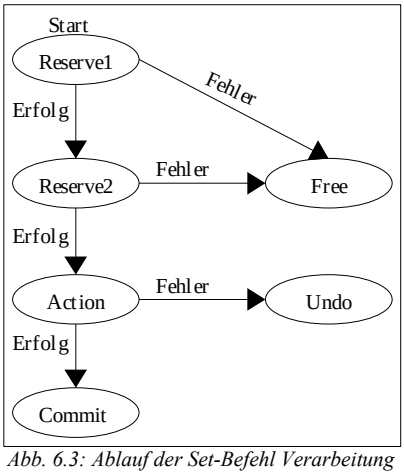


Abb. 6.3: Ablauf der Set-Befehl Verarbeitung

Jedoch müssen nicht alle Fälle bearbeitet werden. Für die Verarbeitung von eingehenden SNMP-Set-Befehlen genügt der Fall „MODE_SET_ACTION“.
Als letzten Fall der Switch-Case Anweisung, gibt es noch „DEFAULT“. Dieser Fall gilt aber als tiefergehender Fehlerzustand des Net-SNMP Agenten und sollte nach Möglichkeit nie erreicht werden.

Das erste Objekt, das in dieser Funktion implementiert wird, soll Linux-Konsolenbefehle empfangen und auf dem Agenten ausführen können.
Mit den Get-Befehlen wird der zuletzt gesendete Konsolenbefehl ausgelesen und an den Manager gesendet.
Mit dem SNMP-Set-Befehl wird ein neuer Linux-Konsolenbefehl vom Manager aus an den Agenten übertragen und dort von diesem in dessen System ausgeführt.

```
    switch (reqinfo->mode) {

     case MODE_GET:
         snmp_set_var_typed_value(requests->requestvb, ASN_OCTET_STR,
                                  (char *) &superUserCommand,
                                   sizeof(superUserCommand) );
         break;

     case MODE_SET_RESERVE1:
         break;
     case MODE_SET_RESERVE2:
         break;
     case MODE_SET_FREE:
         break;

     case MODE_SET_ACTION:
            sprintf(superUserCommand,"%s",(requests->requestvb->val.string));
            system(superUserCommand);
         break;

     case MODE_SET_COMMIT:
         break;
     case MODE_SET_UNDO:
         break;

     default:
         snmp_log(LOG_ERR,
                  "unknown mode (%d) in handle_superUserCommand\n",
                   reqinfo->mode);
         return SNMP_ERR_GENERR;
     }

     return SNMP_ERR_NOERROR;
}
```

Listing 6.10.4: Inhalt der Datei „studienarbeit.c“, Teil 4

Der obige Code Ausschnitt zeigt den Rest der Methode „handle_superUserCommand“. In der Switch-Case Anweisung werden hier im Beispiel nur die beiden Fälle „MODE_GET“ und „MODE_SET_ACTION“, die für das Lesen bzw. Schreiben des Objektes „superUserCommand“ zuständig sind, verwendet. Alle anderen Fälle müssen aber trotzdem für die einwandfreie Funktion zumindest aufgelistet sein.

Der „MODE_GET“ Fall beinhaltet hier nur eine C-Funktion, die bereits weitestgehend automatisch erstellt wurde. Nur die Char-Array Variable „superUserCommand“, welche die Managementinformation des gleichnamigen Objektes beinhaltet sowie die Länge der Variable müssen eingetragen werden.
Diese diese beiden Parameter und die Datentypangabe „ASN_OCTET_STR“ entsprechen auch den benötigten Werten für die Kodierungsregeln BER, dargestellt in Abbildung 4.9.

Der C-Code des Falles „MODE_SET_ACTION“ wurde komplett eingefügt. Die C-Funktion „sprintf“ kopiert den, über einen SNMP-Set-Befehl empfangenen, neuen Objekt-Wert in das Char-Array „superUserCommand“. Anschließend wird über die C-Funktion „system“ der Objekt-Wert, der ein gültiger Linux-Konsolenbefehl sein sollte, auf dem Agenten-System ausgeführt. Die hier nicht verwendeten Fälle der Set-Verarbeitung in der Switch-Case Anweisung könnten zum Beispiel zur Überprüfung verwendet werden, ob es sich auch um einen gültigen Linux-Konsolenbefehl handelt oder die hier gültigen Konsolenbefehle auf eine kleine Auswahl beschränken, um das hier nun entstehende gewaltige Sicherheitsloch wieder etwas zu entschärfen.
Direkt im Anschluss folgt dann auch schon die Funktion, die das nächste Objekt „ipv4forwarding“ verwaltet. Das Grundgerüst dieser Funktion ist gleich mit dem der Funktion zuvor.
In der Funktion „handle_ipv4forwarding“ wird im Fall „MODE_GET“ der Switch-Case Anweisung der Inhalt der Datei „ip_forward“, die sich im Verzeichnis „/proc/sys/net/ipv4/“ befindet, ausgelesen, in die Integer Variable ipForwarding geschrieben und dann zum senden an die Net-SNMP-Agenten Funktion „snmp_set_var_typed_value“ übergeben. Die Datei „ip_forward“ enthält normalerweise den Wert 0 oder 1, was jeweils angibt, ob die Weiterleitung von Nachrichten an IP-Adressen anderer Rechner aus- oder eingeschaltet ist. Es wird also der aktuelle Zustand direkt aus dem Agenten-System gelesen.
Im Fall MODE_SET_ACTION wird der empfangene Objekt-Wert in die Variable „ipForwarding“ geschrieben, anschließend zusammen mit dem Linux Befehl „sysctl“ über die C-Funktion „sprintf“ in der Char-Array Variable „sysforward“ verknüpft und dann über die C-Funktion „system“ auf dem Agenten-System ausgeführt.

```
int
handle_ipv4forwarding(netsnmp_mib_handler *handler,
                  netsnmp_handler_registration *reginfo,
                  netsnmp_agent_request_info *reqinfo,
                  netsnmp_request_info *requests)
{
    switch (reqinfo->mode) {

    case MODE_GET:
        stream = fopen("/proc/sys/net/ipv4/ip_forward","r");
        if(stream != NULL){ fscanf(stream,"%d",&ipForwarding); }
        fclose(stream);

        snmp_set_var_typed_value(requests->requestvb, ASN_INTEGER,
                                 (u_char *) &ipForwarding,
                                  sizeof(ipForwarding) );
        break;
    case MODE_SET_RESERVE1:
        break;
    case MODE_SET_RESERVE2:
        break;
    case MODE_SET_FREE:
        break;

    case MODE_SET_ACTION:
        ipForwarding = *(requests->requestvb->val.integer);
        char sysforward[]={ "sysctl net.ipv4.ip_forward=" };
        sprintf(sysforward,"%s%d",sysforward,ipForwarding);
        system(sysforward);
        break;

    case MODE_SET_COMMIT:
        break;
    case MODE_SET_UNDO:
        break;
    default:
        snmp_log(LOG_ERR, "unknown mode (%d) in handle_ipv4forwarding\n",
              reqinfo->mode);
        return SNMP_ERR_GENERR;
    }
    return SNMP_ERR_NOERROR;
}
```

Listing 6.10.5: Inhalt der Datei „studienarbeit.c“, Teil 5

Damit wäre die C-Anwendungs-Datei „studienarbeit.c“ fertig und muss nun nur noch kompiliert werden.

Zwei Möglichkeiten die Dateien zu kompilieren und die Management-Objekte in einen Net-SNMP Agenten einzubinden werden hier nun aufgezeigt.

Die erste Möglichkeit ist die direkte Implementierung in den Net-SNMP Agenten, was während des Kompilierens vor der Installation geschehen muss. Die C-Header Datei „studienarbeit.h“ und die C-Anwendung-Datei „studienarbeit.c“ müssen in das Verzeichnis „agent/mibgroup/“, das sich innerhalb des Net-SNMP Quelltext Verzeichnisses befindet, kopiert werden.

```
Agent # cp studienarbeit.* /'pathToSNMPsourceDirectory'/agent/mibgroup/
Agent # cd /'pathToSNMPsourceDirectory'/
Agent # ./configure ... --with-mib-modules=“smux host ... studienarbeit“ ...
Agent # make && make install
```

Befehl 6.30: Einbinden der „STUDIENARBEIT-Objekte“ direkt in den Net-SNMP Agenten

Über die Konfigurationsoption „--with-mib-modules="studienarbeit"“ werden die beiden C-Dateien zusammen mit dem Net-SNMP Agenten kompiliert und installiert. Die beiden Management-Objekte sind dann voll in den Net-SNMP Agenten integriert und jederzeit ansteuerbar.

Die zweite Möglichkeit ist das Kompilieren zu einem eigenständigen AgentX Sub-Agentenprogramm, mittels dem bereits vorgestellten Tool „net-snmp-config“. Der Vorteil hierbei ist, das die Management-Objekte nur dann von einem Manager aus ansteuerbar sind, wenn dieses Programm läuft. Außerdem muss nicht das komplette Net-SNMP Softwarepaket neu kompiliert und installiert werden. Laut Net-SNMP-Entwicklergruppe ist diese Methode allerdings noch nicht ganz ausgereift und es kann zu Problemen kommen. Während der praktischen Ausarbeitung der Studienarbeit traten jedoch keinerlei Probleme auf, was aber daran liegen kann, dass nur einfache SMI-Konstrukte verwendet wurden.
Das Programm muss in dem Verzeichnis ausgeführt werden, indem sich die C-Anwendungs- und die C-Header-Datei befinden. Der folgende Befehl kompiliert aus den zuvor erstellten Dateien das Programm mit Namen „studienProgie“, das anschließend im gleichen Verzeichnis zu finden ist.

```
Agent # net-snmp-config --compile-subagent studienProgie studienarbeit.c
generating the tmporary code file: netsnmptmp.25262.c
[ . . . ]
subagent program studienProgie created
```

Befehl 6.31: Kompilieren der „STUDIENARBEIT-Objekte“ in einen AgentX Sub-Agenten

Das Programm „studienProgie“ kommuniziert über das Erweiterungsprotokoll AgentX mit einem Master Net-SNMP-Agenten. Dort muss natürlich zuvor AgentX über die Konfigurationsdatei aktiviert worden sein (Anweisung: master agentx). Um Zugriff via SNMP auf die vom Programm „studienProgie“ verwalteten Management-Objekte „superUserCommand“ und „ipv4forwarding“ zu erhalten, muss es gestartet werden. Mit dem Beenden des Programms endet auch die Möglichkeit des Zugriffs. Gestartet und gestoppt wird das Programm wie folgt. Eventuell muss es zuvor in das Verzeichnis „/var/agentx/“ abgelegt werden.

```
Agent # ./studienProgie &
[1] 25502

Agent # killall studienProgie
[1]+  Terminated              ./studienProgie
```

Befehl 6.32: Starten des Sub-Agenten

Solange das Programm läuft können die beiden Management-Objekte verwendet werden.

Das hier definierte Management-Objekt „superUserCommand“ entspricht eigentlich nicht der SNMP-Philosophie, da es nicht den Status einer bestimmten Ressource des Agenten-Systems beschreibt und keine eindeutige Funktionalität besitzt. Über dieses Objekt können beliebige Linux Befehle an den Agenten gesendet werden, die dieser lokal ausführt.

Zum Beispiel können so nun neue Routen-Konfigurationen, über den Befehl „route“, vom Manager aus auf dem Agenten-System eingestellt werden.
Der folgende Befehl fügt einen Eintrag in die Kernel Routing Tabelle des Agenten-Systems hinzu, der den Agenten das Netz 192.168.0.0/30 mit einer Metric von fünf über die Netzwerkschnittstelle eth1 erreichen lässt.

```
Manager # snmpset -v1 -c private 10.0.0.2 studienarbeit.1.1.0 s "route add -net 192.168.0.0/30 metric 5 eth1"
STUDIENARBEIT-MIB::superUserCommand.0 = STRING: route add -net 192.168.0.0/30 metric 5 eth1
```

Befehl 6.33: Verwendungsbeispiel des Objektes „superUserCommand“ der „STUDIENARBEIT-MIB“

Der Agent besitzt nach dem Senden des Befehls diese Route was mit dem Befehl „route -n“ über die Kernel-Tabelle festgestellt werden kann.

```
Agent # route -n
Kernel IP Routentabelle
Ziel            Router      Genmask          Flags  Metric  Ref  Use  Iface
192.168.0.4     0.0.0.0     255.255.255.252  U      0       0    0    eth0
10.0.0.0        0.0.0.0     255.255.255.0    U      0       0    0    eth1
192.168.0.0     0.0.0.0     255.255.255.252  U      5       0    0    eth1
127.0.0.0       0.0.0.0     255.0.0.0        U      0       0    0    lo
```

Befehl 6.34: Überprüfen der Netzwerk-Routen mit dem Befehl „route“

Auch neue IP-Routen können mit dem entsprechenden Befehl „ip route“ an den Agenten übergeben und dort konfiguriert werden.

```
Manager # snmpset -v1 -c private 10.0.0.2 studienarbeit.1.1.0 s "ip route add 192.168.0.0/30 via 10.0.0.1"
STUDIENARBEIT-MIB::superUserCommand.0 = STRING: ip route add 192.168.0.0/30 via 10.0.0.1
```

Befehl 6.35: Übergeben einer neuen IP-Route via SNMP über das Objekt „superUserCommand“

Das Management-Objekt „superUserCommand“ beinhaltet hier allerdings nicht die Ausgabe des übergebenen Linux Konsolenbefehls, sondern lediglich den zuletzt übergebenen Befehl selbst, weswegen ein Fehler bei der Ausführung des Befehls auf dem Agenten beim Manager nicht bemerkt wird. Allerdings kann über andere, bereits eingebundene Management-Objekte durchaus das Resultat begutachtet werden, in diesem Beispiel der Übergabe einer neuen IP-Route, über das Management-Objekt „ipTable“ der IP-Gruppe aus der MIB-II. Aufgrund des von Net-SNMP verwendeten Cache können diese Veränderungen stellenweise allerdings erst mit einiger Verzögerung überprüft werden.

```
Manager # snmptable -v1 -c public -Cw 110  10.0.0.2  ip.ipRouteTable
SNMP table: RFC1213-MIB::ipRouteTable

 ipRouteDest  ipRouteIfIndex  ipRouteMetric1 ipRouteMetric2 ipRouteMetric3 ipRouteMetric4 ipRouteNextHop
  10.0.0.0            1              0              ?              ?              ?          0.0.0.0
  127.0.0.0           3              0              ?              ?              ?          0.0.0.0
  192.168.0.0         1              1              ?              ?              ?          10.0.0.1
  192.168.0.4         2              0              ?              ?              ?          0.0.0.0

SNMP table RFC1213-MIB::ipRouteTable, part 2

 ipRouteType  ipRouteProto  ipRouteAge  ipRouteMask      ipRouteMetric5  ipRouteInfo
  direct          local          ?      255.255.255.0          ?       SNMPv2-SMI::zeroDotZero
  direct          local          ?      255.0.0.0              ?       SNMPv2-SMI::zeroDotZero
  indirect        local          ?      255.255.255.252        ?       SNMPv2-SMI::zeroDotZero
  direct          local          ?      255.255.255.252        ?       SNMPv2-SMI::zeroDotZero
```

Befehl 6.36: Auslesen der Routing-Tabelle des Agenten via SNMP

Als weiteres Beispiel für das Ausführen eines Konfigurations-Programms auf dem Agenten wird hier nun noch „sysctl“ aufgeführt. Mit diesem Programm wird auch beim neu erstellten Objekt „ipv4forwarding“ gearbeitet. Mit dem Befehl „sysctl“ lassen sich Linux-Kernel Einstellungen zur Laufzeit konfigurieren. Diese Einstellungen sind im Verzeichnisses „/proc/sys/“ zu finden. Mit dem Befehl „sysctl -a“ wird eine Liste aller möglichen Einstellungen aufgelistet. Zum Beispiel befindet sich die „StandardTimeToLive“ Einstellung für IP-Pakete im Verzeichnis „/proc/sys/net/ipv4/“ in der dortigen Datei „ip_default_ttl“. Die Einstellung kann wie folgt mit „sysctl“ konfiguriert werden.

```
Agent # sysctl net.ipv4.ip_default_ttl
net.ipv4.ip_default_ttl = 64
Agent # sysctl net.ipv4.ip_default_ttl=16
net.ipv4.ip_default_ttl = 16
```

Befehl 6.37: Konfiguration der Netzwerkschnittstellen mit dem Befehl „sysctl“

Als Eingabe folgt dem Befehl die Datei mit dem zu konfigurierenden Wert, wobei die einzelnen Unterverzeichnisse bis zur Datei durch Punkte getrennt werden müssen.

Statt den Befehl direkt in der Konsole des Agenten einzugeben kann er nun auch via SNMP über das Objekt „superUserCommand“ vom Manager aus an den Agenten gesendet und dort ausgeführt werden.

```
Manager # snmpset -v1 -c private 10.0.0.2 studienarbeit.1.1.0 s "sysctl net.ipv4.ip_default_ttl=16"
STUDIENARBEIT-MIB::superUserCommand.0 = STRING: sysctl net.ipv4.ip_default_ttl=16
```

Befehl 6.38: Übermittlung des „sysctl“ Befehls an den Agenten via SNMP

Diese Möglichkeit der Konfiguration des Agenten wurde während der praktischen Ausarbeitung der Studienarbeit erarbeitet da für einen Großteil der Objekte, deren Managementinformationen Net-SNMP automatisch nach der Installation zur Verwaltung via SNMP einrichtet, kein Schreibzugriff hergestellt werden konnte, sondern lediglich Lesezugriff erlaubt war. Nur die Informationen der Objekte der Gruppe „system“ und der „UCD“-MIB-Module konnten verändert werden. Das Management-Objekt „superUserCommand“ zeigt damit in etwa die Mächtigkeit, die SNMP ebenfalls im Konfigurationsmanagement haben kann.

Allerdings wäre es eher im Sinne von SNMP nicht den kompletten Linux-Befehl auf dem Manager einzugeben und an den Agenten zu übertragen, wie das beim Objekt „superUserCommand“ passiert, sondern lediglich die zu verändernde Objekt-Variable, die dann auf dem Agenten mit dem entsprechenden Linux-Befehl in das System übernommen wird, wie dies beim zweiten Objekt „ipv4forwarding“ geschieht. Das zweite, neu erstellte Management-Objekt „ipv4forwarding“ ist eher im Sinne von SNMP, da hier lediglich der Wert des Objektes, das einen fest definierten und klar abgrenzbaren Aufgabenbereich besitzt, verändert werden kann.

```
Manager # snmpset -v1 -c private 10.0.0.2 studienarbeit.1.2.0 i ipForwarding
STUDIENARBEIT-MIB::ipv4forwarding.0 = INTEGER: ipForwarding(1)
```

Befehl 6.39: Abfrage IPv4-Weiterleitung über das „ipv4forwarding“ Objekt

Das dargestellte Abfrage-Beispiel ist so nur ausführbar, wenn der Manager auch Zugriff auf die MIB-Datei „STUDIENARBEIT-MIB.txt“ hat. Über die dort enthaltene SMI-Definition des Objektes wird auch der Begriff „ipForwarding“ in den entsprechende Integerwert, in diesem Fall 1, abgebildet. Kann der Manager nicht auf die MIB-Datei zugreifen, so ist ein Zugriff nur über die Eingabe der nummerischen Werte möglich.

```
Manager # snmpset -v1 -c private 10.0.0.2  .1.3.6.1.4.1.9876.1.2.0  i  0
iso.3.6.1.4.1.9876.1.2.0 = INTEGER: 0
```

Befehl 6.40: Setzen der IPv4-Weiterleitung via SNMP über die nummerischen Werte

Hier wird auch das Problem von SNMP offenbar, dass für eine komfortable und fehlerfreie Anwendung die MIB-Dateien auf dem Manager nicht nur verfügbar, sondern auch in der gleichen Version wie deren Implementierung auf dem Agenten vorhanden sein müssen, wenn der Manager die nummerische OID des Objektes nicht kennen soll oder kann.

6.6 Anwendungsbeispiele

Eine Stärke von SNMP ist das Verarbeiten von Managementinformationen im Sinne des Leistungsmanagements. Net-SNMP bindet standardmäßig einen Großteil von Informationen des Agenten-Systems automatisch bei der Installation ein und stellt sie einem SNMP-Manager zum Auslesen zur Verfügung. Der Vorteil der mit SNMP verwalteten Managementinformation ist, dass diese in einer Form vorliegt, die eine einfache Weiterverarbeitung erlaubt. Auch der Datentyp der Managementinformation ist den beteiligten Prozessen bekannt. Über die SMI-Definition des Objektes sind noch weitere Informationen über die Managementinformation vorhanden. So ist es für externe Programme einfach, SNMP-Managementinformationen für grafische Darstellungen oder statistische Berechnungen auszuwerten und zu verwenden.

Das folgende Bash-Programm, mit dem Namen „ifTraffic.sh“, liest im Sekundentakt den Datenverkehr einer Netzwerkschnittstelle von einem Agenten aus. Die ausgelesenen Daten werden in eine CSV Datei (Comma Separated Values) gespeichert. Das Programm bekommt die IP-Adresse des Agenten, einen Community-Namen, die Bezeichnung der Netzwerkschnittstelle und einen Dateinamen, zum speichern der Informationen, beim Start übergeben. Es beginnt damit den Namen des Agenten sowie die Anzahl seiner Netzwerkschnittstellen via SNMP auszulesen. Die Anzahl der Netzwerkschnittstellen abzufragen ist notwendig, um über die angegebene Netzwerkschnittstellenbezeichnung die zugehörige Indexnummer aus der Interface-Tabelle zu erhalten. Über eine While-Schleife werden dann im Sekundentakt die Datenverkehrsinformationen der Netzwerkschnittstelle vom Agenten abgefragt und in die CSV-Datei gespeichert.

```
#!/bin/bash

AGENT_IP=$1
COM=$2
IF_NAME=$3
FILE=$4

DATA=`snmpgetnext -Oqv -v2c -c$COM $AGENT_IP IF-MIB::ifNumber SNMPv2-MIB::sysName`
set -- $DATA
IF_NUM=$1
SYS_NAME=$2

BULK_REQ=`snmpbulkget -Oqv -v2c -c$COM -Cn0 -Cr$IF_NUM $AGENT_IP IF-MIB::ifDescr`
echo "Name: $SYS_NAME ,  Interface: $IF_NAME : $IF_NUMBER" >> $FILE

for IF_DESCR in $BULK_REQ
do
  count=$((count+1))
  if [ "$IF_NAME" == "$IF_DESCR" ]
  then
    NUM=$count
    break
  fi
done

TABLE_HEAD="ifInOctets , ifOutOctets , ifInUcastPkts , ifOutUcastPkts"
echo  "$TABLE_HEAD" >> $FILE

TRUE=1
count=0

REQ_MIB="IF-MIB::ifInOctets.$NUM IF-MIB::ifOutOctets.$NUM"
REQ_MIB="$REQ_MIB IF-MIB::ifInUcastPkts.$NUM IF-MIB::ifOutUcastPkts.$NUM"

while [ "$TRUE" ]
do
  count=$((count+1))
  for IFOCTS in `snmpget -Oqv -v1 -c $COM $AGENT_IP $REQ_MIB`
  do
    echo -n " $IFOCTS , " >> $FILE
  done
  sleep 1
  echo >> "$FILE"
done

exit 0
```

Listing 6.11: Inhalt des Programm „ifTraffic.sh“

Das Bash-Programm wird dann beispielsweise wie folgt gestartet.

```
Manager # ./ifTraffic.sh 10.0.0.2 public eth1 ifTrafficAgent_eth1.csv
```

Befehl 6.41: Start des Bash-Programms „ifTraffic.sh“

In der angegebenen Datei „ifTrafficAgent_eth1.csv“ werden dann Informationen über die ein- und ausgehenden Bytes und Datenpakete der Netzwerkschnittstelle „eth1“ des Agenten gesammelt.
Nach einiger Zeit beinhaltet die Datei Daten der folgenden Art.

```
Name: Agent , Interface: eth1
ifInOctets , ifOutOctets , ifInUcastPkts , ifOutUcastPkts
 29458 ,  29458 ,  249 ,  249 ,
 29848 ,  29848 ,  251 ,  251 ,
 30238 ,  30238 ,  253 ,  253 ,
 30628 ,  30628 ,  255 ,  255 ,
 31018 ,  31018 ,  257 ,  257 ,
 31408 ,  31408 ,  259 ,  259 ,
 31798 ,  31798 ,  261 ,  261 ,
```

Listing 6.12: Inhalt der Datei „ifTrafficAgent_eth1.csv“

Eine solche CSV Datei kann zum Beispiel mit Tabellenkalkulationsprogrammen, auch grafisch, weiterverarbeitet und ausgewertet werden.

Zahlreiche professionelle Programme bauen in ähnlicher Form wie dieses Bash-Programm auf den Kommandozeilen-Programmen des Net-SNMP Managers auf. Programme wie MRTG (Multi Router Traffic Grapher) zeichnen direkt eine grafische Leistungskurve aus den empfangenen Informationen der Überwachten Management-Objekte, sofern die Objekt-Werte nummerisch sind. Nachteil vieler professioneller Programme für den Manager ist allerdings, dass sie für größere Abfrageintervalle von ca. fünf Minuten ausgelegt sind um die Netzwerke nicht allein durch Management-Abfragen über Gebühr zu belasten. Für eine Simulations-Umgebung wie VNUML wäre dieses Leistungsmanagement aber wohl in vielen Fällen nicht detailreich genug.

Die zweite Stärke von SNMP ist die Fähigkeit des SNMP-Agenten auf eintretende Ereignisse selbständig zu reagieren und dem Manager eine Meldung zukommen zulassen. Bereits in Net-SNMP eingebundene Management-Objekte lassen sich einfach über die Konfigurationsdatei überwachen, wie im Kapitel 6.5.1.3 dargestellt. Zusätzlich bietet Net-SNMP mit dem Kommandozeilen-Programm „snmptrap“ eine etwas flexiblere Möglichkeit über die Einbettung in ein beliebiges Script-Programm.

Das folgende Bash-Programm mit Namen „zebraTest.sh“ überprüft in einer While-Schleife alle 30 Sekunden, ob ein Prozess „zebra“ im Agenten-System läuft. Ist dies nicht der Fall so wird eine SNMPv2-Trap unter dem Community-Namen „trapatonie“ an den Manager mit der IP-Adresse 10.0.0.1 gesendet. An Stelle der ersten Hochzeichen wird die Systemzeit des Net-SNMP Agenten, die unter „SNMPv2-MIB::sysUptime“ zu finden ist, automatisch angegeben. Die Meldung selbst wird über das Objekt „noZebra“ beschrieben, welches in der MIB-Datei „ZEBRA-NOTIFY-MIB.txt“ definiert ist, die weiter unten noch aufgeführt wird.

```
#!/bin/bash

NULL=0
TRUE=1
while [ "$TRUE" ]
do
 set -- `ps -A | grep -c zebra`
 RUN=$1

 if [ "$RUN" == "$NULL" ]
 then
  set -- `snmptrap -v2c -c trapatonie 10.0.0.1 "" ZEBRA-NOTIFY-MIB::noZebra`
 fi

 sleep 30
done
exit
```

Listing 6.13: Inhalt der Script-Datei „zebraTest.sh“

Damit diese Meldung so wie sie hier aufgeführt ist versendet werden kann muss noch die MIB-Datei „ZEBRA-NOTIFY-MIB.txt“ mit dem Notification-Objekt „noZebra“ erstellt werden. Über die SMI-Definition bekommt das Objekt mit Namen „noZebra“ eine Identifikationsnummer, so dass auch auf dem Manager, sofern dort über die MIB-Datei verfügt wird, aus der Beschreibung des Objektes der Grund der Meldung erfahren werden kann. Die zugehörige MIB-Datei hat den folgenden Inhalt.

```
ZEBRA-NOTIFY-MIB DEFINITIONS ::= BEGIN
IMPORTS
 NOTIFICATION-TYPE FROM SNMPv2-SMI
 studienarbeit reserved FROM STUDIENARBEIT-MIB;

notify OBJECT IDENTIFIER ::= { reserved 1 }

noZebra NOTIFICATION-TYPE
        OBJECT { INTEGER }
        STATUS current
        DESCRIPTION "no running Zebra"
        ::= { notify 123 }
END
```

Listing 6.14: Definition des Notification-Objects „noZebra“

```
+--studienarbeit(9876)
  |
  +--server-mgmt(1)
     |
     [...]
  |
  +--reserved(2)
    |
    +--notify(1)
      |
      +--noZebra(123)
```

Befehl 6.42: „snmptranslate“ über „studienarbeit“

Auf der rechten Seite ist eine leicht verkürzte Ausgabe des Befehls „snmptranslate -Tp -IR studienarbeit“ zusehen, nachdem die MIB-Datei eingebunden wurde.

Der Trap-Server des Managers schreibt mit dem Erhalt der Meldungen über das nicht Vorhandensein von Zebra Prozessen Einträge folgender Art in seine Log-Datei (/var/log/snmptrapd.log).

```
2006-04-28 00:14:34 localhost [UDP: [10.0.0.2]:32808]:
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (5840489) 16:13:24.89
SNMPv2-MIB::snmpTrapOID.0 = OID: SNMPv2-SMI::enterprises.9876.2.1.123
```

Listing 6.15: Inhalt der Log-Datei des Trap-Servers nach Erhalt der „noZebra“-Meldung

Diese Meldung kann über den folgenden Eintrag in die Konfigurationsdatei „snmptrapd.conf“ des Trap-Servers das Bash-Programm „zebra.sh“ auf dem Manager starten. Beim Start des Programms wird der Parameter „down“ übergeben.

```
trapcommunity trapatonie
traphandle ZEBRA-NOTIFY-MIB::noZebra      /tmp/zebra.sh down
```

Listing 6.16: Konfiguration der „snmptrapd.conf“ für den Empfang der Ereignismeldung „noZebra“

Hier ist nun ein Beispiel aufgeführt, wie ein Bash-Programm wie „zebra.sh“ aussehen kann. Das Programm „zebra.sh“ liest die vom Trap-Server übergebenen Werte aus und speichert sie zusätzlich nochmal in eine Textdatei mit Namen „zebraOutput.txt“. Innerhalb der While-Schleife im Programm wird die empfangene Meldung auseinandergenommen und auf die Variablen „oid“ (Objekt ID) und „val“ (Objekt-Wert) verteilt. Diese Variablen werden hier in einer While-Schleife ausgelesen, da über eine Meldung mehrere OIDs und Objekt-Werte gesendet werden können.
Die Datei „zebra.sh“ muss unter Linux die nötigen Rechte freigeschaltet bekommen (chmod 755 zebra.sh) um ausführbar zu sein. Sie beinhaltet den folgenden Bash-Script-Code.

```
#!/bin/bash
read host
read ip
vars=

while read oid val
do
  if [ "$vars" == "" ]
  then
    vars="$oid = $val"
  else
    vars="$vars, $oid = $val"
  fi
done

echo "trap: $1 $host $ip $vars" >> zebraOutput.txt
```

Listing 6.17: Inhalt des Bash-Programms „zebra.sh“

In die Datei „zebraOutput.txt“ werden durch das Bash-Programm Einträge der folgenden Art gemacht.

```
trap: down localhost UDP: [10.0.0.2]:32808 DISMAN-EVENT-MIB::sysUpTimeInstance =
0:16:43:50.31, SNMPv2-MIB::snmpTrapOID.0 = ZEBRA-NOTIFY-MIB::noZebra
```

Listing 6.18: Inhalt der Datei „zebraOutput.txt“

Dieses Beispiel zeigt, dass es nicht so schwer ist, bestimmte Ereignisse, die auf dem Agenten eintreten können, mit einer SNMP Meldung zu verknüpfen. Für den Einsatz in der Simulations-Umgebung VNUML sollte diese einfache Möglichkeit auch weitestgehend ausreichen. Natürlich können Notification-Objekte auch über die C-API direkt in den Net-SNMP Agenten eingebunden werden.

Bei der Verwendung von Net-SNMP in VNUML Netzwerk-Simulationen konnte festgestellt werden, dass die SNMP-Nachrichten insgesamt durchaus einen sehr hohen Datenverkehr verursachen, auch wenn die einzelne SNMP-Nachricht recht klein ist. Je mehr Managementinformationen mit einer einzelnen Nachricht übertragen werden können, um so effizienter aber meist auch unflexibler ist der Einsatz von SNMP. Deswegen ist beim praktischen Einsatz von SNMP gerade im Sinne des Leistungsmanagements nicht nur die Management-information, die abgefragt werden soll und das Abfrage-Intervall ein wichtiger Faktor, sondern auch die technische Umsetzung über die Anzahl der verwendeten SNMP-Nachrichten.

Noch deutlicher wird dies beim Einsatz von verschlüsselten Nachrichten mit SNMPv3, wobei hier neben dem zusätzlichen Datenverkehr auch noch der Ver- und Entschlüsselungsaufwand für jede Nachricht auf den beteiligten Netzwerk-Geräten hinzu kommt.

Mit den Protocol-Analyzer-Programmen „Ethereal“ und „Tcpdump“ kann die Funktions- und Arbeitsweise von SNMP auf Basis von Net-SNMP sehr gut eingesehen, analysiert und überprüft werden. Auch die Zusammenarbeit zwischen SNMP und den Protokollerweiterungen SMUX und AgentX kann verfolgt werden. Das Programm „Ethereal“ erkennt SNMP-Nachrichten und kann sie in die einzelnen Byte-Felder aufspalten und die Zugehörigkeit zu den entsprechenden Protokoll-Fragmenten, wie IP und UDP, aufzeigen. Es besitzt eine grafische Oberfläche und ist deswegen besser vom Host aus einzusetzen, während „Tcpdump“ ohne grafische Oberfläche auch innerhalb der UML-Rechner zu verwenden ist. Da die virtuellen Netzwerkschnittstellen der UML-Rechner, bis auf die internen wie den „localhost“, im Host-System abgebildet sind, wie in Abbildung 5.8 zu sehen ist, kann Ethereal in den meisten Fällen ohne Probleme zur Verkehrsdatenanalyse vom Host aus eingesetzt werden.

Im folgenden ist eine Tabelle dargestellt, über die der unterschiedliche Ressourcenverbrauch zwischen den Versionen SNMPv1/-v2c und SNMPv3 etwas verdeutlicht werden soll. In der Tabelle sind Leistungsdaten zu SNMP-Anfragen an den Agenten aufgeführt. Es wurde immer die gleiche Anfrage aber mit unterschiedlicher Objekt-Anzahl pro Nachricht und unterschiedlicher SNMP Version gestellt. Bei der verwendeten Netzwerktopologie handelt es sich um die der isolierten Netzwerk-Simulation aus Abbildung 6.1, der Agent ist also direkter Nachbar des Managers.
Über ein Bash-Script werden vom Manager aus die Managementinformationen der drei Objekte „sysName“, „sysContact“ und „sysDescr“ aus der Gruppe „system“ vom Agenten abgefragt. Das Bash-Programm für die SNMPv1/-v2c Anfrage über mehrere Nachrichten entspricht dem Programm „snmpQuery.sh aus Listing 6.3, ohne die Abfrage und Verarbeitung der „interfaces“-Objekte. Die Datenausgabe des Bash-Programms gleicht der Ausgabe von „snmpQuery.sh“, wie im Befehl 6.13 dargestellt, ohne die Ausgabe der „interfaces“-Objekte. Auch die erhaltenen Managementinformationen sind die gleichen, da die drei Objekte feste Werte beinhalten. Es werden nur Net-SNMP Get-Befehle (snmpget) verwendet. Für SNMPv1/-v2c wird der Community-Name „public“ verwendet. Für SNMPv3 wird der Benutzer „mySysUser“ mit dem Authentifizierungspasswort „my_password“ und dem Verschlüsselungspasswort „other_password“ verwendet. Der zeitliche Aufwand, den das Bash-Script zur Auswertung und zur Ausgabe in der Konsole benötigt, wurde mit dem Linux-Programm „time“ innerhalb des Managers gemessen. Es handelt sich dabei um einen Mittelwert aus mehreren Durchführungen. Der Datenverkehr wurde mit dem Protocol-Analyzer Programm „Ethereal“ im Host-System über der Software-Bridge „Net0“ gemessen.

Versandart der Objekte	SNMP Version	Auswertungszeit (sekunden)	Anzahl gesendeter Nachrichten	gesendete Datenmenge
Jedes Objekt über eine eigene Nachricht	SNMPv1/-v2c	real: ~ 0,960 s	6	705 Byte
	SNMPv3 (noAuthNoPriv)	real: ~ 0,970 s	12	1996 Byte
	SNMPv3 (authNoPriv)	real: ~ 1,050 s	12	2061 Byte
	SNMPv3 (authPriv)	real: ~ 1,160 s	12	2164 Byte
Alle Objekte über eine einzige Nachricht	SNMPv1/-v2c	real: ~ 0,340 s	2	359 Byte
	SNMPv3 (noAuthNoPriv)	real: ~ 0,350 s	4	791 Byte
	SNMPv3 (authNoPriv)	real: ~ 0,380 s	4	815 Byte
	SNMPv3 (authPriv)	real: ~ 0,400 s	4	845 Byte

Tabelle 6.1: Vergleich des Ressourcenverbrauchs der einzelnen SNMP-Versionen

Die zusammengefasste SNMP-Anfrage, bestehend aus einer einzigen Nachricht in der alle Objekte enthalten sind, benötigt eine geringere Versende- und Auswertungszeit als die aufgeteilte SNMP-Anfrage, bestehend aus einer Nachricht für jedes einzelne Objekt. Die Menge der gesendeten Daten ist hier, bei den einzeln verarbeiteten Objekten, jeweils mindestens doppelt so groß gegenüber den in einer Nachricht zusammengefassten Objekten. Bei SNMPv3 wird vor jedem Get/Response Nachrichtenaustausch ein Get/Report Nachrichtenaustausch gestartet, der Managementinformationen aus der „SNMP-USER-BASED-SM-MIB“ überträgt, was zusätzlich noch das Verdoppeln der Anzahl der Nachrichten verursacht. Somit ist die Authentifizierung und Verschlüsselung hier nicht die eigentliche Ursache für die gestiegene Menge an versendeten Daten mit SNMPv3, aber sehr wohl verantwortlich für die längere Auswertungszeit.

6.7 Zusammenfassung

In diesem Kapitel wurde die Software Net-SNMP vorgestellt, installiert, konfiguriert und über Beispiele einige Anwendungsmöglichkeiten erläutert. Da die UML-Rechner von VNUML Linux-Rechner emulieren, wurde festgestellt, dass es hier keine großen Unterschiede zur Installation und Konfiguration auf einem reellen Rechner gibt. Der Host-Rechner einer VNUML-Simulation ist jedoch besser für die Funktion des SNMP-Managers geeignet, da hier auch komplexere Management-Applikationen eingesetzt werden können, die auf den Net-SNMP Manager Programmen aufbauen und die Managementinformationen grafisch oder statistisch direkt weiterverarbeiten. Auch gibt es bei der Installation von Net-SNMP in das Root-Dateisystem der VNUML-Autoren aufgrund fehlender Software, die für die erfolgreiche Installation und Funktion benötigt wird, einige Hürden zu überwinden.
Der Vorteil von Net-SNMP im allgemeinen ist die sehr gute Unterstützung von Linux-Systemen. Es können zahlreiche Managementinformationen über das Linux System und seinen Zustand direkt nach der Installation via SNMP abgefragt werden.
Nachteilig bei der Software Net-SNMP für die Anwendung als VNUML-Management-Werkzeug sind aber einige nachvollziehbare Einschränkungen, wie die Beschränkung auf das Lesen der Managementinformationen vieler Objekte und das standardmäßig eingestellte Aktualisierungs-Intervall des Net-SNMP Caches von gut einer Minute. Mit dem Abstellen dieser Beschränkungen ist Net-SNMP sicherlich eine sehr Interessante Erweiterung des VNUML-Managements um ein flexibles Leistungs- und Überwachungsmanagement. Net-SNMP wurde jedoch nicht für Simulations-Umgebungen entworfen, so dass die Einstellungen aus Gründen der Sicherheit und der Entlastung des Netzwerks für den alltäglichen Einsatz nachvollziehbar sind.
Net-SNMP kann in VNUML-Simulationen jedoch auch sehr gut dazu verwendet werden SNMP-Manager-Anfragen auszuarbeiten und für weitestgehend beliebige Netzwerk-Topologien zu Optimieren, um zum Beispiel den SNMP-Datenverkehr so effizient und so klein wie möglich zu gestalten.

Alles in allem verdeutlicht der praktische Einsatz von Net-SNMP in VNUML-Simulationen den sehr interessanten Weg, der mit SNMP bei der plattformübergreifenden Standardisierung des Netzwerk-managements eingeschlagen worden ist. Es werden aber auch die Schwachstellen des SNMP-Management-Modells und der sehr netzwerkbelastenden Client/Server-Architektur erkennbar. So ist es fraglich, ob es mit SNMP auch möglich wäre, dass Netzwerk-Komponenten, die über genug Leistungsreserven verfügen, die Protokollierung ihres Systems für das Leistungsmanagement auch selbst übernehmen und nur noch in recht großen Abständen die zusammenhängenden Protokolldaten in einem Multimedia-Format, zum Beispiel als grafische Leistungskurve in einem Bildformat, an den Manager und dessen Applikationen zur Weiterverarbeitung senden. In komplexeren, modernen Managementumgebungen, in denen die zu verwaltenden Management-Objekte nicht nur „Status-“ oder „Schaltercharakter“ besitzen sollen, sondern auch komplexe Multimedia-Daten, wie spezielle Dateiformate, Bilder, Videostreams oder ganze Programme sein sollen, ist SNMP in seiner jetzigen Form sicherlich nicht geeignet.

Anhang: Anleitung

7 Erstellen eines VNUML-Rechners auf Basis von Gentoo-Linux

Im Laufe der praktischen Ausarbeitung dieser Studienarbeit wurde ein UML-Root-Dateisystem auf Basis der Gentoo-Linux Distribution sowie ein UML-Kernel für den Einsatz in VNUML Netzwerk-Simulationen erstellt. In diesem Kapitel wird die Vorgehensweise dabei aufgezeigt.

7.1 Einleitung

Im Rahmen dieser Studienarbeit wurde die SNMP-Software Net-SNMP in ein UML-Root-Dateisystem installiert, um in VNUML Netzwerk-Simulationen zum Einsatz zu kommen. Es erwies sich jedoch als äußerst schwierig Net-SNMP mit der gewünschten Konfiguration in das von den VNUML-Autoren bereitgestellte UML-Root-Dateisystem zu installieren. In der schriftlichen Ausarbeitung der Studienarbeit werden einige Hinweise gegeben, wie bei der Installation von Software in dieses Root-Dateisystem im allgemeinen vorgegangen werden kann.
Die Installation von Net-SNMP und Quagga, insbesondere die manuelle Installation über den Quellcode, war jedoch nur sehr umständlich und nicht mit den gewünschten Konfigurationen zu realisieren.
Die VNUML-Projektgruppe verwendet für ihr UML-Root-Dateisystem die Linux-Distribution Debian. Die Installationsroutine von Debian mit den „APT“-Tools installiert für gewöhnlich vorkompilierte Software in Form von Binärpaketen, deren Konfiguration nur begrenzt beeinflusst werden kann. Da es sich bei dem UML-Root-Dateisystem der VNUML-Autoren zusätzlich um ein sehr klein gehaltenes Debian-Linux-System handelt, in dem nur die notwendigste Software installiert ist, gibt es gerade hier Probleme damit, das manuelle Kompilieren und Installieren von Software, für eine individuelle Konfiguration, durchzuführen.

Aufgrund der Schwierigkeiten bei der Installation von Net-SNMP wurde während der Ausarbeitung der Studienarbeit ein UML-Root-Dateisystem für den Einsatz in VNUML erstellt, das auf der Gentoo-Linux Distribution basiert. Gentoo verfügt, wie Debian, über eine Online-Installationsroutine, genannt „Portage“, über die eine riesige Auswahl an Linux-Software von Servern aus dem Internet heruntergeladen und in das lokale System installiert werden kann. Der große Unterschied zwischen Debian und Gentoo ist hier jedoch, dass Gentoo lediglich den Quelltext der Software aus dem Internet herunterlädt, lokal kompiliert und dann installiert. Dementsprechend bietet die Installationsroutine von Gentoo auch sehr umfangreiche Einstellungsmöglichkeiten für die Installation der Software. Gentoo-Linux ist somit eine Art „Linux from Scratch“ plus komfortable Installationsroutine. Es kann daher auch davon ausgegangen werden, dass es in Gentoo-Linux eher möglich ist Software ohne größere Probleme auch manuell zu kompilieren und zu installieren.
Bei Gentoo kann es aufgrund des lokalen kompilierens der Software allerdings auch öfters zu Problemen kommen, weil zum Beispiel verschiedene Versionen voneinander abhängiger Programme nicht miteinander klar kommen. Oftmals wird zum Lösen solcher Probleme die Hilfe der Benutzer-Gemeinde benötigt. Einige Internet-Adressen von Gentoo-Linux und der Benutzer-Gemeinde von Gentoo sind im folgenden aufgelistet:

- http://www.gentoo.org - Gentoo Projekt Homepage
- http://www.gentoo.de - Deutsche Gentoo Projekt Homepage
- http://forums.gentoo.org - Internationale Hilfe-Foren für Probleme mit Gentoo-Linux
- http://gentoo-wiki.com - Zahlreiche Anleitungen für das Einrichten verschiedener Programme
- http://de.gentoo-wiki.com - Anleitungen für Programm-Einrichtungen in deutscher Sprache
- http://gentoo-portage.com - Erläuterungen zu Softwarepaketen und deren Konfigurationsmöglichkeiten

7.2 Erstellen eines VNUML-Root-Dateisystems auf Basis von Gentoo-Linux

Gentoo-Linux ist eine quellbasierte Linux-Meta-Distribution, die das individuelle Einrichten des kompletten Systems erlaubt. Die Distribution benutzt nicht wie viele andere Distributionen vorkompilierte Programmpakete, sondern Quelltexte, die durch das distributionseigene Paketmanagement kompiliert werden.
Das Basis-System von Gentoo-Linux ist in verschiedenen Kompilierstufen, den sogenannten „Stages“, erhältlich. Derzeit wird zwischen drei „Stages“ unterschieden, bei denen das Gentoo Basis-System bis zu einer gewissen Stufe fertig kompiliert ist und nur installiert werden muss.

„Stage 3“ beinhaltet ein komplett lauffähiges, minimales Gentoo-Linux Basis-System. „Stage 1“ verlangt vom „Bootstrapping“ angefangen das Kompilieren aller notwendigen Systemanwendungen für die Funktion, was letztlich ein perfekt an die eigene Hardware und eigenen Bedürfnisse angepasstes Betriebssystem erlaubt. Mit „Stage 1“ kann das komplette Linux-System aus den Quelltexten gebaut werden, mit individuell eingestellten Optimierungen. „Stage 2“ liegt mit dem Aufwand für das Kompilieren von Systemanwendungen zwischen Stage 1 und 3.

Gentoo hat ein, von BSD-Betriebssystemen bekanntes, „ports“-ähnliches Paketverwaltungssystem namens „Portage“. Das Hauptwerkzeug von „Portage“ ist das Programm „emerge“. Für ein Softwarepaket sind Abhängigkeiten, Downloadquelle und Ablauf des Kompilierens und der Installation in sogenannten „ebuild“-Scripten festgelegt. Anhand dieser Information ist es „Portage“ möglich Abhängigkeiten selbständig aufzulösen, Updates durchzuführen und vieles mehr. Spezielle optionale Funktionen eines Softwarepaketes werden über „USE-Flags“ gesteuert, die „global“ oder „lokal“ für jedes Paket einzeln festgelegt werden können.
„Portage“ besitzt mit dem „Portage-Tree“ eine Verzeichnis-Struktur die unter „/usr/portage/“ im Gentoo-System zu finden ist. Hier sind die „ebuild“-Scripte, die Gentoo für die Verwaltung der Software benötigt, abgelegt. Die Verzeichnis-Struktur ist hierarchisch organisiert und nach verschiedenen Kategorien in spezifische Softwarepaket-Verzeichnisse untergliedert. Der „Portage-Tree“ ist mit seinen „ebuild“-Scripten das Kernstück der Installationsroutine von Gentoo-Linux. Die heruntergeladenen Quelltexte der Softwarepakete werden für gewöhnlich im Verzeichnis „/usr/portage/distfiles/“ lokal gespeichert. Kompiliert und installiert wird die Software schließlich aus dem Verzeichnis „/var/tmp/portage/“ heraus.

7.2.1 Vorbereiten des Gentoo-Systems

Für die Erstellung eines UML-Root-Dateisystems auf Basis von Gentoo-Linux für den Einsatz in VNUML müssen auf dem Host-System folgende Voraussetzungen erfüllt sein:

- ausreichende Zugriffsrechte auf dem Host-System, am besten root-Rechte (SuperUser)
- bestehende Internetverbindung
- ca. 3 Gigabyte freien Speicherplatz

Zunächst muss ein Gentoo-Linux Basis-System sowie ein aktueller „Portrage-Tree“ von einem Gentoo-Spiegel-Server aus dem Internet heruntergeladen werden. Eine Liste mit zahlreichen Servern, auf denen Gentoo-Linux zum download bereit steht, ist auf der Gentoo Projekt Homepage, unter der Internet-Adresse „www.gentoo.org/main/en/mirror.xml“ zu finden.

Als Gentoo-Linux Basis-System genügt hier ein „Stage3“-System da es nicht erst zeitaufwendig bearbeitet werden muss um als UML-Root-Dateisystem für VNUML oder einen einzelnen UML-Rechner eingesetzt zu werden. Ein ensprechender „Stage3“-Tarball ist auf den Gentoo-Servern im Verzeichnis „gentoo/releases/'Architektur'/current/stages“ zu finden. Der Bezeichner 'Architektur' steht für die Architektur des Rechners, die wegen der Virtualisierung aber eher unerheblich sein dürfte. Um Probleme zu vermeiden sollte aber trotzdem die Architektur des Host-Rechners verwendet werden.
Zusätzlich wird eine aktuelle Ausgabe des „Portage“-Verzeichnisbaums benötigt. Ein „Portage“-Tarball ist auf den Spiegel-Servern im Verzeichnis „gentoo/snapshots/“ zu finden. Für gewöhnlich wird hier täglich eine Aktualisierung veröffentlicht.

Mit dem Download-Programm „wget“ werden nun die benötigten Tarballs von einem Spiegel-Server aus dem Internet auf das lokale Host-System heruntergeladen und im Verzeichnis „download“ gespeichert.

```
host # mkdir download
host # cd download
host download # wget ftp://ftp.uni-koblenz.de/pub/mirrors/gentoo/releases/x86/current/stages/stage3-x86-2006.0.tar.bz2
host download # wget ftp://ftp.uni-koblenz.de/pub/mirrors/gentoo/snapshots/portage-latest.tar.bz2
host download # cd ..
```

Befehl 7.1: Download des „Stage3-“ und des „Portage“-Tarballs

Anschließend folgt das Entpacken der beiden Tarballs.

Der Tarball, der das Gentoo-Linux Basis-System enthält, kann einfach in ein Verzeichnis entpackt werden, hier in das Verzeichnis „gentoo_backup“.

```
host # mkdir gentoo_backup
host # tar xfjvp download/stage3-x86-2006.0.tar.bz2  -C  gentoo_backup/
```

Befehl 7.2: Entpacken des „Stage3“-Tarballs

In dem Verzeichnis „gentoo_backup/“ befindet sich nun schon ein verwendbares UML-Root-Dateisystem, das für den Einsatz in VNUML aber noch entsprechend konfiguriert werden muss.

Der zweite Tarball, der den „Portage-Tree“ enthält, könnte nun einfach im Verzeichnis „gentoo_bakup/usr/“ entpackt werden. Da der „Portage“-Verzeichnisbaum allerdings später aus Platzgründen wieder entfernt werden soll, wird hier zuvor eine Imagedatei erstellt und an diese Stelle in der Verzeichnisstruktur eingebunden. So wird der „Portage“-Verzeichnisbaum in die Imagedatei abgelegt und kann, wenn er später vorerst nicht mehr benötigt wird, einfach aus dem System gelöst werden, ohne gelöscht werden zu müssen. So bleibt er jederzeit verfügbar und kann, auch in laufende UML-Rechner, für die Installation oder Aktualisierung von Software wieder eingebunden werden. Der „Portage“-Verzeichnisbaum kann sehr groß werden, weswegen die Imagedatei nicht zu klein erstellt werden darf. Hier ist sie 2000 MB groß, da der Speicher allerdings nur reserviert und nicht gleich belegt wird, fehlt dem Host-System dieser Speicherplatz erstmal nicht.

```
host # dd if=/dev/zero  of=portage.img  seek=2000  count=1 bs=1M
host # mke2fs -F portage.img
host # mkdir -p gentoo_backup/usr/portage
host # mount -o loop portage.img gentoo_backup/usr/portage
host # tar xfjvp download/portage-latest.tar.bz2  -C  gentoo_backup/usr/
```

Befehl 7.3: Entpacken des „Portage“-Tarballs

Um aus dem UML-Root-Dateisystem heraus auch auf Internet-Adressen zugreifen zu können, muss noch die Datei „resolv.conf“ des lokalen „Resolvers“ aus dem Host-System in das Gentoo-System kopiert werden.

```
host # cp /etc/resolv.conf   gentoo_backup/etc/
```

Befehl 7.4: Kopieren der Datei „resolv.conf“ für den Zugriff auf Internet-Adressen

Das Gentoo-System ist nun soweit fertig vorbereitet. Über den Befehl „chroot“ kann in die Gentoo-System-Umgebung gewechselt werden. Zuvor sollte noch das „proc“-Verzeichnis gemountet werden. Nach dem Wechsel der Umgebung müssen auch die Umgebungsvariablen für den Benutzer aktualisiert werden.

```
host # mount -t proc proc gentoo_backup/proc
host # chroot  gentoo_backup  /bin/bash
/ # env-update &&  source /etc/profiles
>>> Regenerating /etc/ld.so.cache. . .
```

Befehl 7.5: Wechsel in die „chroot“-Umgebung in das Gentoo-Linux-System

Damit kann nun mit der Konfiguration des Gentoo-Systems begonnen werden.

7.2.2 Konfiguration des Gentoo-Systems für VNUML

Die Konfiguration des Gentoo-Systems für den Einsatz in VNUML ist recht kompliziert und kann hier nicht in vollem Umfang erläutert, sondern lediglich oberflächlich aufgezeigt werden. Zahlreiche Konfigurationsdateien müssen modifiziert werden, damit ein UML-Rechner auf Basis von Gentoo-Linux über VNUML gestartet und in eine Netzwerk-Simulation eingebunden werden kann.
Zu Beginn wird hier dem System ein Passwort gegeben, mit dem sich der UML-Benutzer „root“ später einloggen kann. Dafür wird das einfache Passwort 'xxxx' gesetzt, das auch von den VNUML-Autoren für deren UML-Rechner verwendet wird.

```
/ # passwd
New UNIX password: xxxx
BAD PASSWORD: it is too short
Retype new UNIX password: xxxx
passwd: password updated successfully
```

Befehl 7.6: Setzen eines Passwortes für den UML-Rechner Benutzer „root“

Über die Datei „/etc/fstab“ müssen die „Mountpunkte“ für die verschiedenen Partitionen des UML-Rechners gesetzt werden. Die folgende „fstab“-Datei ist für die Verwendung mit „devfs“ und für die VNUML Version 1.5 ausgelegt. Als Text-Editor bietet sich unter Gentoo-Linux das Programm „nano“ zum editieren an.

```
<fs>            <mntpoint>      <type>      <opts>                  <dump/pass>
/dev/ubd/0      /               ext2        defaults                    0 1
/dev/ubd/1      /mnt/vnuml      ext2        defaults                    0 0

devpts          /dev/pts        devpts      mode=0622                   0 0
proc            /proc           proc        defaults                    0 0
shm             /dev/shm        tmpfs       nodev,nosuid,noexec         0 0
```

Listing 7.1: Konfiguration der Datei „/etc/fstab“

Die Einstellungen in der Datei „/etc/fstab“ gehören zu den wichtigsten Konfigurationen des UML-Rechners. Ist die Datei falsch konfiguriert, so funktioniert der UML-Rechner nicht.

VNUML in der Version 1.5 verlangt nun noch die Erstellung eines „Mountpunktes“ für die zweite Partition, hier das Verzeichnis „/mnt/vnuml/“, sowie für die Funktion des VNUML-Managements via SSH notwendig, einen Link auf dieses Verzeichnis vom SSH-Verzeichnis des UML-Benutzers „root“ aus.

```
/ # mkdir /mnt/vnuml
/ # ln -sf /mnt/vnuml /root/.ssh
```

Befehl 7.7: Erstellen des „Mountpunktes“ für die zweite Partition und SSH-Verlinkung

Für die automatische, individuelle Einrichtung der UML-Rechner der Netzwerk-Simulationen geht VNUML so vor, dass es innerhalb der zweiten Partition (/dev/ubd/1) eine Script-Datei erstellt, die beim Start des UML-Rechners ausgeführt wird und diesen, entsprechend den Vorgaben der Szenario-Beschreibung, konfiguriert. Eine genauere Erläuterung der Vorgehensweise ist in Kapitel 5.5 zu finden.
Die Script-Datei, genannt „umlboot“, und eine SSH-Authentifizierungs-Datei, genannt „authorized_keys“, liegen dann im gestarteten UML-Rechner im Verzeichnis „/mnt/vnuml/“, bzw. aufgrund des Links auch im Verzeichnis „/root/.ssh/“, was für die Funktion der Authentifizierungs-Datei notwendig ist.

Damit die Script-Datei beim Start des UML-Rechners ausgeführt wird, muss sie aus einem Runlevel heraus referenziert sein. Hierfür muss ein Link aus dem Default-Runlevel auf die Datei „/mnt/vnuml/umlboot“ gesetzt werden, was in Gentoo-Linux ein Init-Script voraussetzt. Dieses Init-Script, hier genannt „vnumlboot“, muss im Verzeichnis „/etc/init.d/“ abgelegt sein und den folgenden Inhalt besitzen.

```
#!/sbin/runscript

depend() {
need localmount
use net
}

start() {
 ebegin "umlboot starts network interfaces config"
  if [ ! -x /mnt/vnuml/umlboot ]
  then
         eerror   "Unable to locate /mnt/vnuml/umlboot script !"
         return 1
  fi

 /mnt/vnuml/umlboot
 retval=$?
 eend ${retval} "Failed to run /mnt/vnuml/umlboot script !"
}
```

Listing 7.2: Das Init-Script „vnumlboot“ für die Referenz auf „umlboot“

Dieses Init-Script muss ausführbar sein und über den Befehl „rc-update“ in das Default-Runlevel gesetzt werden.

```
/ # chmod 755 /etc/init.d/vnumlboot
/ # rc-update add vnumlboot default
* vnumlboot added to runlevel default
*rc-update complete
```

Befehl 7.8: Setzen des Init-Scripts „vnumlboot“ in den Default-Runlevel

Die Terminal-Ausgabe des Gentoo UML-Rechners wird über die Datei „/etc/inittab“ konfiguriert. Normalerweise öffnen sich sechs X-Terminals für jeden UML-Rechner wenn die Ausgabe nicht beim Start über die „con“-Optionen eingeschränkt wird.

```
# TERMINALS
#c1:12345:respawn:/sbin/agetty 38400 tty1 linux
#c2:12345:respawn:/sbin/agetty 38400 tty2 linux
#c3:12345:respawn:/sbin/agetty 38400 tty3 linux
#c4:12345:respawn:/sbin/agetty 38400 tty4 linux
#c5:12345:respawn:/sbin/agetty 38400 tty5 linux
#c6:12345:respawn:/sbin/agetty 38400 tty6 linux

0:12345:respawn:/sbin/agetty 38400 tty0
```

Listing 7.3: Konfiguration der Login-Terminals über die Datei „/etc/inittab“

Mit dieser Konfiguration in der Datei „inittab“ öffnet sich kein X-Terminal beim Start. Der Login für den UML-Rechner öffnet sich direkt in der Konsole von der aus der UML-Rechner gestartet wurde. Ein X-Terminal kann für den UML-Rechner aber trotzdem über die Befehlsoption „con0=xterm“ und bei VNUML über das <con0> Tag der VNUML-Sprache im Szenario aktiviert werden.

Der Eintrag „tty0“ muss sich auch in der Datei „/etc/securetty“ befinden, was aber für gewöhnlich bereits der Fall ist.

VNUML verwendet für die Verwaltung der UML-Rechner der Netzwerk-Simulationen das SSH-Protokoll und dessen Anwendungen. Damit SSH einwandfrei funktioniert muss es noch konfiguriert werden. SSH ist im Gentoo-Linux Basis-System bereits installiert. Es besitzt Konfigurationsdateien für den SSH-Client und für den SSH-Server im Verzeichnis „/etc/ssh/“ sowie das Init-Script „/etc/init.d/sshd“ für den automatischen Start.

Mit dem Befehl „rc-update“ muss das Init-Script „sshd“ in den Default-Runlevel gesetzt werden, um den SSH-Server automatisch während des Bootvorgangs zu starten.

```
/ # rc-update add sshd default
* sshd added to runlevel default
*rc-update complete
```

Befehl 7.9: Setzen des Init-Scripts „sshd“ in den Default-Runlevel

Die VNUML-Autoren empfehlen eine Veränderung der Start-Optionen des SSH-Servers um „sshd -u0“. Dabei wird die Länge des Feldes, das den Remote Host Namen in der „utmp“-Struktur enthält auf Null gesetzt. Dies vermeidet DNS-Anfragen des SSH-Servers die den SSH-Zugriff in VNUML Simulationen verzögern können.
Um diese Modifizierung durchzuführen kann innerhalb der Datei „/etc/init.d/sshd“ in der start()-Methode hinter der aufgeführten Startsequenz „/usr/sbin/sshd“ einfach die Option -u0 gesetzt werden.

Nun sollten noch die nötigen Schlüssel für die verschiedenen SSH-Protokolle erzeugt werden. Dies geschieht wie folgt.

```
/ # /usr/bin/ssh-keygen -t rsa1 -b 1024 -f /etc/ssh/ssh_host_key -N ''
/ # /usr/bin/ssh-keygen -d -f /etc/ssh/ssh_host_dsa_key -N ''
/ # /usr/bin/ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key -N ''
```

Befehl 7.10: Erzeugen der lokalen SSH-Schlüssel für den UML-Rechner

Die verschiedenen SSH-Schlüssel müssen in die Datei „/etc/ssh/“ abgelegt werden. Sind die SSH-Schlüssel beim Start des UML-Rechners nicht vorhanden, so werden sie durch das Init-Script „sshd“ erzeugt. Dabei wird aber die Boot-Sequenz des Gentoo-UML-Rechners erheblich verlängert. Außerdem können auch Probleme mit der SSH-Verbindung zum Host-System entstehen. Die im UML-Rechner erstellten SSH-Schlüssel liegen bei einer VNUML Simulation für gewöhnlich in der COW-Imagedatei und sind beim Verlust dieser ebenfalls gelöscht. Ohne COW-Imagedatei müssen die SSH-Schlüssel beim nächsten Durchführen der Simulation wieder erstellt werden. Das Host-System speichert aber eventuell SSH-Schlüssel und zugehörige IP-Adressen in einer Datei ‚meist genannt „known_hosts“, im Verzeichnis „.ssh/“ des SSH-Benutzers ab. Der Host kann nun SSH-Verbindungen aus Sicherheitsgründen ablehnen, wenn von einer bereits bekannte IP-Adresse aus mit einem anderen SSH-Schlüssel eine Verbindung aufgebaut werden soll. Wurde der SSH-Schlüssel bereits in der Imagedatei erstellt ist es für alle VNUML-Rechner in allen Netzwerk-Simulationen immer der Selbe und diese Probleme treten nicht auf.

Abschließend muss noch die Konfigurationsdatei „/etc/ssh/sshd_config“ des SSH-Servers konfiguriert werden. So verwendet VNUML in der Version 1.5 für gewöhnlich das SSH-Protokoll 1, Gentoo-Linux aber standardmäßig nur das SSH-Protokoll 2. Auch die Authentifizierungs-Datei „authorized_keys“ kann hier eingestellt werden und die bereits installierte PAM-Unterstützung (Pluggable Authentication Module) kann, sofern bei SSH nicht benötigt, abgestellt werden. Für den Zugriff auf den SSH-Server des UML-Rechners muss auch die Variable „PasswordAuthentication“ aktiviert sein.

```
[...]
Protocol 2,1
[...]
AuthorizedKeysFile .ssh/authorized_keys
[...]
PasswordAuthentication yes
[...]
#UsePAM no
[...]
```

Listing 7.4: Konfiguration der Datei „/etc/ssh/sshd_config“

UML-Rechner auf Basis von Gentoo-Linux benötigen in ihrer unveränderten Basiskonfiguration eine sehr, sehr lange Zeit für das vollständige Hochfahren. Um die Bootzeit erheblich zu verkürzen, müssen einige Einstellungen in verschiedenen Init-Scripten vorgenommen werden. Diese Modifizierungen sind unter dem Begriff „Flying with Gentoo“ in den Internet-Benutzer-Hilfe-Foren von Gentoo zusammengefasst.

Mit der Konfiguration der folgenden Modifizierungen wird die Länge der Startzeit von einzelnen UML-Rechnern und ganzen VNUML Netzwerk-Simulationen auf ca. ein Drittel der Startzeit ohne diese Modifizierungen verkürzt. Werden alle Modifizierungen durchgeführt so liegt die Startzeit einer VNUML Netzwerk-Simulation mit dem Gentoo-UML-Root-Dateisystem sehr nahe bei der Startzeit einer Netzwerk-Simulation mit dem Debian-UML-Root-Dateisystem der VNUML-Autoren. Ohne diese Modifizierungen ist ein vernünftiges Arbeiten mit den Gentoo-UML-Rechnern in VNUML aufgrund der sehr langen Startzeit nicht wirklich angenehm.

Im folgenden wird zuerst der ursprüngliche Quellcode in den einzelnen Dateien aufgezeigt und anschließend der modifizierte Quellcode, in dem die Veränderungen zusätzlich durch Fett-Schrift markiert sind.

In der Datei „/etc/init.d/modules“ innerhalb der Methode start():

```
ebegin "Calculating module dependencies"
   /sbin/modules-update
eend $? "Failed to calculate dependencies"
```

Listing 7.5.1: Original Code der Datei „/etc/init.d/modules“

Diese Stelle im Quellcode muss wie folgt um eine if-Anweisung erweitert werden:

```
if [[ /etc/modules.d -nt /etc/modules.conf ]] ; then
   ebegin "Calculating module dependencies"
      /sbin/modules-update
   eend $? "Failed to calculate dependencies"
else
   einfo "Module dependencies are up-to-date"
fi
```

Listing 7.5.2: Modifizierter Code der Datei „/etc/init.d/modules“

In der Datei „/etc/init.d/localmount“ innerhalb der Methode start():

```
mount -at noproc,noshm,no${NET_FS_LIST// /,no} > /dev/null
```

Listing 7.6.1: Original Code der Datei „/etc/init.d/localmount“

Diese Stelle im Quellcode der Datei muss um die Option F, für das parallele Mounten, erweitert werden:

```
mount -aFt noproc,noshm,no${NET_FS_LIST// /,no} > /dev/null
```

Listing 7.6.2: Modifizierter Code der Datei „/etc/init.d/localmount“

In der Datei „/etc/init.d/bootmisc“ innerhalb der Methode start():

```
if [[ -x /sbin/env-update.sh ]] ; then
   ebegin "Updating environment"
      /sbin/env-update.sh > /dev/null
   eend 0
fi
```

Listing 7.7.1: Original Code der Datei „/etc/init.d/bootmisc“

Diese Stelle im Quellcode muss um eine if-Anweisung erweitert werden:

```
if [[ -x /sbin/env-update.sh ]] ;  then
   if [[ /etc/env.d -nt /etc/profile.env ]] ; then
      ebegin "Updating environment"
      /sbin/env-update.sh > /dev/null
      eend 0
   else
      einfo "Environment up-to-date"
   fi
fi
```

Listing 7.7.2: Modifizierter Code der Datei „/etc/init.d/bootmisc“

In der Datei „/etc/conf.d/rc“:

```
RC_PARALLEL_STARTUP=no
```

Listing 7.8.1: Original Code der Datei „/etc/conf.d/rc“

Für das parallele Starten der Dienste beim Hochfahren muss die Variable auf „yes“ gesetzt werden:

```
RC_PARALLEL_STARTUP=yes
```

Listing 7.8.2: Modifizierter Code der Datei „/etc/conf.d/rc“

Um die Länge der Bootzeit noch weiter zu verkürzen können einige nicht benötigte Dienst-Programme aus den Runleveln entfernt werden. Dazu dient das bereits vorgestellte Programm „rc-update“. Mittels „rc-update“ können die Init-Scripte, die auf die Programme verweisen, angezeigt und aus den Runleveln entfernt werden.
Über den Befehl „rc-update show“ können alle Init-Scripte und die zugehörigen Runlevel, aus denen sie referenziert werden, angezeigt werden.

```
/ # rc-update show
      bootmisc      |  boot
      checkfs       |  boot
      checkroot     |  boot
      [...]
      sshd          |          default
      vnumlboot     |          default
```

Befehl 7.11: Anzeigen der Init-Scripte, die beim Hochfahren automatisch ausgeführt werden

Ein Teil dieser Init-Scripte kann aus den Runleveln entfernt werden, da ihre Funktion für die virtuelle Netzwerk-Umgebung nicht notwendig ist. Im folgenden sind fünf Init-Scripte aufgelistet, deren entfernen kein wirklicher Nachteil für die Funktion des Gentoo-Systems bedeutet:

```
/ # rc-update del consolefont    boot
      "   "      domainname     boot
      "   "      hostname       boot
      "   "      keymaps        boot
      "   "      local          default
      "   "      local          nonetwork
```

Befehl 7.12: Entfernen einiger nicht benötigter Init-Scripte aus deren Runlevel

Damit ist das Gentoo-Root-Dateisystem für den Einsatz in VNUML fertig konfiguriert. Für die vollständige Funktion fehlt allerdings noch der „devfs“-Daemon, der im Kapitel 7.2.3 installiert wird.

Während der Arbeit mit VNUML Netzwerk-Simulationen wurde festgestellt, dass es aufgrund der aktivierten „rp_filter“ Variable (Reverse Patch) in Netzwerken mit alternativen Routen zwischen UML-Rechnern zu Problemen kommen kann, wenn ein UML-Rechner empfangene Datenpakete über eine andere Netzwerkschnittstelle beantworten möchte und nicht über die Netzwerkschnittstelle, über die er die Datenpakete auch empfangen hat. Ist dies der Fall, verwirft der UML-Rechner die eingegangen Datenpakete unbeantwortet da es sich für ihn auch um ein Sicherheitsproblem handeln könnte. Das Verhalten der „rp_filter“-Variable ist im RFC 1812 „Requirements for IP Version 4 Routers“ beschrieben. Um dieses Verhalten zu ändern und jeglichen Verkehr zu ermöglichen muss in der Datei „/etc/sysctl.conf“ die Filter Variable deaktiviert werden. Diese Datei verändert die Kernel-Variablen, die sich im gestarteten UML-Rechner im Verzeichnis „/proc/sys/“ befinden und dann auch mit dem Programm „sysctl“ beeinflusst werden können.

```
net.ipv4.conf.all.rp_filter = 0
```

Listing 7.9: Deaktivieren der „rp_filter“ Variable über die Datei „/etc/sysctl.conf“

Hier sind jetzt noch einige Anmerkungen zu der Konfiguration des UML-Root-Dateisystems für den Einsatz in VNUML Version 1.6 sowie für „udev“ statt „devfs“ aufgeführt.

Um das UML-Root-Dateisystem in VNUML Version 1.6 verwenden zu können muss die Konfiguration an zwei Stellen verändert werden, in der Datei „/etc/fstab“ sowie der Link „/root/.ssh“ auf das Verzeichnis „/mnt/vnuml/“.

VNUML Version 1.6 verwendet kein „ext2“-Dateisystem für die zweite Partition sondern „iso9660“.

```
<fs>            <mntpoint>      <type>    <opts>              <dump/pass>
/dev/ubd/1      /mnt/vnuml      iso9660   defaults                  0 0
```

Listing 7.10: Konfiguration der zweiten Partition in „/etc/fstab“ für VNUML 1.6

Statt des Links auf das Verzeichnis „/mnt/vnuml/“ verlangt VNUML 1.6 ein eigenständiges Verzeichnis „/root/.ssh“,da in die zweite Partition aufgrund des ISO-Formats nun auch nicht mehr geschrieben werden kann.

```
/ # mkdir /root/.ssh/
```

Befehl 7.13: Konfiguration des „.ssh“-Verzeichnisses für VNUML 1.6

Ansonsten müssen keine weiteren Veränderungen für die Verwendung des Root-Dateisystems mit VNUML 1.6 durchgeführt werden.

Neuere Linux-Kernel haben keine Unterstützung mehr für das Geräte-Management „devfs“, sondern nur noch für das dynamische Geräte-Management „udev“. Um neuere Linux-Kernel verwenden zu können muss mit „udev“ statt „devfs“ gearbeitet werden. Dafür muss entweder die Datei „/etc/fstab“ direkt verändert werden, so dass nicht die Gerätedateien „/dev/ubd/0“ und „/dev/ubd/1“, sondern stattdessen „/dev/ubda“ und „/dev/ubdb“ eingebunden werden, oder es können die Vorzüge des dynamischen Geräte-Managements „udev“ genutzt werden, um eine „udev“-Regel zu erstellen, die die Gerätedateien des älteren „devfs“ entsprechend generiert. Dafür muss in der Datei „/etc/udev/rules.d/50-udev.rules“ der folgende Eintrag gesetzt werden.

```
#uml udev
KERNEL==“ubda“, NAME=“%k“, SYMLINK=“ubd/0“
KERNEL==“ubdb“, NAME=“%k“, SYMLINK=“ubd/1“
```

Listing 7.11: Alternative Konfiguration für „udev“ über die „udev“-Regeldatei 50-udev.rules

Dieser Eintrag erspart Veränderungen in der Datei „/etc/fstab“, so dass letztlich am UML-Root-Dateisystem nichts geändert werden muss, wenn zwischen „devfs“- und „udev“-Kerneln gewechselt wird. Gentoo-Linux verfügt zusätzlich über eine Variable in der Datei „/etc/conf.d/rc“, die mit „devfs“, „udev“ oder „auto“ das verwendete Geräte-Management bestimmt und daher auf „auto“ stehen sollte.

Trotzdem muss bislang immer bei einem Wechsel von einem „devfs“-Kernel auf einen „udev“-Kernel im Quelltext der VNUML-Parser-Datei „vnumlparser.pl“ die Stelle „root=/dev/ubd/0“ für „devfs“ zu „root=/dev/ubda“ für „udev“ verändert werden, damit VNUML mit den jeweiligen Kerneln umgehen kann. Dies ist ohne Probleme möglich da „vnumlparser.pl“ ein Perl-Script-Programm ist und lediglich interpretiert wird.
Neben diesen Einstellungen wird auch die „udev“-Software selbst benötigt, die im Gentoo-System allerdings bereits installiert ist.

7.2.3 Installation benötigter Software

Software wird in Gentoo mittels des Programms „emerge“ in das System installiert und aktualisiert. Wichtig ist dabei der Zugriff auf den „Portage“-Verzeichnisbaum. Mit dem Befehl „emerge --help“ werden alle Befehlsoptionen angezeigt. Über den Befehl „man emerge“ wird die Manual-Page angezeigt, die ausführliche Erklärungen über die Möglichkeiten von „emerge“ bietet.

Die folgenden Optionen von „emerge“ sind weitestgehend die wichtigsten:

- emerge --sync - aktualisiert den „Portage-Tree“ und die „ebuild“-Scripte
- emerge „Softwarepaket“ - installiert ein Softwarepaket inklusive aller abhängigen Softwarepakete
- emerge -s „Suchwort“ - Zum suchen nach Paketen, in deren Namen das Suchwort enthalten ist
- emerge -S „Suchwort“ - Zum suchen nach Paketen, in deren Beschreibung das Suchwort ist
- emerge -p „Softwarepaket“ - Installiert nichts, zeigt lediglich die von dem Softwarepaket abhängigen Pakete an, bzw. was installiert werden würde, wenn -p nicht angegeben wäre
- emerge -pv „Softwarepaket“ - Installiert nichts und zeigt neben den Abhängigkeiten auch die möglichen USE-Flags der Softwarepakete an
- emerge -O „Softwarepaket“ - versucht nur das angegebene Softwarepaket zu installieren, ohne die abhängigen Softwarepakete
- emerge -f „Softwarepaket“ - lädt lediglich die Quelltexte des angegebenen und der abhängigen Softwarepakete aus dem Internet, ohne die Installation durchzuführen
- emerge -u „Softwarepaket“ - aktualisiert die installierte Software inklusive der Software der abhängigen Pakete
- emerge --info - zeigt Informationen bzgl. der Installationsroutine an

- emerge --unmerge „Softwarepaket“ - deinstalliert die Software des angegebenen Paketes

Die Konfiguration der Installationsroutine „Portage“ wird über die Datei „/etc/make.conf“ eingestellt. Bei einem Stage3-System muss hier allerdings nichts verändert werden. Über diese Datei sind zum Beispiel die Compiler-Flags der für das Kompilieren verwendeten „GNU Compiler Collection“ einstellbar, um die Software optimal an die Systemumgebung anzupassen. Über die Anweisung „USE“ sind auch „globale“ USE-Flags setzbar, die dann für alle Softwarepakete gelten. So werden mit dem folgenden „USE“ Eintrag alle Pakete, die diese USE-Flags Unterstützen, ohne grafische Ausgabe aber mit SNMP, SMUX, SSL und IPv6 installiert.

```
USE="-X snmp smux ssl ipv6"
```

Listing 7.12: globale Einstellung der USE-Flags über die Datei „/etc/make.conf“

Über die Datei „make.conf“ können auch die Gentoo-Internet-Server eingestellt werden, von denen die Softwarepakete dann heruntergeladen werden. Standardmäßig ist zwar eine Verbindung zu einem Server gegeben, diese kann jedoch zu langsam sein oder fehlschlagen. Über die Variable „GENTOO_MIRRORS“ kann eine Liste von erreichbaren Server-Adressen angegeben werden, von denen dann die Softwarepakete für die Installation heruntergeladen werden. Mit dem Programm „mirrorselect“, das allerdings erst noch installiert werden muss, kann die Verbindung zu zahlreichen Gentoo-Servern getestet werden und die Server mit den besten Verbindungen können ausgewählt werden.

```
GENTOO_MIRRORS="ftp://ftp.uni-koblenz.de/pub/mirrors/gentoo/"
```

Listing 7.13: Einstellen von Gentoo-Servern für den Software-Download

Mit dem Befehl „man make.conf“ wird die Manual-Page der „make.conf“ Datei angezeigt, die die zahlreichen Konfigurationsanweisungen der Datei erläutert. In der Datei „/etc/make.conf.example“ sind Beispiele für die Konfiguration der Datei zu finden.

Die USE-Flags können auch individuell für jedes Softwarepaket „lokal“ angegeben werden. Dafür muss einfach nur die „USE“-Anweisung dem „emerge“ Befehl voran gestellt werden. Über die Befehlsoption -pv des „emerge“ Befehls können die verwendbaren USE-Flags eines Softwarepaketes angezeigt werden.
Die Erläuterungen für „globale“ USE-Flags sind in der Datei „/usr/portage/profile/use.desc“ aufgeführt. Erläuterungen für „lokale“, paketspezifische USE-Flags sind im selben Verzeichnis in der Datei „use.local.desc“ aufgelistet.

So finden sich in dieser Datei für das Softwarepaket „net-snmp“ die folgenden Einträge.

```
net-analyzer/net-snmp:diskio - Enable the use of diskio mibs
net-analyzer/net-snmp:elf - Enable the use of elf utils to check uptime on some
sytems
net-analyzer/net-snmp:mfd-rewrites - Use MFD rewrites of mib modules where
available
net-analyzer/net-snmp:rpm - Enable the rpm snmp probing
net-analyzer/net-snmp:smux - Enable the smux MIBS module
```

Listing 7.14: USE-Flags zu net-snmp erläutert in der Datei „/usr/portage/profiles/use.local.desc“

Die hier aufgelisteten USE-Flags müssen aber nicht unbedingt für das aktuelle Softwarepaket gültig sein.

Nach dem Softwarepaket „net-snmp“ oder „snmp“ im allgemeinen kann über „emerge -s“ oder „emerge -S“ gesucht werden.

```
/ # emerge -s net-snmp
*  net-analyzer/net-snmp
     Latest version available: 5.2.1.2-r1
     Latest version installed: [ Not Installed ]
     Size of files: 3,779 kB
     Homepage:      http://net-snmp.sourceforge.net/
     Description:   Software for generating and retrieving SNMP data
     License:       as-is BSD
```

Befehl 7.14: Suche nach der Software „Net-SNMP“

Gentoo-Linux kennt bei der automatischen Installation über den Befehl „emerge“ verschiedene „Software-Zweige“ die über die Variable „ACCEPT_KEYWORDS“ eingestellt werden. Standardmäßig ist hier der „stabile“ Zweig eingestellt. Für „x86“-Architekturen, wie die hier verwendete, wäre das ACCEPT_KEYWORDS=“x86“. Sollen neuere Softwarepakete aus dem dazugehörigen Test-Zweig verwendet werden so kann dem Befehl „emerge“ die Variable ACCEPT_KEYWORDS=“~x86“ vorangestellt werden.

```
/ # ACCEPT_KEYWORDS=“~x86“ emerge -s net-snmp
*  net-analyzer/net-snmp
     Latest version available: 5.2.2-r3
     Latest version installed: [ Not Installed ]
     Size of files: 3,827 kB
     Homepage:      http://net-snmp.sourceforge.net/
     Description:   Software for generating and retrieving SNMP data
     License:       as-is BSD
```

Befehl 7.15: Suche nach der aktuellen „Net-SNMP“-Software aus dem Test-Zweig

Mittels des Befehls „emerge --sync“ kann der „Portage-Tree“ aktualisiert werden, um neue „ebuild“-Scripte mit neuen Software-Versionen zu erhalten. Dies nimmt jedoch einige Zeit in Anspruch und ist aufgrund des aktuellen Verzeichnisbaums vorerst nicht notwendig.
Über den Befehl „emerge -u system“ kann die komplette Systemsoftware aktualisiert werden. Über den Befehl „emerge -u world“ wird jede installierte Software, soweit möglich, aktualisiert. Alle diese Vorgänge nehmen jedoch sehr viel zeit in Anspruch und sollten vorerst nicht ausgeführt werden. Die zusätzliche Optionsangabe -pv zeigt das komplette Ausmaß eines solchen Vorgehens an, ohne etwas zu installieren.

Um das Root-Dateisystem nun zu komplettieren muss nur noch der „devfs“-Daemon installiert werden. Das aktuelle Gentoo-Linux Basis-System ist nur noch mit „udev“ ausgestattet und würde deswegen nicht mit den derzeitigen VNUML Versionen und Kerneln ohne „devfs“-Unterstützung funktionieren. Mittels des Befehls „emerge“ muss das Softwarepaket „devfsd“ installiert werden.

```
/ # emerge devfsd
```

Befehl 7.16: Installation des „devfs“-Daemon

Von nun an ist das UML-Root-Dateisystem einsatzfähig und kann, wie im Kapitel 7.2.4 beschrieben, in eine Imagedatei abgelegt und schließlich mit einem UML-Kernel zusammen, auch in VNUML, verwendet werden.
Im folgenden werden noch einige Beispiele für die Installation von Softwareprogrammen aufgeführt, die Teilweise auch in dem UML-Root-Dateisystem der VNUML-Autoren zu finden sind.

Zuerst wird hier jedoch die Installation von SNMP zusammen mit Quagga und der SMUX-Unterstützung aufgezeigt.
Über den folgenden Befehl können die dafür benötigten Softwarepakete zusammen mit den möglichen USE-Flags angezeigt werden. Hier werden die Optionen der aktuelleren Version aus dem Test-Zweig angezeigt.

```
/ # ACCEPT_KEYWORDS="~x86" emerge -pv net-snmp quagga

These are the packages that would be merged, in order:

Calculating dependencies... done!
[ebuild  N    ] net-analyzer/net-snmp-5.2.2-r3  USE="ipv6 perl smux ssl tcpd -X -doc -elf -lm_sensors -mfd-rewrites -minimal -rpm" 3,827 kB
[ebuild  N    ] net-misc/quagga-0.99.4  USE="ipv6 pam snmp -bgpclassless -multipath -ospfapi -realms -tcp-zebra -tcpmd5" 2,173 kB

Total size of downloads: 6,001 kB
```

Befehl 7.17: Anzeigen der Installationsattribute für Net-SNMP und Quagga

Je nachdem wie viele Abhängigkeiten noch nicht erfüllt sind, kann die Ausgabe hier variieren. Die Software kann dann zum Beispiel mit den folgenden Einstellungen installiert werden.

```
/ # ACCEPT_KEYWORDS="~x86" USE="mfd-rewrites tcp-zebra tcpmd5 -pam" emerge net-snmp quagga
```

Befehl 7.18: Installation von Net-SNMP und Quagga

Einige USE-Flags wurden bereits über die Datei „make.conf“ gesetzt, so dass auf deren Angabe hier verzichtet werden kann.

Net-SNMP kann durchaus auch mit seinem grafischen MIB-Browser „tkmib“, der hier wegen der Angabe des USE-Flags „-X“ außen vor bleibt, installiert werden. Aufgrund des mittlerweile modular aufgebauten X-Window-Systems „xorg-x11“ wird hierbei nur noch ein X-Client und nicht mehr der komplette X-Server und damit auch nicht mehr so viel Software installiert.
Die grafischen Ausgaben von Programmen können nicht in den UML-Rechnern selbst angezeigt werden aber aufgrund der Client/Server Architektur des X-Window-Systems kann ein grafisches Programm in den UML-Rechnern gestartet und über eine aktive Netzwerk-Verbindung auf dem Host-System angezeigt werden.
Dafür muss in den UML-Rechnern ein X-Client installiert sein und die Umgebungsvariable „DISPLAY“ gesetzt werden. Der X-Server des Host-Systems muss über die Programme „xhost“ oder „xauth“ entsprechend konfiguriert werden, damit er Netzwerk-Verbindungen annimmt. Alternativ dazu kann im Host auch ein „nested X-Server“ gestartet werden. Im folgenden wird im Bezug auf das „SimpleNet“ Beispiel aus Kapitel 5 zwischen dem UML-Rechner „uml1“ und dem Host eine solche Verbindung aufgebaut. Dafür wird auf dem Host ein „nested X-Server“ wie folgt eingerichtet.

```
host # Xnest -ac -geometry 1024x768+0+0 :1
```

Befehl 7.19: Einrichten eines „nested X-Servers“ auf dem Host-System

Hierbei öffnet sich ein Fenster von 1024x768 Pixeln Größe an der Position 0;0 auf dem Desktop des Host-Systems.
Innerhalb des UML-Rechner muss nun noch die „DISPLAY“ Umgebungsvariable gesetzt werden, die auf den X-Server des Hosts verweist. Anschließend kann dann ein Programm mit grafischer Ausgabe, zum Beispiel der Net-SNMP MIB-Browser „tkmib“, gestartet werden.

```
uml1 # export DISPLAY=“192.168.0.1:1“
uml1 # tkmib
```

Befehl 7.20: Einrichten des UML-Rechners für die grafische Ausgabe und Start von „tkmib“

Die grafische Benutzerschnittstelle des Net-SNMP MIB-Browsers „tkmib“ erscheint dann in dem zuvor geöffneten Fenster des „nested X-Servers“ auf dem Host.

Für den besseren Umgang mit den UML-Rechnern in Netzwerken und einer einfachen Diagnose-Möglichkeit der Rechner und des Netzes, kann unter anderem noch die folgende Software installiert werden.

```
/ # USE=“-gtk“ emerge traceroute tcpdump nmap syslog-ng vixie-cron netkit-telnetd
```

Befehl 7.21: Installation einiger Diagnose- und Hilfswerkzeuge

Das Softwarepaket „traceroute“ enthält das gleichnamige Programm mit dem die IP-Adress-Zwischenstationen in einem Netzwerk, die auf dem Weg zur IP-Adresse der Gegenstelle liegen, angezeigt werden können. Mit „nmap“ wird der gleichnamige Portscanner installiert mit dem offene Ports und die dahinter laufenden Server-Dienste erkannt werden können. Der Portscanner verfügt über eine grafische Benutzerschnittstelle auf „gtk“-Basis, die hier aus Platzgründen nicht installiert werden soll. Das Programm „syslog-ng“ ist ein Syslog-Server für die System-Protokollierung und „vixie-cron“ ist ein Cron-Daemon zur zeitbezogenen Jobsteuerung. Die beiden Dienst-Programme können auch in ein Runlevel für den automatischen Start gesetzt werden. Das Softwarepaket „netkit-telnetd“ enthält „Telnet“, das zum Beispiel für die Konfiguration der Quagga Routing-Dienste benötigt wird, wenn kein „vtysh“-Terminal verfügbar ist.

Im UML-Root-Dateisystem der VNUML-Autoren ist auch die Firewall „iptables“ und das Point-to-Point Protocol „ppp“ installiert, dessen Verwendung VNUML auch über die Szenario-Definition anbietet.
Diese beiden Pakete besitzen hier die Gentoo-Linux-Kernel-Sources in ihrer Abhängigkeits-Struktur. Innerhalb des UML-Root-Dateisystems ist jedoch kein Kernel-Quellcode vorhanden und wird auch nicht benötigt. Die beiden Programme möchten den Kernel-Quellcode der Gentoo-Distribution aber mit installieren, was über die Option -O des Befehls „emerge“ verhindert werden sollte.

```
/ # emerge -O iptables ppp
```

Befehl 7.22: Installation von „iptables“ und „ppp“

Des weiteren verfügt das Debian-System der VNUML-Autoren noch über zahlreiche weitere Programme, wie zum Beispiel den Web-Server „Apache“, den DNS-Server „Bind“ und einen DHCP-Server. Diese Programme können natürlich ebenfalls über „Portage“ in das lokale System installiert werden. Je nachdem wie das Gentoo-System später eingesetzt werden soll.

Ab und zu werden auch Konfigurationsdateien des Gentoo-Systems mit der Installation oder Aktualisierung eines Softwarepaketes erneuert. Da in diesen Dateien auch individuelle Konfigurationen eingestellt sein können, die die Systemstabilität eventuell beeinflussen, bietet Gentoo-Linux die System-Werkzeuge „etc-update“, oder alternativ dazu „dispatch.conf“, an um zwischen dem Fortbestehen der vorhandenen Konfigurationsdatei oder dem Ersetzen durch die update Version zu entscheiden. Beide Programme bieten eine umfassende Erläuterung über ihre Manual-Page.

Anstatt Softwarepakete über den Befehl „emerge“ in einem Schritt zu installieren, können über den Befehl „ebuild“ auch nur die einzelnen Teilschritte durchgeführt werden. So lädt der folgende Befehl die Software „net-snmp“ von einem Server aus dem Internet herunter und entpackt sie lediglich im Verzeichnis „/var/tmp/portage/“ ohne sie zu kompilieren und zu installieren.

```
/ # ebuild /usr/portage/net-analyzer/net-snmp/net-snmp-5.2.2-r3.ebuild unpack
```

Befehl 7.23: Herunterladen und Entpacken der Software Net-SNMP über das Gentoo-Programm „ebuild“

Somit kann die Software auch manuell aus dem Quellcode kompiliert und installiert werden, um Einstellungen zu konfigurieren, die auch „Portage“ nicht bietet. Wenn die Software zuvor über „Portage“ ohne Probleme automatisch installiert wurde, sollte eine manuelle Installation ebenfalls ohne Probleme durchführbar sein.

Ist die benötigte Software installiert, kann die „chroot“-Umgebung über den Befehl „exit“ wieder verlassen werden.

7.2.4 Erstellen der Imagedatei

Hier wird nun die Imagedatei erstellt, in die das zuvor konfigurierte UML-Root-Dateisystem hinein kopiert werden muss, um in VNUML verwendet werden zu können. Das System hätte auch direkt in der Imagedatei erstellt werden können. Die Imagedatei hat jedoch eine feste Größe und es ist Anfangs unbekannt wie groß das UML-Root-Dateisystem letztlich wird, weswegen die Imagedatei höchstwahrscheinlich entweder viel zu groß oder zu klein erstellt worden wäre.

Nachdem verlassen der „chroot“-Umgebung sollten zuerst die eingebundenen Verzeichnisse gelöst werden.

```
host # umount gentoo_backup/usr/portage
host # umount gentoo_backup/proc/
```

Befehl 7.24: Lösen der eingebunden Verzeichnisse aus dem Gentoo-Systems

Sofern sich kein benötigter Software-Quellcode im Verzeichnis „/var/tmp/portage/“ befindet, kann der Inhalt des Verzeichnisses gelöscht werden um weiteren Speicherplatz zu sparen.
Was nun noch fehlt sind die Module des Kernels, die im Verzeichnis „gentoo_backup/lib/modules/“ abgelegt sein müssen. Dort müssen sie sich in einem Verzeichnis befinden, dessen Name der Bezeichnung des verwendeten UML-Kernels entspricht.
Die VNUML-Autoren liefern ihre Kernel-Module zusammen mit dem Kernel, gepackt in einem Tarball, so dass auch der UML-Kernel der VNUML-Autoren verwendet werden kann, nachdem die Kernel-Module innerhalb des „lib/modules/“ Verzeichnisses entpackt wurden.

Die Imagedatei kann nun erstellt werden. Diese besitzt eine feste Größe, die bei der Erstellung festgelegt werden muss. Um die benötigte Größe zu bestimmen kann mit dem Programm „du“ der belegte Speicherplatz des UML-Root-Dateisystems im Verzeichnis „gentoo_backup/“ festgestellt werden.

```
host # du -hs gentoo_backup/
622M  gentoo_backup/
```

Befehl 7.25: Bestimmen der benötigten Größe für die Imagedatei

Die Größe der Imagedatei sollte die Größe des belegten Speicherplatzes um einiges überschreiten, damit es im laufenden Betrieb nicht zu Fehlern kommt. Mit dem Programm „dd“ (Disk Dump) wird die Imagedatei erstellt und anschließend mit dem „ext2“-Dateisystem formatiert.

```
host # dd if=/dev/zero of=gentoo_root_fs bs=1M count=750
host # mke2fs -F gentoo_root_fs
```

Befehl 7.26: Erstellen und Formatieren der Imagedatei

Hier hat die Imagedatei den Namen „gentoo_root_fs“ und eine Größe von 750 MB. Diese 750 MB werden hier nicht über die „seek“ Option des Befehls „dd“ nur reserviert, sondern gleich im Host belegt. Die Festlegung der Größe über die Option „seek“ ist allerdings auch möglich.

Während der Ausarbeitung wurde festgestellt, dass „Journalingdateisysteme“ wie „ext3“ oder „reiserfs“, eher nicht für die Formatierung geeignet sind, da sie zum einen mehr Speicherplatz in den Imagedateien benötigen und zum anderen auch mehr Arbeitsspeicher für den Start des UML-Rechners vom Host verlangen. Ihnen muss über die Befehlsoption „mem“ oder über das Tag <mem> der VNUML-Sprache in der Szenario-Definition etwas Speicher zugewiesen werden damit sie einwandfrei funktionieren und der UML-Rechner fehlerfrei startet. Da aufgrund des COW-Mechanismus ohnehin nichts in diese original Imagedatei geschrieben wird sollte die Konsistenz des Dateisystems auch mit „ext2“ lange erhalten bleiben. Sollen hier trotzdem andere Dateisysteme als „ext2“ verwendet werden muss auch die Datei „/etc/fstab“ entsprechend eingerichtet sein.

Nun kann die Imagedatei in die Verzeichnis-Struktur des Hosts eingebunden und die Dateien des UML-Root-Dateisystems aus dem Backup-Verzeichnis hinüber kopiert werden, was eine längere Zeit in Anspruch nimmt.

```
host # mkdir mntpoint
host # mount -o loop gentoo_root_fs mntpoint/
host # cp -Rdipv gentoo_backup/*  mntpoint/
host # umount mntpoint
```

Befehl 7.27: Kopieren der Daten in die Imagedatei

Anschließend muss die Imagedatei mit dem Befehl „umount“ wieder aus der Verzeichnisstruktur des Hosts gelöst werden, um Probleme während des Startens des UML-Rechners zu vermeiden.

Zu guter Letzt können noch die eher lästigen Konsistenzabfragen des „ext2“-Dateisystems mit dem Programm „tune2fs“ abgeschaltet werden.
Es sollte vorher allerdings trotzdem ein Konsistenzcheck mit dem Programm „fsck.ext2“ durchgeführt werden.

```
host # fsck.ext2 -p gentoo_root_fs
host # tune2fs -i 0 -c 0 gentoo_root_fs
tune2fs 1.38 (30-Jun-2005)
Setting maximal mount count to -1
Setting interval between checks to 0 seconds
```

Befehl 7.28: Abschalten des Konsistenzchecks des UML-Root-Dateisystems

Somit wäre das UML-Root-Dateisystem auf Basis von Gentoo-Linux fertig und zusammen mit einem UML-Kernel in VNUML Netzwerk-Simulationen einsetzbar.

7.3 Erstellen eines UML-Kernels

Im Laufe der Arbeit mit VNUML kam es immer wieder zu Problemen mit den X-Terminals der UML-Rechner. Stellenweise wurden X-Terminals einzelner UML-Rechner einer VNUML Netzwerk-Simulation nicht geöffnet, schlossen sich nach kurzem erscheinen wieder oder blieben ohne jeglichen Inhalt. Durch Veränderungen in der Konfiguration des UML-Kernels konnten solche Probleme stellenweise behoben werden.
Im folgenden wird nun erläutert wie aus dem Linux-Kernel-Quellcode ein UML-Kernel für den Einsatz in VNUML erstellt werden kann.
Zunächst wird der Linux-Kernel-Quellcode benötigt. Hier ist darauf zu achten, dass der verwendete Kernel „devfs“ unterstützt, was bei den offiziellen Vanilla-Kerneln ab der Version größer gleich 2.6.13 nicht mehr der Fall ist. Hier wird nun der Vanilla-Kernel in der Version 2.6.12.6 von „http://www.kernel.org“ verwendet.

```
host # cd downloads
host downloads # wget ftp://ftp.kernel.org/pub/linux/kernel/v2.6/linux-2.6.12.6.tar.bz2
host downloads # mkdir ../kernel
host downloads # tar xfpjv linux-2.6.12.6.tar.bz2 -C ../kernel/
host downloads # cd ../kernel/linux-2.6.12.6/
```

Befehl 7.29: Herunterladen und Entpacken des Linux-Kernel-Quellcodes

Nachdem entpacken des Tarballs kann mit der Konfiguration des Kernels direkt begonnen werden. Wichtig dabei ist, dass allen Kernel-Befehlen die Option „ARCH=um“ angehängt wird.

Der Linux-Kernel besitzt eine eingestellte Standardkonfiguration. Soll an der Kernel-Konfiguration der VNUML-Autoren weitestgehend festgehalten werden, so können deren Einstellungen in den neuen Kernel übernommen werden, auch wenn die Kernel-Version nicht die gleiche ist. Die VNUML-Autoren legen ihren UML-Kerneln die Konfigurationsdatei bei, deren Einstellungen wie folgt in einen neuen Kernel übernommen werden können. Dabei muss die Konfigurationsdatei im Quellcode-Verzeichnis des Linux-Kernels mit dem Namen „.config“ abgelegt sein.

```
host linux-2.6.12.6 # cp config-VNUML-Kernel .config
host linux-2.6.12.6 # make oldconfig ARCH=um
```

Befehl 7.30: Übernehmen der VNUML-Kernel Konfigurationseinstellungen

Ist die Version der Konfigurationsdatei kleiner als die verwendete Kernelversion, so kann es vorkommen, dass nach bestimmten Einstellungen gefragt wird, die in der Konfigurationsdatei nicht definiert sind. Hier gibt es jedoch stets eine Standard-Einstellung, die dann einfach bestätigt werden kann.
Anschließend kann mit der individuellen Konfiguration des Kernels fortgefahren werden. Über die Option „menuconfig“ wird ein einfaches Konsolenmenu für die Konfiguration des UML-Kernels geöffnet

```
host linux-2.6.12.6 # make menuconfig ARCH=um
```

Befehl 7.31: Konfigurieren des UML-Kernels

Ein wesentlich übersichtlicheres grafisches Menü, dessen erscheinen aber abhängig von den Einstellungen des Host-Systems ist, kann über die Optionen „xconfig“ oder „gconfig“, anstelle von „menuconfig“, geöffnet werden. Hier werden dann in einem Teilbereich des grafischen Fensters auch direkt Erläuterungen zu den einzelnen Kernel-Konfigurationen angezeigt.
Ist der UML-Kernel fertig konfiguriert kann er wie folgt kompiliert werden, was eine recht lange Zeit in Anspruch nehmen sollte.

```
host linux-2.6.12.6 # make ARCH=um
```

Befehl 7.32: Kompilieren des UML-Kernels

Der fertige UML-Kernel liegt dann als Anwendungsprogramm im Kernel-Quellcode-Verzeichnis, meist unter der Bezeichnung „linux“ oder „vmlinux“.
Nun müssen noch die Kernel-Module in das UML-Root-Dateisystem kopiert werden. Dafür muss nur das Verzeichnis angegeben werden, in dem sich das Wurzelverzeichnis („/“) des Root-Dateisystems befindet. Die Kernel-Module werden dann automatisch in das dortige Unterverzeichnis „lib/modules/'Kernel'“ kopiert.

```
host linux-2.6.12.6 # mount -o loop ../../gentoo_root_fs ../../mntpoint
host linux-2.6.12.6 # make modules_install INSTALL_MOD_PATH=../../mntpoint/ ARCH=um
host linux-2.6.12.6 # umount ../../mntpoint
host linux-2.6.12.6 # cp vmlinux ../../
```

Befehl 7.33: Installation der UML-Kernel-Module

Damit wäre der UML-Kernel fertig „gebaut“ und kann zusammen mit dem UML-Root-Dateisystem verwendet werden.

7.4 Starten des UML-Rechners

Mit dem UML-Kernel und dem UML-Root-Dateisystem kann nun schon die Funktion des UML-Rechners, auch außerhalb von VNUML, getestet werden.

Mit den folgenden Befehlen wird eine virtuelle Netzwerkschnittstelle im Host-System erstellt. Anschließend wird ein UML-Rechner gestartet, der diese Netzwerkschnittstelle verwenden kann.

```
host # tunctl -t tap0 -f /dev/net/tun
host # ./vmlinux ubd0=cow_fs,gentoo_root  eth0=tuntap,tap0,fd:fe:0:0:1:1,192.168.0.254 umid=gentooUML
```

Befehl 7.34: Starten eines einzelnen UML-Rechners

Beim Start kommt es zwar zu einer Fehlermeldung aufgrund der fehlenden zweiten Partition. Dies beeinflusst die Funktion des UML-Rechners allerdings nicht.

Informationen zur weiteren Vorgehensweise finden sich im Kapitel 5.6, in den Befehlen 5.15 und 5.16.

Soll in einen laufenden Gentoo-UML-Rechner neue Software installiert werden so wird, neben der Internetverbindung, auch der „Portage-Tree“ benötigt. Dieser wurde zu Beginn der Anleitung in eine Imagedatei gespeichert, die auch in den gestarteten UML-Rechner gemountet werden kann. Dafür wird das UML-Management-Konsolen-Programm „uml_mconsole“ der „Usermode-Utilities“ auf dem Host benötigt. Über die Option „umid“ wird beim Start des UML-Rechners die Konsolen-Schnittstelle angegeben, in die vom Host aus eingeloggt werden kann. Über die Management-Konsole muss dann die Imagedatei als Gerätedatei in dem UML-Rechner angelegt werden.
Hier wird die Imagedatei „portage.img“ als Gerätedatei „/dev/ubd/2“ angelegt.

```
host # uml_mconsole gentooUML
(gen_con) config ubd2=portage.img
OK
```

Befehl 7.35: Anlegen einer Imagedatei als Gerätedatei im laufenden UML-Rechner

Anschließend kann die neue Gerätedatei über den Befehl „mount“, innerhalb des UML-Rechners in dessen Verzeichnisstruktur eingebunden werden.

```
uml # mount /dev/ubd/2 /usr/portage
```

Befehl 7.36: Einbinden der neuen Gerätedatei in die Verzeichnisstruktur des UML-Rechner

VNUML startet die UML-Rechner für die Netzwerk-Simulationen ebenfalls mit einer aktiven Management-Konsolen-Schnittstelle, so dass letztlich auch während einer VNUML-Simulation die Installation von Software mit Hilfe der Installationsroutine „Portage“ möglich wäre.

7.5 Zusammenfassung

Die Anleitung zum Erstellen eines UML-Rechners für VNUML auf Basis von Gentoo-Linux verdeutlicht noch einmal die enorme Freiheit und Mächtigkeit, die das Simulationsprogramm VNUML letztlich bietet. Über die Gentoo Installationsroutine „Portage“ stehen, wie über Debians „APT“, eine Fülle von Linux-Programmen zur freien Verfügung, die ohne größeren Aufwand und Hintergrundwissen direkt in das System installiert und verwendet werden können. Gentoo-Linux bietet mit „Portage“ eine sehr flexible Installationsroutine, die eine enorme Vielfalt von Konfigurationsmöglichkeiten für die benötigte Software eröffnet. Darüber hinaus ist es eine weitverbreitete Linux-Distribution mit einer sehr großen Benutzer-Gemeinde, die über zahlreiche „Tutorials“ und „Workarounds“ für aktuelle Programme und deren Probleme verfügt.
Ein Großteil der im Umlauf befindlichen Linux-Distribution sollte ebenfalls als Basis eines UML-Root-Dateisystems für VNUML Netzwerk-Simulationen verwendet werden können. Somit kann mit VNUML auch ein heterogenes Netzwerk mit Linux-Systemen unterschiedlicher „Hersteller“ (Distributionen) aufgebaut und deren zusammenwirken getestet werden.

Abbildungsverzeichnis

Abb. 3.1: Die Funktionsbereiche des Netzwerkmanagements 9

Abb. 4.1: Das SNMP Manager/Agenten Modell 12
Abb. 4.2: Historische Entwicklung von SNMP [9] intro-standards.pdf 13
Abb. 4.3: Standardisierungsprozess der IETF [9] intro-standards.pdf 14
Abb. 4.4: Netzwerkmanagement-Architektur des SNMP-Modells [7] 17
Abb. 4.5: SNMP-Architektur in TCP/IP-Netzen 18
Abb. 4.6: Architektur des SNMPv3-Managers [9] snmpv3.pdf 19
Abb. 4.7: Architektur des SNMPv3-Agenten [9] snmpv3.pdf 20
Abb. 4.8: Ausschnitt des Objekt-Registrierungsbaumes der ISO 22
Abb. 4.9: Struktur mit BER kodierter Daten 24
Abb. 4.10: Beispiele für die Kodierung von ASN.1-Datentypen mit BER [4] S. 675 24
Abb. 4.11: Beispiel für die Kodierung eines komplexeren ASN.1 Datensatzes mit BER 25
Abb. 4.12: Baum-Ausschnitt der Gruppe „interfaces“ 31
Abb. 4.13: Ablauf einer GetRequest Kommunikation [9] snmpv2.pdf 33
Abb. 4.14: Ablauf einer GetNextRequest Kommunikation [9] snmpv2.pdf 34
Abb. 4.15: Ablauf einer GetBulkRequest Kommunikation [9] snmpv2.pdf 34
Abb. 4.16: Ablauf einer SetRequest Kommunikation [9] snmpv2.pdf 34
Abb. 4.17: Ablauf einer Trap Meldung [9] snmpv2.pdf 35
Abb. 4.18: Ablauf der InformRequest Meldung [9] snmpv2.pdf 35
Abb. 4.19: Aufbau der SNMP-PDU 36
Abb. 4.20: Aufbau der SNMPv1 und -v2c Nachrichten 36
Abb. 4.21: Aufbau der SNMPv3 Nachrichten [8] snmpv3.pdf 36
Abb. 4.22: Entschlüsselung einer SNMPv1 Response Nachricht [6] S. 272 37
Abb. 4.23: Master/Sub-Agenten Architektur 41

Abb. 5.1: vnumlgui, ein grafisches Benutzerinterface zu VNUML 46
Abb. 5.2: Topologie des Netzwerk-Szenarios „SimpleNet“ 47
Abb. 5.3: Komplette virtuelle Netzwerktopologie der VNUML Simulation des „SimpleNet“-Szenarios 50
Abb. 5.4: allgemeiner Aufbau eines Linux-Betriebssystems 52
Abb. 5.5: Der Copy On Write (COW) Mechanismus 55
Abb. 5.6: Mehrere COW-Imagedateien über einer Basis-Imagedatei 55
Abb. 5.7: Manuell erstellter Teil des „SimpleNet“-Szenarios 56
Abb. 5.8: Verteilung der Prozesse des virtuellen Netzwerks im Host-System 59
Abb. 5.9: Netz-Topologie der UML-Internet-Verbindung 61
Abb. 5.10: Einige Quagga Routing Dienste 63

Abb. 6.1: Möglichkeiten des Einsatzes:
SNMP als Erweiterung des VNUML-Managements oder in isolierter Simulations-Umgebung 72
Abb. 6.2: MIB-Modul „studienarbeit“ im ISO-Registrierungsbaum 93
Abb. 6.3: Ablauf der Set-Befehl Verarbeitung 96

Tabellenverzeichnis

Tabelle 4.1: Einige wichtige RFC-Dokumente zu SNMP 16
Tabelle 4.2: Lexikalische Konventionen von SMI 21
Tabelle 4.3: Basis-Datentypen, die SNMP verwenden kann 21
Tabelle 4.4: BER-Kodierungsnummern einiger universeller Datentypen 24
Tabelle 4.5: Einige SMIv2 Datentypen für SNMP Management-Objekte 26
Tabelle 4.6: Einige SMIv2 Textkonventionen (Entnommen RFC 2579) 27
Tabelle 4.7: Als Liste dargestellter Ausschnitt der Gruppe „interfaces“ 31
Tabelle 4.8: Einige Gruppen der MIB-II 32
Tabelle 4.9: Die in SNMPv1 definierten Trap-Nachrichten 35
Tabelle 4.10: „error-status“ Codes der GetResponse Nachricht [5] S. 373 36

Tabelle 6.1: Vergleich des Ressourcenverbrauchs der einzelnen SNMP-Versionen 104

Listingverzeichnis

Listing 4.1: Definition eines Integers in ASN.1 21
Listing 4.2: Definition eines wertebeschränkten Integer-Datentyps in ASN.1 21
Listing 4.3: Tagging von Datentypen 23
Listing 4.4: Definition der Textkonvention „RunState“ 27
Listing 4.5: Textuelles Gerüst der MIB-Dateien 28
Listing 4.6: Definition eines Objektes in SMIv2 28
Listing 4.7: Auszug aus der SNMPv2-MIB (Entnommen RFC 3418) 29
Listing 4.8: Auszug der Definition der Tabelle „ifTable“ (Entnommen RFC 2863 - The Interfaces Group) 30
Listing 4.9: Definition eines Notification-Objektes für SNMP-Meldungen 31

Listing 5.1.1: Inhalt der Datei SimpleNet.xml, Teil 1 48
Listing 5.1.2: Inhalt der Datei SimpleNet.xml, Teil 2 48
Listing 5.1.3: Inhalt der Datei SimpleNet.xml, Teil 3 49
Listing 5.2: Inhalt der Datei „/etc/fstab“ des UML-Root-Dateisystems bei Verwendung von „devfs“ 54
Listing 5.3: Inhalt der „zebra“-Konfigurationsdatei „zebra.conf“ 66
Listing 5.4: Inhalt der „ripd“-Konfigurationsdatei ripd.conf 67
Listing 5.5: Inhalt der Datei „snmpd.log“ nach Zustandekommen einer SMUX-Verbindung 68

Listing 6.1: Eintrag einer Quelltext-Downloadquelle für die Debian APT-Tools in die Datei „sources.list“ 72
Listing 6.2: Möglichkeiten der Objekt-Bezeichnung des Managers 76
Listing 6.3: Bash-Script „snmpQuery.sh“ mit eingebundenen Net-SNMP Anwendungen des Managers 79
Listing 6.4.1: Konfiguration der Zugriffsrechte in der Datei „snmptrapd.conf“ des Trap-Server 82
Listing 6.4.2: Konfiguration des Trap-Servers für das Verarbeiten von Meldungen 82
Listing 6.5.1: Konfiguration für den SNMPv1 / -v2c Zugriff in der Datei „snmpd.conf“ 83
Listing 6.5.2: Konfiguration für den SNMPv3 Benutzerzugriff in der Datei „snmpd.conf“ 83
Listing 6.5.3: Konfiguration von VACM Security-Namen in der Datei „snmpd.conf“ 84
Listing 6.5.4: Konfiguration von VACM Gruppen in der Datei „snmpd.conf“ 84
Listing 6.5.5: Konfiguration von VACM-Sichten in der Datei „snmpd.conf“ 84
Listing 6.5.6: Konfiguration von VACM Access Policies in der Datei „snmpd.conf“ 85
Listing 6.5.7: Konfiguration von Systeminformationen des Agenten in der Datei „snmpd.conf“ 85
Listing 6.5.8: Konfiguration der Ziele für SNMP-Meldungen in der Datei „snmpd.conf“ 86
Listing 6.5.9: Konfiguration der Überwachung von Prozessinstanzen in der Datei „snmpd.conf“ 86
Listing 6.5.10: Konfiguration der Überwachung der Speicherkapazität und Dateigröße 87
Listing 6.5.11: Konfiguration für das Senden von Standard-Ereignismeldungen 87
Listing 6.5.12: Konfiguration für das Senden einer Meldung bei fehlgeschlagener Authentifizierung 87
Listing 6.5.13: Konfiguration für das Senden von Ereignismeldungen für fehlende Prozessinstanzen 87
Listing 6.5.14: Konfiguration der Einbindung externer Programme über „exec“ 88
Listing 6.5.15: Konfiguration von extern verwalteten MIB-Bibliotheken über „pass“ und „pass_persist“ 89
Listing 6.5.16: Konfiguration von Quagga SMUX-Peers in der Datei „snmpd.conf“ 90
Listing 6.5.17: Konfiguration des AgentX Sub-Agenten in der Datei „snmpd.conf“ 91
Listing 6.6: Beispiel eines Script-Programms „testprogie.sh“ zum Einbinden in den Net-SNMP Agenten 88
Listing 6.7: Beispiel für ein durch „pass“ angesteuertes Bash-Script „vtyshQuery.sh“ 89
Listing 6.8: OID der Quagga beiligenden MIBs (MIB-Module) 90
Listing 6.9: Inhalt der MIB-Datei „STUDIENARBEIT-MIB.txt“ 93
Listing 6.10.1: Inhalt der Datei „studienarbeit.c“, Teil 1 95
Listing 6.10.2: Inhalt der Datei „studienarbeit.c“, Teil 2 95
Listing 6.10.3: Inhalt der Datei „studienarbeit.c“, Teil 3 95
Listing 6.10.4: Inhalt der Datei „studienarbeit.c“, Teil 4 96
Listing 6.10.5: Inhalt der Datei „studienarbeit.c“, Teil 5 97
Listing 6.11: Inhalt des Programm „ifTraffic.sh“ 101
Listing 6.12: Inhalt der Datei „ifTrafficAgent_eth1.csv“ 101
Listing 6.13: Inhalt der Script-Datei „zebraTest.sh“ 102
Listing 6.14: Definition des Notification-Objects „noZebra“ 102
Listing 6.15: Inhalt der Log-Datei des Trap-Servers nach Erhalt der „noZebra“ Meldung 103
Listing 6.16: Konfiguration der „snmptrapd.conf“ für den Empfang der Ereignismeldung „noZebra“ 104

Listing 6.17: Inhalt des Bash-Programms „zebra.sh“ 105
Listing 6.18: Inhalt der Datei „zebraOutput.txt“ 105

Listing 7.1: Konfiguration der Datei „/etc/fstab“ 109
Listing 7.2: Das Init-Script „vnumlboot“ für die Referenz aus dem Runlevel auf „umlboot“ 109
Listing 7.3: Konfiguration der Login-Terminals über die Datei „/etc/inittab“ 110
Listing 7.4: Konfiguration der Datei „/etc/ssh/sshd_config“ 111
Listing 7.5.1: Original Code der Datei „/etc/init.d/modules“ 111
Listing 7.5.2: Modifizierter Code der Datei „/etc/init.d/modules“ 111
Listing 7.6.1: Original Code der Datei „/etc/init.d/localmount“ 111
Listing 7.6.2: Modifizierter Code der Datei „/etc/init.d/localmount“ 111
Listing 7.7.1: Original Code der Datei „/etc/init.d/bootmisc“ 112
Listing 7.7.2: Modifizierter Code der Datei „/etc/init.d/bootmisc“ 112
Listing 7.8.1: Original Code der Datei „/etc/conf.d/rc“ 112
Listing 7.8.2: Modifizierter Code der Datei „/etc/conf.d/rc“ 112
Listing 7.9: Deaktivieren der „rp_filter“ Variable über die Datei „/etc/sysctl.conf“ 113
Listing 7.10: Konfiguration der zweiten Partition in „/etc/fstab“ für VNUML 1.6 113
Listing 7.11: Alternative Konfiguration für „udev“ über die „udev“-Regeldatei 50-udev.rules 113
Listing 7.12: globale Einstellung der USE-Flags über die Datei „/etc/make.conf“ 114
Listing 7.13: Einstellen von Gentoo-Servern für den Software-Download 114
Listing 7.14: USE-Flags zu net-snmp erläutert in der Datei „/usr/portage/profiles/use.local.desc“ 115

Befehlsverzeichnis

Befehl 4.1: SNMPv1 Anfrage, ausgeführt mit Net-SNMP 38

Befehl 5.1: Starten der Netzwerk-Simulation „SimpleNet“ 50
Befehl 5.2: Kopieren der benötigten Konfigurationsdateien vom Host auf die UML-Rechner 50
Befehl 5.3: Start der Dienste auf den UML-Rechnern 50
Befehl 5.4: Ausschnitt der Ausgabe des Simulations-Kommandos „mgmt“ 50
Befehl 5.5: Beenden einer Netzwerk-Simulation 51
Befehl 5.6: Abbruch einer Netzwerk-Simulation und Löschen aller Backup-Dateien 51
Befehl 5.7: Routenverfolgung von „uml1“ zu „uml3“, vom Host via „ssh“ aus initiiert 51
Befehl 5.8: Kopieren der Datei „ripd.conf“ auf den UML-Rechner „uml2“ mittels „scp“ 51
Befehl 5.9: Starten eines UML-Rechners 54
Befehl 5.10: Verschmelzen einer Basis-Imagedatei mit der zugehörigen COW-Imagedatei 55
Befehl 5.11.1 : Aufbau des virtuellen Netzes im Host 56
Befehl 5.11.2 : Konfiguration der Management-Netzwerkschnittstellen des Hosts 56
Befehl 5.11.3 : Hinzufügen der Netzwerkschnittstellen zur Bridge 57
Befehl 5.11.4: Erstellen der Imagedatei für das Konfigurations-Script des UML-Rechners „uml1“ 57
Befehl 5.11.5: Kopieren des SSH-Schlüssels 57
Befehl 5.11.6 : Erstellen der Script-Datei „umlboot“ für die Konfiguration des UML-Rechners „uml1“ 57
Befehl 5.11.7: Erstellen der Image- und Script-Datei für die Konfiguration des UML-Rechners „uml2“ 58
Befehl 5.11.8: Manuelles Starten der beiden UML-Rechner „uml1“ und „uml2“ 59
Befehl 5.12 : Verwendung der UML-Management-Konsole „uml_mconsole“ 58
Befehl 5.13: Vorbereitung der Imagedatei für die Software Installation 60
Befehl 5.14: Verlassen der „chroot“-Umgebung und lösen der Imagedatei 60
Befehl 5.15: Einrichten eines NAT-Routers auf dem Host für einen UML-Rechner 61
Befehl 5.16: Einrichten einer Internetverbindung für den UML-Rechner „uml1“ 61
Befehl 5.17: Vergrößern einer vorhanden Imagedatei 62
Befehl 5.18: Erstellen einer neuen Imagedatei für UML-Rechner 63
Befehl 5.19: Herunterladen und Entpacken des Quagga-Quellcodes 64
Befehl 5.20: Manuelle Installation von Quagga 64
Befehl 5.21: Lokalisierung der Quagga Routing-Dienstprogramme 64
Befehl 5.22: Überprüfen der Referenz auf die Net-SNMP-Bibliothek 65
Befehl 5.23: Aufspüren der MIB-Dateien der Quagga Routing-Dienste 65
Befehl 5.24: Start des Quagga Kernel-Routing-Table-Managers „zebra“ 65

Befehl 5.25: Start des Quagga RIP-Routers „ripd“ 65
Befehl 5.26: Einloggen in die „zebra“ -Terminal-Shell via „Telnet“ 66
Befehl 5.27: Wechseln in den „Enable Mode“ von „zebra“ 67
Befehl 5.28: Auflistung aller gültigen Anweisungen innerhalb „zebra“ 67
Befehl 5.29: Einloggen in die „ripd“-Terminal-Shell 67
Befehl 5.30: Einloggen in die „vtysh“-Terminal-Shell 68
Befehl 5.31: Anzeigen von RIP Routing-Informationen über „vtysh“ in der Konsole 68
Befehl 5.32: SNMP-Abfrage von Managementinformationen des RIP-Routers 69

Befehl 6.1: Download des Net-SNMP Quellcodes mittels Debians „apt-get“ 73
Befehl 6.2: Installation der Software, die Net-SNMP für seine Funktion benötigt 73
Befehl 6.3: Konfiguration für das Kompilieren von Net-SNMP 73
Befehl 6.4: Installation von Net-SNMP 74
Befehl 6.5: Allgemeines Verwendungs-Schema der Kommandozeilen-Programme des Managers 75
Befehl 6.6: Get-Anfrage mit Net-SNMPs „snmpget“ 76
Befehl 6.7: SNMPv3 Get-Anfrage mit Net-SNMPs „snmpget“ 77
Befehl 6.8: GetNext-Anfrage mit Net-SNMPs „snmpgetnext“ 77
Befehl 6.9: GetBulk Anfrage mit Net-SNMPs „snmpbulkget“ 77
Befehl 6.10: Set-Aufforderung mit Net-SNMPs „snmpset“ 77
Befehl 6.11: Auflistung aller verfügbaren Managementinformationen der UDP-MIB mit „snmpwalk“ 78
Befehl 6.12: Abfrage von Objekt-Definitionen mittels „snmptranslate“ 78
Befehl 6.13: Ausführung und Ausgabe des Bash-Programms „snmpQuery.sh“ 80
Befehl 6.14: Starten des Trap-Servers „snmptrapd“ mit Konsolenausgabe 81
Befehl 6.15: Starten des Trap-Servers „snmptrapd“ mit Konfigurations- und Log-Datei 81
Befehl 6.16: Erstellen einer Konfigurationsdatei für den Agenten mittels „snmpconf“ 82
Befehl 6.17: Starten des Net-SNMP Agenten 82
Befehl 6.18: Einrichten eines SNMPv3-Benutzers mit dem Programm „net-snmp-config“ 83
Befehl 6.19: „snmpwalk“-Ausgabe der VACM-Community „allpublic“ 85
Befehl 6.20: Abfrage der Prozessüberwachungstabelle „ucdavis.prTable“ 87
Befehl 6.21: Ausgabe des Bash-Scripts „testprogie.sh“ 88
Befehl 6.22: Anzeigen der Tabelle der durch „exec“ ausführbaren Programme auf dem Agenten 89
Befehl 6.23: SNMP-Abfrage der durch das Script „vtyshQuery.sh“ verwalteten Objekte 90
Befehl 6.24: Starten des Net-SNMP Agenten als AgentX Sub-Agent 91
Befehl 6.25: Versenden von SNMP-Meldungen 91
Befehl 6.26: Kompilieren eines AgentX Sub-Agenten für Net-SNMP 91
Befehl 6.27: Eingliederung der „STUDIENARBEIT-MIB“ in die Net-SNMP MIB-Umgebung 94
Befehl 6.28: Anzeigen der neuen Objekte mit „snmptranslate“ 94
Befehl 6.29: Erstellen der C-Dateien auf Basis der STUDIENARBEIT-MIB 94
Befehl 6.30: Einbinden der „STUDIENARBEIT-Objekte“ direkt in den Net-SNMP Agenten 98
Befehl 6.31: Kompilieren der „STUDIENARBEIT-Objekte“ in einen AgentX Sub-Agenten 98
Befehl 6.32: Starten des Sub-Agenten studieProgie 98
Befehl 6.33: Verwendungsbeispiel des Objektes „superUserCommand“ der „STUDIENARBEIT-MIB“ 98
Befehl 6.34: Überprüfen der Netzwerk-Routen mit dem Befehl „route“ 99
Befehl 6.35: Übergeben einer neuen IP-Route via SNMP über das Objekt „superUserCommand“ 99
Befehl 6.36: Auslesen der Routing-Tabelle des Agenten via SNMP 99
Befehl 6.37: Konfiguration der Netzwerkschnittstellen mit dem Befehl „sysctl“ 99
Befehl 6.38: Übermittlung des „sysctl“ Befehls an den Agenten via SNMP 100
Befehl 6.39: Abfrage IPv4-Weiterleitung über das „ipv4forwarding“ Objekt 100
Befehl 6.40: Setzen der IPv4-Weiterleitung via SNMP über die nummerischen Werte 100
Befehl 6.41: Start des Bash-Programms „ifTraffic.sh“ 101
Befehl 6.42: „snmptranslate“ über „studienarbeit“ 102

Befehl 7.1: Download des „Stage3-“ und des „Portage“-Tarballs 107
Befehl 7.2: Entpacken des „Stage3“-Tarballs 108
Befehl 7.3: Entpacken des „Portage“-Tarballs 108
Befehl 7.4: Kopieren der Datei „resolv.conf“ für den Zugriff auf Internet-Adressen 108
Befehl 7.5: Wechsel in die „chroot“-Umgebung in das Gentoo-Linux-System 108

Befehl 7.6: Setzen eines Passwortes für den UML-Rechner Benutzer „root“ 108
Befehl 7.7: Erstellen des „Mountpunktes“ für die zweite Partition und der SSH-Verlinkung 109
Befehl 7.8: Setzen des Init-Scripts „vnumlboot“ in den Default-Runlevel 110
Befehl 7.9: Setzen des Init-Scripts „sshd“ in den Default-Runlevel 110
Befehl 7.10: Erzeugen der lokalen SSH-Schlüssel für den UML-Rechner 110
Befehl 7.11: Anzeigen der Init-Scripte, die beim Hochfahren automatisch ausgeführt werden 112
Befehl 7.12: Entfernen einiger nicht benötigter Init-Scripte aus den Runleveln 112
Befehl 7.13: Konfiguration des „.ssh“-Verzeichnisses für VNUML 1.6 113
Befehl 7.14: Suche nach dem Softwarepaket „Net-SNMP“ 115
Befehl 7.15: Suche nach der aktuellen „Net-SNMP“-Software aus dem Test-Zweig 115
Befehl 7.16: Installation des „devfs“-Daemon 115
Befehl 7.17: Anzeigen der Installationsattribute für Net-SNMP und Quagga 116
Befehl 7.18: Installation von Net-SNMP und Quagga 116
Befehl 7.19: Einrichten eines „nested X-Servers“ auf dem Host-System 116
Befehl 7.20: Einrichten des UML-Rechners für die grafische Ausgabe und Start von „tkmib“ 116
Befehl 7.21: Installation einiger Diagnose- und Hilfswerkzeuge 116
Befehl 7.22: Installation von „iptables“ und „ppp“ 117
Befehl 7.23: Herunterladen und Entpacken der Software Net-SNMP über das Gentoo-Programm „ebuild“ 117
Befehl 7.24: Lösen der eingebunden Verzeichnisse aus dem Gentoo-Systems 117
Befehl 7.25: Bestimmen der benötigten Größe für die Imagedatei 118
Befehl 7.26: Erstellen und Formatieren der Imagedatei 118
Befehl 7.27: Kopieren der UML-Root-Dateisystem-Daten in die Imagedatei 118
Befehl 7.28: Abschalten des Konsistenzchecks des UML-Root-Dateisystems 118
Befehl 7.29: Herunterladen und Entpacken des Linux-Kernel-Quellcodes 119
Befehl 7.30: Übernehmen der VNUML-Kernel Konfigurationseinstellungen 119
Befehl 7.31: Konfigurieren des UML-Kernels 119
Befehl 7.32: Kompilieren des UML-Kernels 119
Befehl 7.33: Installation der UML-Kernel-Module 119
Befehl 7.34: Starten eines einzelnen UML-Rechners 120
Befehl 7.35: Anlegen einer Imagedatei als Gerätedatei im laufenden UML-Rechner 120
Befehl 7.36: Einbinden der neuen Gerätedatei in die Verzeichnisstruktur des UML-Rechner 120

Literaturverzeichnis

[1] „*VNUML - Virtual Network User Mode Linux*", Richard Arndt, Seminar Routing 2005, AG Rechnernetze, Universität Koblenz

[2] „*Simple Network Management Protocol*", Memet Edemen, Diplomarbeit 2005, AG Rechnernetze, Universität Koblenz

[3] „*Using Net-SNMP under Linux and µClinux*", Patrice Kadionik, Ausarbeitung 2003, ENSEIRB School of Electrical Engineering Telecommunication and Computer Science http://www.enseirb.fr/~kadionik/embedded/snmp/english/net-snmp_english.html

[4] „*Computernetzwerke*", Andrew S. Tanenbaum, S. 666 ff, 3. rev. Auflage (dt.), Pearson Studium 2000

[5] „*SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*", William Stallings, Third Edition, Addison Wesley 1999

[6] „*Managing Internetworks with SNMP"* , Mark A. Miller, Third Edition, M&T Books 1997

[7] „*SNMP*", Jürgen Klein, Techniker Projekt 2003, Werner-Siemens-Schule http://www.jklein.de/technikerseite.htm

[8] „*Netzwerkmanagement*", Raffaele Fossati, Daniel Hasler, Andreas Przygienda, Walter Wellauer, Seminar Kommunikationssysteme 2000, Universität Zürich

[9] The Simple Web - Tutorial slides http://www.simpleweb.org/tutorials/slides.html

[10] Request For Comments (RFC) Bezugsquelle http://www.rfc-editor.org

[11] Die freie Enzyklopädie http://www.wikipedia.de

[12] Online Lexikon für Informationstechnologie http://www.itwissen.info

[13] „*VNUML on Gentoo Guide*", Jaques Landru, Online HOW-TO 2004, Gentoo Technologies, http://www.enic.fr/people/landru/viminal/vnuml.gentoo/how-to/vnuml-gentoo-guide.html