INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN
Lehrstuhl Prof. Dr. Heinz-Gerd Hegering

Fortgeschrittenenpraktikum

Erweiterung eines SNMPv2-Agenten um eine Schnittstelle zur Interprozeßkommunikation

Erwin Hainzinger

21. September 1994

Betreuer: Markus Gutschmidt und Dr. Bernhard Neumair

Inhaltsverzeichnis

Kapitel 1

Motivation und erste Überlegungen

1.1 Derzeitige Situation

Um in einem heterogenen Netzwerk viele Rechner zu überwachen werden unteranderem Netz- und Systemmanagementagenten eingesetzt. Diese Agenten stellen managementrelevante Daten des Rechners, auf dem sie laufen, bereit. Diese Daten liegen in der Management Information Base (MIB) bereit und können über ein Managementprotokoll gelesen bzw. geschrieben werden. Heute wird häufig als Managementprotokoll das Simple Network Management Protocol Version 1 (SNMPv1) verwendet, doch immer mehr wird sich SNMPv2 durchsetzen.

Aktuelle Implementierungen von Agenten enthalten gleichzeitig das Managementprotokoll und die MIB, d.h. beide Aufgaben werden durch genau einen Prozeß realisiert. Deshalb ist zum Ändern der MIB es nötig, den gesamten Agenten anzuhalten, neu zu kompilieren, zu linken und ihn wieder zu starten.

Durch dieses Fortgeschrittenpraktikum soll nun ein SNMPv2-Agent so erweitert werden, daß er die Möglichkeit bereitstellt, Teile, der durch ihn zugreifbaren MIB, durch andere Prozesse zu implementieren.

Es gibt bereits seit längerem ein Protokoll zur Interprozeßkommunikation für SNMPv1 Agenten, das diese Möglichkeit bietet. Dieses SNMP Multiplex-Protokoll (SMUX-Protokoll) müßte man also nur geeignet erweitern und implementieren um es auch in SNMPv2 Agenten einsetzen zu können. Eine zweite Möglichkeit, welche sich erst im Verlaufe des Fopras bot, ist das SNMP Distributed Protocol Interface Version 2.0 (SNMP DPI2.0). Dieses Protokoll wurde im März 1994 von IBM als Entwurf einer RFC15xx bekanntgegeben und kurz darauf als Experimental RFC1592 angenommen. Dieses Protokoll ist bereits für SNMPv2 Agenten geeignet.

Als Agent, der erweitert werden soll, wurde der CMU-Agent von der Carnegie Mellon University ausgewählt, da dieser sowohl SNMPv1 als auch SNMPv2 sprachig ist. Dadurch ist es möglich, ihn heutzutage mit SNMPv1 und in Zukunft mit SNMPv2 Managementstationen abzufragen.

1.2 Überlegungen zur Lösung

Der Agent kommuniziert über das SNMP-Protokoll mit der Managementstation. Kommt eine Anfrage von der Managementstation, sucht der Agent die Variable in seiner MIB. Diese MIB setzt sich aus seiner internen Teil-MIB und den Teil-MIBs der einzelnen Subagenten zusammen, siehe dazu die Abbildung 1.1. Eine Teil-MIB, welche aus einem oder mehreren Teilbäumen besteht, ist ein Ausschnitt der kompletten MIB, die der Agent kennt. Der Subagent ist dabei ein eigener Prozeß, der eine eigene MIB besitzt, die er über ein Protokoll, dem Agenten zur Verfügung stellt. Damit der Agent die gesamte MIB kennt, müssen sich die Subagenten bei ihm anmelden und ihre MIB dem Agenten als Teil-MIB bekanntgeben. Subagenten können sich jederzeit an- oder abmelden und nur Teile oder ihre komplette MIB aktivieren. Die Anzahl der Subagenten an einem Agenten sollte nicht begrenzt sein und die Subagenten sollten völlig unabhängig vom Agenten sein. Ihre einzige Verbindung zum Agenten ist das Interprozeßkommunikationsprotokoll.

Einleitend eine Erklärung von OID (Object Identifier): Alle Variablen der MIB sind durch eine OID bezeichnet. Diese Bezeichnung ist standardisiert und weist auf eine eindeutige Position in der MIB. Man kann die MIB als Baum darstellen, wobei jedem Knoten eine Zahl zugewiesen wird und dieser Knoten sich weiter verzweigen oder eine Variable beherbergen kann. Folgt man nun einem Pfad von der Wurzel bis zur Variablen und merkt sich die Nummern der durchlaufenen Knoten erhält man die OID, z.B. syscontact = (1.3.6.1.4.1.1.2.0). Im weiteren Verlauf wird nicht immer zwischen der Variablen und dessen OID unterschieden, es sollte aber aus dem Zusammenhang klar sein. Entsprechend werden Teilbäume bezeichnet, die jetzt definiert werden.

Damit der Subagent dem Agenten mitteilen kann, welche Variablen sich bei ihm befinden, müssen mehrere Variablen zu einer Menge zusammengefaßt werden. Der Subagent kann nicht jede einzelne Variable an den Agenten melden, das wäre zu umständlich. Da die MIB strukturiert und eine Ordnung darauf definiert ist, kann man die MIB in Bereiche zerlegen. Da sich die MIB als Baum darstellen läßt, kann man einen Ast dieses Baumes als Teilbaum bezeichnen. Ein Teilbaum repräsentiert also einen Teil des MIB-Baumes und sein Bezeichner (Teilbaum-OID) ist der Pfad durch den MIB-Baum bis zu seiner Wurzel, wie es in Abbildung 1.2 gezeigt ist. Alle Variablen dessen OIDs die Teilbaum-OID als Präfix haben, müssen in diesem Teilbaum enthalten sein. Deshalb genügt es, dem Agenten die Teilbäume des Subagenten mitzuteilen. Befindet sich eine bestimmte Variable nicht in diesem Teilbaum, gibt es sie in der kompletten MIB des Agenten nicht.

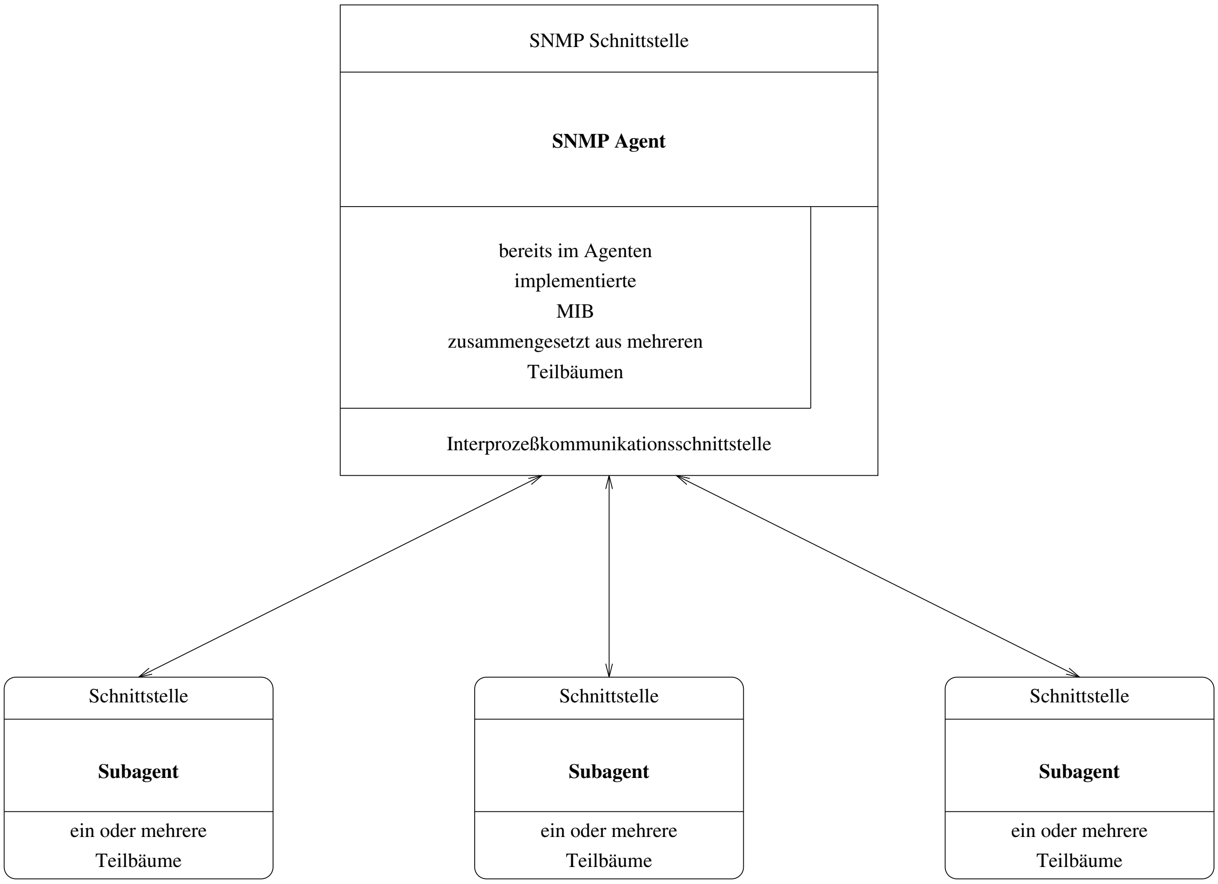

Abbildung 1.1: Subagenten am Agenten

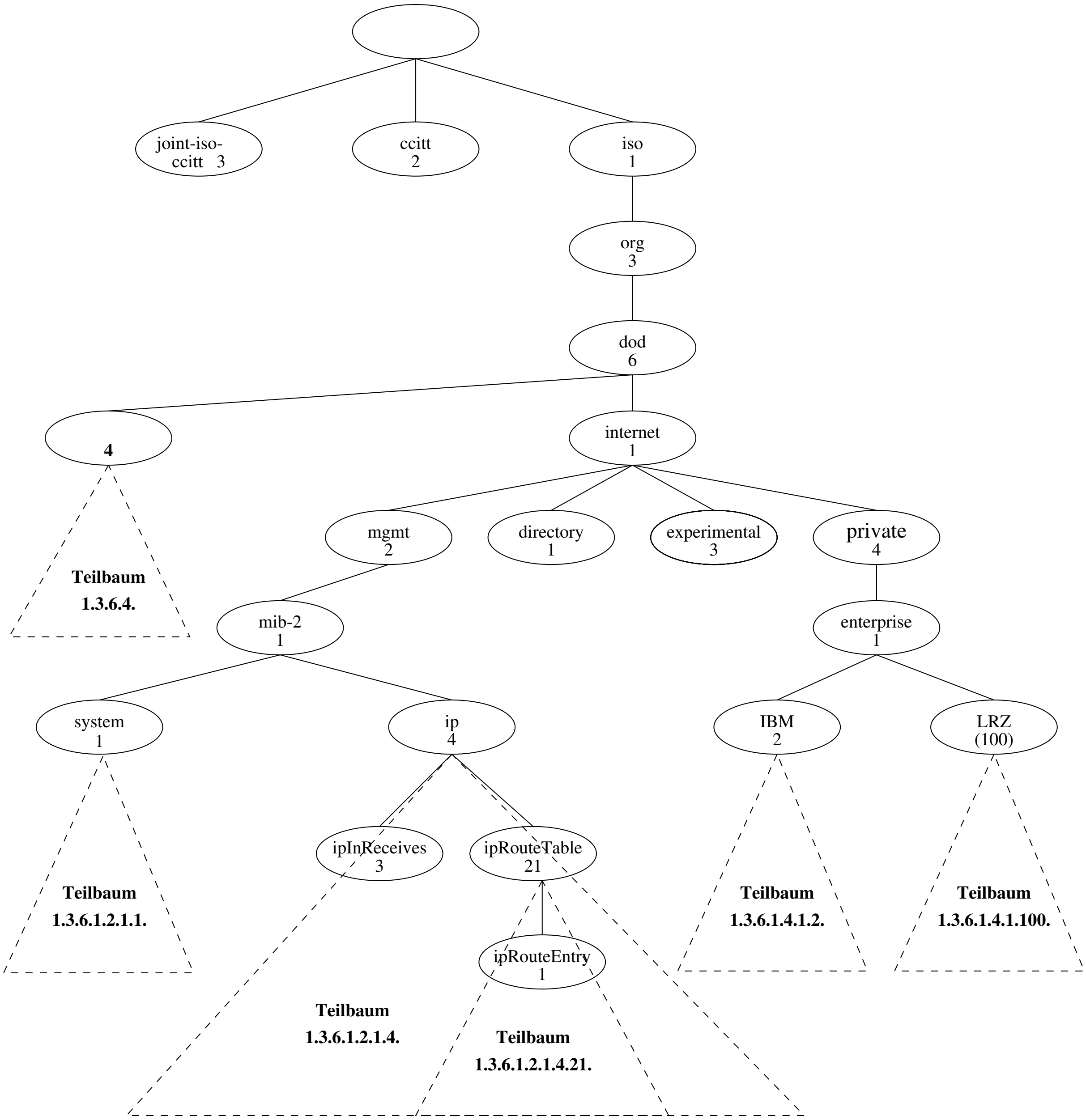

Abbildung 1.2: Teilbäume in der MIB

Kapitel 2

"Vergleich" zwischen SMUX und DPI

2.1 Kurze Beschreibung des SMUX-Protokolls

Das SMUX-Protokoll ist ein sehr einfaches Protokoll, siehe [8]. Der Agent wartet auf einen Verbindungsaufbauwunsch eines Subagenten, sobald die Verbindung aufgebaut ist, erhält er ein OpenPDU vom Subagenten. Diesen Verbindungsaufbauwunsch kann er ablehnen, indem er ein ClosePDU sendet und die Verbindung abbricht oder er akzepiert es und wartet anschließend auf die RReqPDUs, mit denen sich die Teilbäume vom Subagenten anmelden. Auf die RReqPDUs antwortet der Agent mit RRspPDUs um den Erfolg der Registrierung der Teilbäume anzuzeigen.

Der Subagent kann auch TRAPs an den Agenten schicken, die dieser an die Managementstation weiterleitet.

Erhält der Agent eine SNMP GetRequest PDU, GetNextRequest PDU oder SetRequest PDU, welche Variablen beinhaltet, die bei einem Subagenten registriert sind, wird eine entsprechende SNMP PDU erzeugt, in der nur die Variablen enthalten sind, welche beim Subagenten registriert sind. Diese SNMP PDU wird vom Subagenten durch eine SNMP GetResponse PDU beantwortet, worauf der Agent die einzelnen PDUs von den verschiedenen Subagenten wieder zu einer SNMP Response PDU vereinigt und an die Managementstation zurücksendet. Bei einer SetRequest PDU ist die Abwicklung etwas aufwendiger, hier muß die Bearbeitung über ein einfaches zwei-phasen Bestätigungsprotokoll, zwischen dem Agenten und Subagenten, abgewickelt werden, da die Setoperation atomar ist.

Desweiteren ist es dem Subagenten gestattet jederzeit einen seiner Teilbäume wieder abzumelden.

Soll die Verbindung geschlossen werden, wird ein ClosePDU gesendet, wobei sowohl der Agent als auch der Subagent den Abbruch der Verbindung veranlassen können.

2.2 Schwächen von SMUX

Hier werden nur einige Gründe aufgezählt, warum SMUX nicht besonders für eine Interprozeßkommuniktionsschnittstelle für SNMPv2 Agenten geeignet ist.

Die Definition des SMUX Protokolls im RFC1227 ist sehr kurz und läßt einige Fragen offen, so werden auch nur die wirklich notwendigsten Funktionen für SNMPv1 abgedeckt. Desweiteren erweckt der RFC1227 nicht den Eindruck hier einen allgemeingültigen Standard gefunden zu haben, der in Zukunft breitere Unterstützung erfährt und in kommerziellen Produkten Anwendung finden wird. Das SMUX Protokoll wurde auch von nur acht Personen an einem Nachmittag ausgearbeitet, man kann also kaum mehr erwarten.

Bei den PDUs zwischen dem Agenten und dem Subagenten wird bei Get, Set und GetNext PDUs ein großer Overhead mitübertragen. Die Subagenten erhalten ein fast kompletes SNMP Paket, weshalb sie auch die Definitionen von ASN.1 und SNMP PDUs kennen müssen.

Die angemeldeten Subagenten werden in die MIB eingetragen und können dadurch sogar von der Managementstation aus abgemeldet werden. Das ist eine schöne Sache, aber überflüssig. Der Agent soll als ein einziger Prozeß erscheinen, für die Managementstation sollen die Subagenten unsichtbar sein.

Die Erweiterung des SMUX Protokolls für SNMPv2 (GetBulk, InformPDU, Trap2) wäre kein Standard mehr, nur selbst entwickelte Subagenten könnten damit kommunizieren.

Die Implementierung des neu entwickelten SMUX Protokolls wäre sehr aufwendig, da es kaum Unterstützung gibt. Mir ist nur der ISODE-Agent bekannt, der das SMUX Protokoll implementiert hat, dieser Agent ist allerdings nur ein SNMPv1 Agent. Außerdem existiert zu diesem Agenten weder eine Dokumentation noch ist der Quellcode ausreichend kommentiert. Es ist zwar möglich Teile der SMUX Implementierung in diesem Agenten zu finden, aber den genauen Ablauf festzustellen und die Schnittstelle zwischen dem SMUX Protokoll und dem Agenten zu finden ist eine sehr zeitaufwendige Angelegenheit.

Auch das Fehlen eines Subagenten mit SMUX-Protokoll, der zur Entwicklung und zum Testen der Schnittstelle nötig wäre, erschwert die Implementierung der Schnittstelle.

Aufgrund dieser Probleme und dem Erscheinen des DPI-Protokolls wurde die Arbeit an der Erweiterung und Implementierung des SMUX-Protokolls eingestellt.

2.3 Entscheidung für das DPI-Protokoll

Viele Probleme die sich beim SMUX-Protokoll stellten sind bereits beim DPI-Protokoll [3] gelöst. SNMP DPI20 wurde bereits an SNMPv2 angepaßt, ist besser durchdacht und mächtiger als SMUX. Es ist vor allem ein Standard, der von einer großen Firma entwickelt wurde, und auch unterstützt wird. So werden bereits fast alle nötigen Funktionen zur Implementierung eines Subagenten bereitgestellt und es existiert ein kleiner Subagent, der aus diesen Funktionen zusammengesetzt ist.

Einige der Funktionen für den Subagenten (z.B. das Erzeugen von DPI Paketen oder empfangene DPI Pakete parsen) können auch im Agenten verwendet werden. Desweiteren sind viele Konstanten bereits definiert und werden übernommen.

Ein weiterer großer Vorteil ist das Vorhandensein eines Beispielsubagenten, der die Definitionen des DPI–Protokolls einhält. Damit läßt sich die DPI–Schnittstelle im CMU–Agenten entwickeln und testen.

Kapitel 3

Beschreibung des SNMP DPI20 Protokolls

3.1 Einleitung

Durch das DPI Protokoll ist es möglich, die MIB eines Agenten zu erweitern, ohne diesen neu zu kompilieren, ja sogar ohne ihn zu stoppen. Die Subagenten (clients) melden sich beim Agenten via der DPI Schnittstelle an und registrieren ihre Teilbäume. Zuerst benötigt der Subagent jedoch die DPI Portnummer, die er durch ein SNMP GET Packet an den Agenten erhält, um sich anmelden zu können. Nachdem ihm der DPI Port bekannt ist, schickt der Subagent ein DPI OPEN Packet an den Agenten um eine Verbindung aufzubauen und sich anzumelden. Der Agent bearbeitet die Anfrage und schickt ein entsprechendes DPI RESPONSE Paket zurück.

Jetzt können sich die Teilbäume, die der Subagent bereitstellt, beim Agenten registrieren, damit der Agent weiß, welche Variablen sich im Subagenten befinden.

Sind dem Agenten die Variablen in seinen Subagenten bekannt ist die Erweiterung der MIB in diesem Agenten abgeschlossen.

3.2 Die DPI Architektur

Der SNMP Agent kann folgende Anfragen an den Subagenten schicken:
GET, GETNEXT, GETBULK, SET, COMMIT, UNDO, UNREGISTER und CLOSE.
Die ersten vier werden direkt von den entsprechenden SNMP Anfragen abgeleitet, die von der Managementstation kommen. Die restlichen vier sind SNMP DPI spezifische Anfragen, wobei jedoch COMMIT und UNDO noch zur korrekten Durchführung einer SNMP SET

Anfrage dienen. Auf alle Anfragen vom Agenten antwortet der Subagent mit einem RESPONSE Paket, ausgenommen auf eine CLOSE Anfrage, darauf wird einzig die Verbindung abgebaut.

Dem Subagenten stehen folgende Anfragen zu Verfügung:
OPEN, REGISTER, TRAP, UNREGISTER, ARE_YOU_THERE und CLOSE.
Der Subagent erhält vom Agenten auf alle Anfragen ein RESPONSE Paket, ausgenommen auf TRAP und CLOSE Anfragen. Das TRAP Paket wird vom Agenten in ein SNMP TRAP Paket verwandelt und an die Managementstation weitergeleitet. Erhält der Agent ein CLOSE Paket baut er die Verbindung ab und entfernt den Subagenten aus seiner Subagenten-Liste.

Einen Überblick über den SNMP Agenten mit seinen Schnittstellen zur Managemantstation (SNMP Protokoll) und zum Subagenten (DPI Protokoll) erhält man in Abbildung 3.1, wobei die Reihenfolge der hier aufgeführte Punkte inetwa dem normale Ablauf der Kommunikation zwischen dem Agenten und einem Subagenten entspricht.

- Der SNMP Agent kommuniziert mit dem SNMP Manager über das SNMP Protokoll.
- Der SNMP Agent holt sich die Werte zu den Variablen aus seiner statisch gelinkten MIB, falls sie nicht in einem Subagenten implementiert sind.
- Der SNMP Subagent, welcher als eigenständiger Prozeß sogar auf anderen Rechnern laufen kann, kann eine Verbindung zum Agenten aufbauen. Dabei kann er zwischen UDP und TCP Verbindungen wählen.
- Sobald die Verbindung aufgebaut ist, sendet der Subagent ein OPEN Paket und ein oder mehrere REGISTER Anfragen zum Agenten, um sich und seine MIB Teilbäume zu registrieren.
- Der Subagent antwortet auf die OPEN und REGISTER Anfragen mit RESPONSE Paketen um Erfolg oder Mißerfolg anzuzeigen.
- Der Agent decodiert die empfangenen SNMP Pakete. Enthält ein solches Paket Variablen für einen Teilbaum, der in einem Subagenten implementiert ist, sendet der Agent ein entsprechendes DPI Paket an den Subagenten. Ist die Anfrage für ein SNMP GETBULK verwandelt der Agent dieses in mehrere DPI GETNEXT und leitet diese an die entsprechenden Subagenten. Bei einer Anfrage für ein DPI SET benützt der Agent ein 2-phasen Bestätigungsschema, d.h. er sendet dem Subagenten eine Sequenz von SET/COMMIT, SET/UNDO oder SET/COMMIT/UNDO Paketen.
- Der Subagent schickt die Antwort in einem RESPONSE Paket zurück.
- Darauf vereint der Agent die DPI RESPONSE Pakete zu einem SNMP RESPONSE Paket und schickt es an die Managementstation.

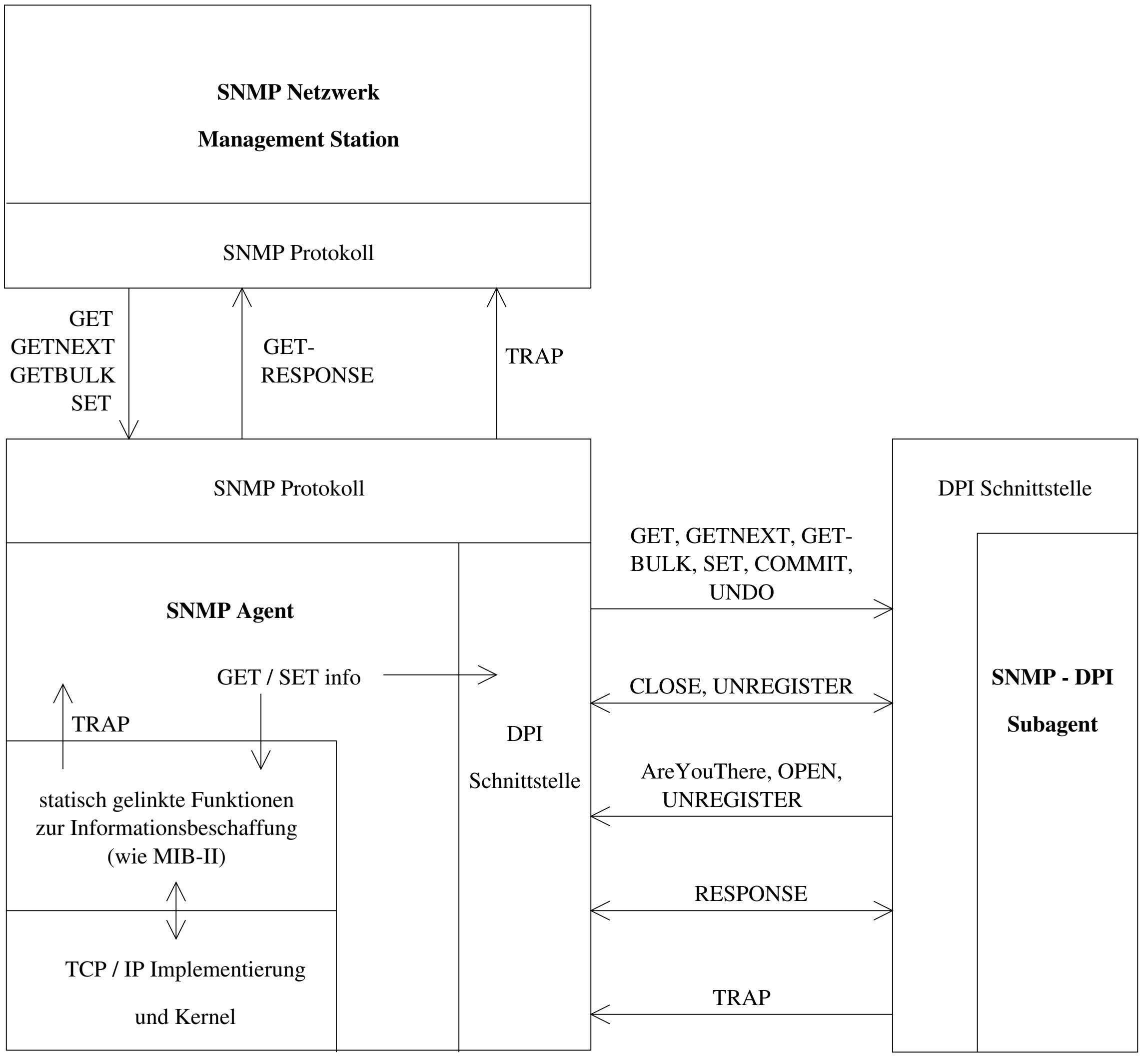

Abbildung 3.1: Die DPI Schnittstelle

- Will der Subagent eine wichtige Statusänderung melden, sendet er ein DPI TRAP Paket an den Agenten, der es in ein SNMP TRAP Paket konvertiert und an die Managementstation weiterleitet.
- Wenn der Subagent seine Arbeit beenden will, sendet er ein DPI UNREGISTER für jeden Teilbaum und ein DPI CLOSE Paket an den Agenten. Der Agent antwortet nur auf die DPI UNREGISTER Pakete mit DPI RESPONSE Paketen.
- Es gibt kein RESPONSE Paket zu einem CLOSE Paket, der Agent beendet die Verbindung und entfernt alle noch registrierten Teilbäume des Subagenten.
- Auch der Agent kann ein DPI UNREGISTER Paket senden, z.B wenn ein Teilbaum mit einer höheren Priorität registriert wurde, und erhält darauf vom Subagenten ein DPI RESPONSE Paket.
- Der Agent kann auch ein DPI CLOSE Paket an den Subagenten schicken, um anzuzeigen, daß er die Verbindung beendet.
- Um zu Prüfen ob die Verbindung noch besteht, kann der Subagent ein DPI ARE_YOU_THERE Paket an den Agenten senden. Erhält er ein DPI RESPONSE Paket vom Agenten, das keinen Fehler meldet, ist alles in Ordnung. Ein DPI RESPONSE Paket mit Fehlermeldung oder kein Paket signalisieren ein Problem mit der Verbindung.

3.3 SNMP DPI Paket Format

3.3.1 Standardheader eines DPI Paketes

Die Header der verschiedenen DPI Pakete enthalten die gleichen Informationen. Nach dem Header folgen dann die für den Pakettyp entsprechenden Daten. In Tabelle 3.1 ist der Header dargestellt.

3.3.2 OPEN

Damit der Subagent mit dem DPI fähigen Agenten kommunizieren kann, müssen eine Verbindung aufgebaut und einige Vereinbarungen getroffen werden. Dazu sendet der Subagent ein OPEN Paket an den Agenten, folgende OPEN spezifische Daten sind darin enthalten:

- ein Timeout-Wert
- die maximale Anzahl von Variablen pro DPI Paket

OFFSET	FELD
0	Die Länge dieses Paketes
2	Protokollhauptversion
3	Protokollunterversion
4	Protokollrelease
5	Paket-id
7	Pakettype

Tabelle 3.1: SNMP DPI Paketheader

- gewählter Zeichensatz
- die Subagenten-ID als nullterminierter String
- eine Subagentenbeschreibung als nullterminierter String
- ein optionales Paßwort

Der Subagent erwartet ein RESPONSE Paket vom Agenten, das den Erfolg oder Mißerfolg anzeigt. Trat ein Fehler auf, wird die Verbindung beendet.

3.3.3 CLOSE

Wenn der Subagent die Verbindung mit dem Agenten beenden will, schickt er diesem ein SNMP DPI CLOSE Paket. Daraufhin schließt der Agent die Verbindung und entfernt alle Teilbäume, die zu diesem Subagenten gehören, aus seiner Teilbaumliste. Der Subagent wird abschließend aus der Subagentenliste entfernt und die Aktion ist beendet, der Subagent erhält kein RESPONSE Paket mehr.

Umgekehrt kann auch der Agent die Verbindung beenden, indem er ein SNMP DPI CLOSE Paket an den Subagenten schickt.

Ein CLOSE Paket besteht aus dem Standardheader, plus dem Grund für das Beenden der Verbindung.

3.3.4 ARE_YOU_THERE

Der Subagent erhält durch das ARE_YOU_THERE Paket die Möglichkeit zu überprüfen, ob die Verbindung zum Agenten noch existiert. Dies ist nötig, weil der Subagent passiv auf Anfragen vom Agenten wartet und deshalb nicht erfährt, daß der Agent abgestürzt ist oder unvorschriftsmäßig beendet wurde. Falls alles in Ordung ist, erhält der Subagent vom Agenten ein RESPONSE Paket mit einem Errorcode 0 und einen Errorindex 0. Ist die Verbindung nicht in Ordung, erhält der Subagent kein oder ein RESPONSE Paket, mit einem Errorcode ungleich 0, auf sein ARE_YOU_THERE Paket.

Ein ARE_YOU_THERE Paket besteht nur aus dem Standardheader.

3.3.5 REGISTER

Um einen Ast (Teilbaum) im MIB Baum zu registrieren, sendet der Subagent ein SNMP DPI REGISTER Paket an den Agenten. Ein Subagent kann mehrere Teilbäume haben, für die je ein REGISTER Paket zu senden ist.

Ein REGISTER Paket besteht aus dem Standardheader, plus speziellen REGISTER Daten. Diese Daten sind:

- die verlangte Priorität.
  Dabei gibt es drei Möglichkeiten:

  1. n = -1 : Es wird die beste Priorität verlangt.
  2. n = 0 : Die nächstbeste Priorität wird verlangt.
  3. n $\geq$ 1 : Die Priorität n wird verlangt.

  Je niedriger die Nummer n, desto höher die Priorität. Die vergebene Priorität für den registrierten Teilbaum wird dem Subagenten durch ein RESPONSE Paket mitgeteilt.

- ein Timeout-Wert.
  Ist dieser Wert 0, wird der Timeout vom Subagenten genommen.

- eine Anzeige, ob der Subagent die MIB view selection handhaben will.
  Nicht alle Agenten können das und geben deshalb die entsprechende Antwort.

- eine Anzeige, ob der Subagent in der Lage ist GETBULK durchzuführen.

- die Teilbaum-ID, des Teilbaums, der registriert werden soll.

Der Agent antwortet auf ein REGISTER Paket mit einem RESPONSE Paket, mit gleicher Paket-ID und dem Erfolg bzw. Mißerfolg der Aktion.

3.3.6 UNREGISTER

Um einen Teilbaum abzumelden sendet der Subagent ein UNREGISTER Paket an den Agenten. Dieses Paket besteht aus dem Standardheader, plus der Bezeichnung des Teilbaums, der abgemeldet werden soll, und dem Grund des Abmeldens.

Der Agent antwortet darauf mit einem RESPONSE Paket, mit der gleichen Paket-ID des UNREGISTER Paketes, welches Erfolg oder Scheitern signalisiert.

3.3.7 GET / GETNEXT

Wenn der Agent eine SNMP GET bzw. GETNEXT Anfrage, für eine Variable, die in einem Subagenten implementiert ist, erhält, dann wird ein DPI GET bzw. GETNEXT an den Subagenten weitergeleitet. In diesem Paket ist der Standardheader plus folgende Daten:

- der Community–Name, welcher in der SNMP PDU enthalten ist
- pro Variable zwei nullterminierte Strings (Teilbaum-ID und Instance-ID Paar)

3.3.8 GETBULK

Falls der Subagent in der Lage ist ein DPI GETBULK Paket zu bearbeiten, wir es ihm durch den Agenten zugestellt, ansonst wandelt der Agent das SNMP GETBULK Paket in DPI GETNEXT Pakete um. In diesem Paket ist der Standardheader plus folgende Daten:

- die Anzahl der Variablen im Paket, die nur einen Nachfolger liefern sollen (Non-repeaters)
- die Anzahl der Nachfolger bei Variablen, von denen mehrere Nachfolger gesucht werden (Max-repetitions)
- pro Variable zwei nullterminierte Strings (Teilbaum-ID und Instance-OID Paar)

3.3.9 SET, COMMIT und UNDO

Wenn der Agent ein SNMP SET Paket erhält, das eine Variable beinhaltet, die in einem Subagenten registiert ist, wird eine der drei Sequenzen von DPI Paketen ausgeführt, dabei erhält der Agent auf jedes Paket eine Antwort.

- SET, COMMIT
  Dies ist der normale Ablauf zum Setzen einer Variable. Nach dem Erhalt eines SET Paketes überprüft der Subagent, ob diese Variable geändert werden kann und die Resourcen (Speicher) zur Verfügung stehen. Anschließend erhält der Subagent ein COMMIT Paket, das ihn dazu veranlaßt diese Änderungen durchzuführen.
- SET, UNDO
  Sind in einer SNMP SET Anfrage mehrere Variablen für verschiedene Subagenten, wird an jeden betroffenen Subagenten ein DPI SET Paket gesendet. Trat hierbei ein Fehler bei einem Subagenten auf, wird an alle anderen Subagenten, die keinen Fehler zurücklieferten ein UNDO Paket gesendet, damit das SET Paket storniert wird.

- SET, COMMIT, UNDO
  Sehr selten kann es auch vorkommen, daß erst beim COMMIT Paket das Schreiben einer Variablen scheitert. Jetzt muß an alle betroffenen Subagenten ein UNDO Paket gesendet werden, auch an die Subagenten, die COMMIT erfolgreich durchgeführt haben.

Diese Pakete bestehen aus dem Standardheader plus SET spezifischen Daten:

- der Community–Name, von der SNMP PDU
- pro Variable zwei nullterminierte Strings (Teilbaum-ID und Instance-ID) und dem Type, der Länge und dem Wert der Variablen

3.3.10 RESPONSE

Der Subagent antwortet auf ein GET, GETNEXT, GETBULK, SET, COMMIT, UNDO und UNREGISTER mit einem RESPONSE Paket.
Dieses Paket bestehen aus dem Standardheader, plus RESPONSE spezifischen Daten:

- einem ErrorCode
- einem ErrorIndex
- bei einem erfolgreichem GET, GETNEXT, GETBULK,
  pro Variable ein (Name, Type, Länge, Wert) Tupel.

Bei einer nicht erfolgreichen GET, GETNEXT, GETBULK Anfrage ist es nicht nötig das Tupel (Name, Type, Länge, Wert) zurückzugeben, denn der Agent hat diese Informationen bereits.

Die Teilbaum-ID und die Paket-ID werden vom zu beantwortendem Paket übernommen.

Beim RESPONSE Paket zu einem SET, COMMIT oder UNDO ist es nicht nötig irgendwelche Variableninformation zurückzuliefern, denn diese sind dem Agenten noch bekannt.

Beim RESPONSE Paket zu einem REGISTER oder UNREGISTER Paket wird keine Variable mitgeliefert, deshalb ist die Instance-ID 0. Im RESPONSE Paket zu einem REGISTER Paket, das erfolgreich verlief, wird die zugewiesene Priorität im Errorindex-Feld zurückgegeben.

Beim RESPONSE Paket zu einem OPEN, ARE_YOU_THERE oder CLOSE wird nur der Standardheader mit dem Errorcode und dem Errorindex zurückgesandt.

3.3.11 TRAP

Der Subagent kann den Agenten dazu veranlassen ein SNMPv1 oder SNMPv2 TRAP an die Managementstation zu senden, durch Senden eines TRAP Paketes.

Diese Pakete bestehen aus dem Standardheader plus TRAP-spezifischen Daten:

- der Generic und Specific Trap Code
- optional die EnterpriseID als nullterminierter String
- optional eine Menge von (Teilbaum-ID, Instance-ID, Type, Länge, Wert) Tupeln

Kapitel 4

Implementierung der DPI-Schnittstelle im Agenten

4.1 Die Initialisierung des Agenten für DPI

4.1.1 Die MIB–Erweiterung im Agenten

Die Definition der MIB im Agenten, welche sich im File 'mib.txt' befindet, wurde um folgendes erweitert.

```
DPI20-MIB DEFINITIONS ::= BEGIN

    -- Objects in this MIB are implemented in the local SNMP agent.

       IMPORTS
           MODULE-IDENTITY, OBJECT-TYPE, snmpModules, enterprises
                       FROM SNMPv2-SMI;

       ibm OBJECT IDENTIFIER ::= { enterprises 2 }
       ibmDPI OBJECT IDENTIFIER ::= { ibm 2 }
       dpi20MIB OBJECT IDENTIFIER ::= { ibmDPI 1 }

    -- dpi20MIB MODULE-IDENTITY
    -- LAST-UPDATED "9401210000Z"
    -- ORGANIZATION "IBM Research - T.J. Watson Research Center"
    -- CONTACT-INFO " Bert Wijnen
    -- Postal:  IBM International Operations
```

```
    -- Watsonweg 2
    -- 1423 ND Uithoorn
    -- The Netherlands
    -- Tel:  +31 2975 53316
    -- Fax:  +31 2975 62468
    -- E-mail:  wijnen@vnet.ibm.com"
    -- DESCRIPTION "MIB module describing DPI objects."
    -- ::= { snmpModules x }

       dpiPort OBJECT IDENTIFIER ::= { dpi20MIB 1 }

       dpiPortForTCP OBJECT-TYPE
               SYNTAX INTEGER (0..65535)
               ACCESS read-only
               STATUS mandatory
               DESCRIPTION "The TCP port number on which the agent
                         listens for DPI connections.  A zero value
                         means the agent has no DPI TCP port."
               ::= { dpiPort 1 }

       dpiPortForUDP OBJECT-TYPE
               SYNTAX INTEGER (0..65535)
               ACCESS read-only
               STATUS mandatory
               DESCRIPTION "The UDP port number on which the agent
                         listens for DPI packets.  A zero value
                         means the agent has no DPI UDP port."
               ::= { dpiPort 2 }
END
```

Die Implementierung der beiden Variablen

- dpiPortForTCP, (1.3.6.1.4.2.2.1.1.1.0) und
- dpiPortForUDP, (1.3.6.1.4.2.2.1.1.2.0)

findet im File 'snmp_vars.c' statt und die zugehörige Funktion 'var_dpiport()' zum Lesen der Variablen findet sich im File 'dpi.c'.

Eine genaue Beschreibung zur Erweiterung der MIB im CMU-Agenten findet man in der Dokumentation [4] des Fopras "Erweiterung eines SNMPv2-Agenten für Systemmanagementaufgaben" von Bastian Pusch.

4.1.2 Die DPI Ports initialisieren

Beim Starten des Agenten müssen auch die Ports für die DPI Schnittstelle eingerichtet werden. Durch den Aufruf der Funktion 'komm_init()' wird zum einen ein TCP-Socket und zum anderen ein UDP-Socket erzeugt und anschließend an einen Port gebunden. Dabei kann der zugewiesene Port im Bereich zwischen 2000 und 3000 liegen.

Nachdem die Ports angelegt sind, werden sie noch in die MIB eingetragen, damit jeder Subagent die gewünschte Portnummer aus der MIB via SNMP GET lesen kann.

4.1.3 Registrierung der internen Teilbäume

Damit die internen Teilbäume bei der Suche nach dem richtigen Teilbaum mitberücksichtigt werden, müssen diese zu Beginn in die Teilbaumliste aufgenommen werden.

Mit Hilfe der Funktion 'interne_teilbaeume_registrieren()' werden die internen Teilbäume ermittelt und in die Teilbaumliste eingetragen.

Als Priorität erhalten sie die schlechtest mögliche, damit sie von Teilbäumen in Subagenten leicht verdrängt werden können. Außerdem ist ihr Subagentenzeiger NULL, damit sie von externen Teilbäumen unterschieden werden können.

4.1.4 Ermittlung der TRAP Ziele

Sollte ein Subagent ein DPI TRAP Paket schicken, ist der Agent dafür verantwortlich dieses an alle bekannten Managementstationen weiterzuleiten. Damit er weiß auf welchen Rechnern die Managementstationen sich befinden muß er sich die Daten aus der Datei 'trap.conf' holen. Diese Datei befindet sich mit den anderen Konfigurationsdateien im Verzeichnis '/etc'.

In dieser Datei sind alle Rechner eingetragen, die ein SNMPv1 TRAP Paket erhalten sollen. Dazu sind folgende Informationen nötig:

- der Rechnername
- die IP-Adresse in Punktnotation
- und der Communitystring für die Managementstation auf dem Zielrechner

Diese Informationen sind für jeden Rechner in einer Zeile angeordnet und durch Leerzeichen zu trennen. Kommentarzeilen sind durch ein '#' am Zeilenbeginn zu kennzeichnen.

Eine kleine Beispieldatei:

```
# Dies ist eine Beispieldatei für die Datei 'trap.conf'
# Syntax: Rechnername IP-Adresse Communitystring

sunhegering7 129.187.214.1 public
sun4 129.187.214.2 public

#Dieser Rechner erhält keine TRAP Pakete
#sunhegering2 129.187.214.3 public
```

Der Agent liest diese Datei und legt die gefundenen Informationen in der 'trap_ziele' Liste ab.

4.2 Die SNMP Pakete

4.2.1 SNMP GET Anfrage

Der CMU–Agent übernimmt das Parsen des SNMP GET Headers und erzeugt gleichzeitig den SNMP RESPONSE Header. Man erhält schließlich einen Zeiger auf den Beginn der Variablenliste im SNMP Paket. Für jede dieser Variablen wird nun nacheinander die gesuchte Variablen-OID aus dem Paket geholt, der Wert dieser Variablen aus der MIB ermittelt und die Variablen-OID mit ihrem gefundenen Wert, Type und Länge in das SNMP RESPONSE Paket geschrieben.

Für jede Variablen-OID muß dazu der richtige Teilbaum gefunden werden. Dazu wird die 'teilbaum_liste' bei der besten Priorität beginnend durchlaufen, bis ein Teilbaum gefunden wird, dessen ID Präfix der gesuchten Variablen-OID ist. Dabei kann dieser Teilbaum im Agenten selbst oder in einem Subagenten implementiert sein.

Ist es ein interner Teilbaum wird auf die Funktionen des CMU–Agenten zurückgegriffen um den Wert, Typ, Länge und Zugriffsrecht zu erhalten. Handelt es sich jedoch um einen Teilbaum in einem Subagenten muß ein DPI GET Paket erzeugt werden. In diesem DPI GET Paket befindet sich nur eine Variable, obwohl auch mehrere Variablen möglich wären. Der Grund hierfür ist, daß die Variablen im SNMP Paket nicht alle für den gleichen Teilbaum sein müssen und besonders dann bei SNMP GETNEXT Paketen viele Aspekte der Teilbaumauswahl berücksichtigt werden müssen.

Im DPI RESPONSE Paket sollte sich dann der Wert, der Type und die Länge der gesuchten Variablen befinden oder der Grund (Fehlerart) warum der Wert nicht geliefert werden konnte. Dabei darf die Wartezeit auf das RESPONSE Paket die vorher vereinbarten Timeout-Zeit nicht überschreiten.

Die gefundenen Werte bzw. die Fehlerart werden in das SNMP RESPONSE Paket geschrieben, auch wenn überhaupt kein geeigneter Teilbaum gefunden wurde, wird die entsprechende Fehlermeldung zurückgeliefert.

Sobald für alle Variablen in der Variablenliste die Werte geholt wurden, wird der Header des SNMP RESPONSE Paketes vervollständigt und das Paket an die Managementstation geschickt.

4.2.2 SNMP SET Anfrage

Die SNMP SET Anfrage verläuft im Prinzip genauso wie die SNMP GET Anfrage, nur das die Werte zu einer Variablen-OID nicht geholt werden, sondern geliefert und in die MIB geschrieben werden sollen.

Es werden also aus dem SNMP Paket neben der Variablen-OID auch der Wert, der Type und die Länge ermittelt. Im Gegensatz zu den anderen Pakettypen müssen aber alle Variablen aus der Variablenliste gesetzt werden oder keine. Deshalb sind bei einer SNMP SET Variablenliste bis zu drei Durchläufe nötig.

Dabei muß man wieder interne Teilbäume und Teilbäume in Subagenten unterscheiden. Ist die Variable in einem Subagenten zu finden muß ein DPI SET Paket erzeugt werden. Aus den gleichen Gründen wie im DPI GET Paket befindet sich auch hier nur eine Variable im Paket.

Das DPI RESPONSE Paket signalisiert dann, ob die Aktion erfolgreich durchgeführt werden kann oder nicht möglich ist.

Beim ersten Durchlauf wird geprüft, ob die Variable existiert, sie verändert werden kann, der Typ richtig ist und genügend Speicher vorhanden ist. Sollte das Ganze bei einer Variablen scheitern, wird ein SNMP RESPONSE Paket erzeugt, das das Scheitern des Setzens mitteilt und einen Index, ab welcher Variablen in der Variablenliste die Aktion abbrach, liefert.

Im zweiten Durchlauf werden die Werte in die Variablen kopiert, sollte erst jetzt das Setzen eines Wertes scheitern, müssen die bereits erfolgreich veränderten Variablen zurückgesetzt werden und wieder ein SNMP RESPONSE Paket erzeugt werden, das das Mißlingen der Aktion anzeigt. Sollte auch das Rücksetzen der Variablen scheitern, ist auch das im SNMP RESPONSE Paket zu vermerken.

Verlief alles Ordungsgemäß ist ein SNMP RESPONSE Paket an die Managementstation zu senden, in dem die korrekte Bearbeitung bestätigt wird.

4.2.3 SNMP GETNEXT Anfrage

Die SNMP GETNEXT Anfrage verläuft im Prinzip genauso wie die SNMP GET Anfrage, nur daß nicht die Werte der in der Variablenliste angebenen Variablen geholt werden, sondern die Werte der Nachfolgervariablen.

Das größte Problem ist dabei, den richtigen Teilbaum zu finden. Denn die registrierten Teilbäume dürfen sich überdecken, d.h. ein Teilbaum kann Teile eines anderen Teilbaumes überdecken, was durch die verschiedenen Prioritäten der Teilbäume möglich ist.

Dabei wird folgendermaßen vorgegangen:

- Als erstes wird der Teilbaume mit der höchsten Priorität gesucht, dessen ID Präfix der angegebenen Variablen-OID ist. Wird in diesem Teilbaum die gewünschte Variable gefunden ist man fertig.
- Als nächstes versucht man einen Teilbaum zu finden, der echt Präfix des vorher gefundenen Teilbaumes ist und dabei wieder die höchste Priorität hat. Dabei darf der vorher gefundene Teilbaum natürlich nicht Präfix der in diesem Teilbaum gefundenen Variablen-OID sein. Denn dieser Bereich wird ja schon von dem vorherigen Teilbaum abgedeckt. Die in diesem Teilbaum gefundene Variablen-OID muß also ein Nachfolger von allen im vorherigen Teilbaum möglichen Variablen-OIDs sein. Hat man eine solche Variable gefunden ist man fertig.
- Den letzten Punkt wiederholt man solange bis man die gesuchte Variable findet bzw. kein solcher Teilbaum mehr zu finden ist.
- Hatte man bisher keinen Erfolg, sucht man sich den nächsten infragekommenden Teilbaum unter Berücksichtigung der Priorität und holt sich von dort die erste registriert Variable.
- Findet man keinen Teilbaum mehr, dessen OID echt Präfix des vorher gefundenen Teilbaumes ist, ist man fertig.
- Hat man doch einen Teilbaum gefunden, muß geprüft werden, ob die erste in diesem Teilbaum registrierte Variablen-OID vor der bereits vorher gefundenen Variablen-OID liegt. Sollte dies der Fall sein, ist diese Variable die gesuchte Variable.
- Die beiden vorhergehenden Punkte wiederholt man nun solange, wie noch geeignete Teilbäume gefunden werden.

Beim Holen der Werte muß wie immer zwischen internen und externen Teilbäumen unterschieden werden.

Sollte während der ganzen Suche kein geeigneter Wert gefunden worden sein, ist als Wert die Konstante 'endOfMIBView' zurückzugeben, womit die Managemantstation weiß, daß in diesem Agenten das Ende der MIB erreicht wurde.

Im SNMP RESPONSE Paket sind die gefundenen Werte einzutragen und an die Managementstation zu senden.

Aufgrund dieser sehr aufwendigen Suche nach dem richtigen Teilbaum ist es empfohlen, dafür zu sorgen keine Teilbäume zu registrieren, die Präfix eines anderen bereits registrierten Teilbaumes sind. Sind alle Teilbäume disjunkt, wird die gesuchte Variable in der Regel beim ersten geeigneten Teilbaum gefunden, wenn nicht gerade das Ende der Teilbaum-MIB erreicht wurde.

4.2.4 SNMP GETBULK Anfrage

Diese Sonderform der SNMP GETNEXT Anfrage wird genauso wie ein SNMP GETNEXT Paket bearbeitet, d.h. die GETBULK Anfrage wird in mehrere GETNEXT Anfragen zerlegt. Bei den Variablen, bei denen mehrere Nachfolger gesucht sind, werden auch mehrere GETNEXT Anfragen gestellt. Dabei wird jeweils die zuletzt gefundene Variablen-OID als neue im GETNEXT-Paket anzugebende Variablen-OID genommen.

4.3 Die DPI Pakete

4.3.1 DPI OPEN

Sobald der CMU-Agent ein Paket an einem seiner DPI Ports erhält versucht er mit dem Absender des Paketes eine Verbindung aufzubauen. Während er bei einer UDP Verbindung nichts machen muß, wird bei einer TCP Verbindung ein neuer Socketdeskriptor erzeugt und mit dem Port der Gegenseite verbunden. Steht die Verbindung, kann er das angekommene DPI OPEN Paket auswerten.

Als erstes muß überprüft werden, ob nicht bereits ein Subagent mit der gleichen Subagenten-ID angemeldet ist. Sollte das der Fall sein und der angemeldete Subagent kommuniziert über eine UDP Verbindung wird durch ein DPI GETNEXT Paket an den angemeldeten Subagenten überprüft, ob dieser Subagent überhaupt noch existiert. Erhält man innerhalb einer gewissen Zeit keine Antwort von ihm, wird er aus der Subagentenliste entfernt und mit ihm seine Teilbäume in der Teilbaumliste. Ist der angemeldete Subagent allerdings noch aktiv, kann der neue Subagent nicht angemeldet werden. Bei einer TCP Verbindung erkennt man sofort, wenn sich die Gegenseite beendet. Deshalb muß kein Kontrollpaket geschickt werden.

Der neue Subagent kann auch nicht angemeldet werden, wenn er nicht mit dem ASCII Zeichensatz arbeiten möchte. Desweiteren wäre es möglich ein Paßwort vom Subagenten zu verlangen, ohne dieses ein Anmeldung nicht erlaubt wäre (wurde in der Implementierung durch Kommentarzeichen ausgeschaltet).

Zur Bestimmung der Timeout-Zeit wird die im DPI OPEN Paket mitgelieferte Subagenten-Timeout-Zeit mit der im Agenten festgelegten maximalen Agenten-Timeout-Zeit verglichen. Solange die Subagenten-Timeout-Zeit kleiner ist, wird diese genommen, ansonsten die Agenten-Timeout-Zeit.

Jetzt können alle weiteren Daten des Subagenten aus dem DPI OPEN Paket übernommen werden und ein Eintrag für diesen Subagenten in der Subagentenliste gemacht werden.

Zum Abschluß wird dem Subagenten durch ein DPI RESPONSE Paket der Verlauf der Anmeldung angezeigt.

Sollte die Anmeldung nicht geklappt haben, wird auch noch ein DPI CLOSE Paket an den Subagenten geschickt und die Verbindung wieder abgebaut.

4.3.2 DPI CLOSE

Durch ein DPI CLOSE Paket kann sich der Subagent vom Agenten abmelden. Erhält der Agent ein solches Paket, sucht er sich den Eintrag aus der Subagentenliste, welcher davon betroffen ist. Er erkennt diesen Eintrag an der Kommunikationsverbindung, denn jeder Subagent hat eine eigene Verbindung. Deshalb wird die Verbindung, auf welcher das DPI CLOSE Paket kam, und die im Eintrag angegebene Subagentenverbindung auf Gleichheit hin untersucht. Dazu wird bei einer TCP-Verbindung der Socketdeskriptor verglichen und bei einer UDP-Verbindung die Zieladresse.

Wurde der Eintrag gefunden, löscht man diesen und mit ihm auch alle Teilbäume aus der Teilbaumliste, welche zu diesem Subagenten gehörten.

Es ist auch möglich, daß der Agent ein DPI CLOSE Paket an den Subagenten schickt, wie das der Fall ist, wenn die Anmeldung eines Subagenten fehlschlug.

Auf ein DPI CLOSE Paket wird kein DPI RESPONSE Paket gesendet.

4.3.3 DPI REGISTER

Mit einem DPI REGISTER Paket kann ein Subagent einen seiner Teilbäume beim Agenten anmelden. Für jeder Teilbaum wird ein eigenes DPI REGISTER Paket an den Agenten geschickt, um ihn zu registrieren.

Jeder Teilbaum erhält eine Priorität beim Agenten, der Teilbaum kann jedoch Einfluß auf die Prioritätsvergabe nehmen. Dazu stehen ihm drei Möglichkeiten zur Auswahl:

- Durch den Wert -1 zeigt er an, die beste Priorität zu wollen.
- Durch den Wert 0 zeigt er an, die nächstbeste Priorität zu wollen. Um diese Priorität zu finden wird der Teilbaum mit der beste Priorität in der Teilbaumliste gesucht, dessen Teilbaum-ID Präfix der neuen Teilbaum-ID ist oder umgekehrt. D.h. der Teilbaum mit der höchsten Priorität, der Variablen des neuen Teilbaumes überdecken

könnte. Wurde ein solcher Teilbaum in der Teilbaumliste gefunden, erhält der neue Teilbaum eine um den Wert 1 bessere Priorität. Wenn nicht, erhält der neue Teilbaum die schlechteste zu vergebende Priorität.

- Durch einen Wert ≥ 1 möchte der Teilbaum genau diese Priorität. Deshalb wird geprüft, ob bereits ein gleicher Teilbaum mit dieser Priorität in der Teilbaumliste exitiert.

Konnte keine geeignete Priorität gefunden werden, ist es nicht möglich den Teilbaum zu registrieren. Dabei können die Prioritäten von 1 bis MAXPRIORITAET (100000) vergeben werden, wobei kleinere Werte bessere Prioritäten sind.

Wünscht der Teilbaum viewselection, d.h. der Communitystring von jedem SNMPv1 Paket soll an den Teilbaum durchgereicht werden, muß die Registrierung des Teilbaumes leider abgelehnt werden.

Desweiteren muß bereits der Subagent in dem der Teilbaum implementiert ist, angemeldet sein, sonst ist die Regstrierung nicht möglich. Denn es muß ein Link vom Teilbaum auf seinen Subagenten in der Subagentenliste hergestellt werden.

Erfüllt der Teilbaum alle Kriterien, kann er in die Teilbaumliste eingetragen werden, mit allen nötigen Daten aus dem DPI REGISTER Paket. Dazu muß aber die richtige Stelle in der Teilbaumliste gefunden werden. Die Teilbaumliste ist nach der Priorität der Teilbäume sortiert und zwar die beste Priorität zuerst. Haben mehrere Teilbäume die gleiche Priorität wird nach der Länge der Teilbaum-ID sortiert und zwar längere IDs zuerst.

War die Registrierung des Teilbaumes erfolgreich, erhält der Subagent ein DPI RESPONSE Paket mit der zugewiesenen Priorität. Konnte die Registrierung nicht durchgeführt werden, erhält er ein DPI RESPONSE Paket mit der entsprechenden Fehlermeldung.

4.3.4 DPI UNREGISTER

Mit einem DPI UNREGISTER Paket kann sich ein Teilbaum, welcher bei einem Subagenten implementiert ist, vom Agenten abmelden.

Der Agent sucht dazu den Teilbaum in seiner Teilbaumliste. Zur eindeutigen Identifizierung genügt es jedoch nicht, nur die Teilbaum-ID zu überprüfen, denn es können mehrere Teilbäume mit der gleichen ID, aber unterschiedlicher Priorität registriert sein. Deshalb muß auch überprüft werden, ob der zugehörige Subagent des Teilbaumes aus der Teilbaumliste mit dem Subagenten, von dem das DPI UNREGISTER Paket kam, identisch ist.

Wieder wird durch ein DPI RESPONSE Paket an den Subagenten angezeigt, ob die Aktion erfolgreich war.

4.3.5 DPI GET

Wann ein DPI GET Paket erzeugt wird und an welchen Subagenten es für welchen Teilbaum gesendet wird, findet man in der Beschreibung von SNMP GET Paketen.

Die Variablen-OID wird von der Agentendarstellung in einen nullterminierten String umgewandelt und aufgeteilt in die Teilbaum-ID und der Instance, so wird z.B. aus der gesuchten Variablen-OID {1,3,6,4,1,2,2,1,1,2,0} die Teilbaum-ID "1.3.6.4.1.2.2.1.1." mit abschließendem Punkt und der Instance "2.0". Darauf wird das Ganze an den Subagenten mit dem Teilbaum "1.3.6.4.1.2.2.1.1" geschickt.

Nachdem das Paket abgeschickt ist, wird auf das Antwortpaket gewartet, allerdings nur die in Timeout vorgegebene Zeit. Kommt ein Paket, wird überprüft ob es vom richtigen Subagenten kommt und in den Puffer geschrieben.

Verlief alles Ordungsgemäß wird das Paket geparsed und kontrolliert, wie die GET Anfrage beim Subagenten verlief. Hat der Subagent das DPI GET Paket erfolgreich bearbeitet befindet sich im DPI RESPONSE Paket der Typ der Variablen, die Länge des Wertes und der Wert selbst.

Sollte der gefundene Wert ein Object Identifier sein, muß er noch von der Stringdarstellung in die Agentendarstellung umgewandelt werden, bei den anderen Typen ist das nicht notwendig.

4.3.6 DPI GETNEXT

Die Erzeugung eines DPI GETNEXT Paketes verläuft genauso wie die Erzeugung eines DPI GET Paketes. Der einzige Unterschied ist die Bezeichnung des Paketes, hier DPI GETNEXT und die Angabe der Instanze der gesuchten Variablen. Ist der Teilbaum Präfix der gesuchten Variablen-OID wird die Instance angegeben, genauso wie bei DPI GET Paketen. Sollte allerdings der Teilbaum nicht Präfix der gesuchten Variablen-OID sein, ist als Instance "0" anzugeben, damit die erste Variable im Teilbaum zurückgeliefert wird.

Auch das Warten und Auswerten des RESPONSE Paketes geschieht wie beim RESPONSE Paket von DPI GET Anfragen. Nur das in diesem Fall noch die gefundene Variablen-OID von der Stringdarstellung in die Agentendarstellung umgewandelt und zurückgegeben werden muß.

4.3.7 DPI SET

Wann ein DPI SET Paket erzeugt wird und an welchen Subagenten es für welchen Teilbaum gesendet wird, findet man in der Beschreibung von SNMP SET Paketen.

Als erstes muß entschieden werden, welche Art von DPI SET Paket erzeugt werden soll. Davon gibt es drei Arten,

- die SNMP_DPI_SET Paketart, welches zur Überprüfung der Variablen im Subagenten dient. Hier soll kontrolliert werden, ob die Variable existiert, sie beschreibbar ist, ausreichend Speicher zur Verfügung steht und der zu setzende Wert wie Type für die Variable korrekt sind.
- die SNMP_DPI_COMMIT Paketart, womit das Schreiben der Variablen durchgeführt wird.
- und die SNMP_DPI_UNDO Paketart, mit welcher vorhergehende Reservierungen bzw. Änderungen rückgängig gemacht werden.

Bevor jedoch der zu schreibende Wert im DPI SET Paket abgelegt werden kann, muß noch der Wert und der Typ von der SNMP Darstellung in die DPI Darstellung umgewandelt werden.

Das so erzeugt Paket wird nun an den entsprechenden Subagenten geschickt und auf ein DPI RESPONSE Paket gewartet. In diesem RESPONSE Paket ist vermerkt, ob die Aktion erfolgreich war oder ein Fehler aufgetreten ist.

4.3.8 DPI TRAP

Trifft von einem Subagenten ein DPI TRAP Paket ein, wird dieses in ein SNMPv1 TRAP Paket verwandelt und an die entsprechenden Managemantstationen gesendet.

Dazu wird ein Port initialisiert, von dem aus die SNMPv1 TRAP Pakete verschickt werden und der anschließend wieder geschlossen wird.

Die Variablenliste des DPI TRAP Paketes wird in eine Variablenliste für ein SNMPv1 TRAP Paket umgewandelt, wobei auch mehrere Variablen in der Liste erlaubt sind. Anschließend werden die Daten des eigenen Rechners aus der 'trap_ziele' Liste geholt. Aus diesen Daten und dem Traptyp wird nun ein SNMP TRAPv1 Paket erzeugt.

Alle Rechner in der 'trap_ziele' Liste erhalten nun dieses Paket, ausgenommen der erste in der Liste, der ja der eigene Rechner ist. Der Zielport ist immer der Port 162.

4.3.9 DPI ARE_YOU_THERE

Damit der Subagent feststellen kann, ob der Agent noch läuft, kann er ein DPI ARE_YOU_THERE Paket an den Agenten schicken.

Der Agent anwortet darauf mit einem DPI RESPONSE Paket, in welchem keine Daten enthalten sind, sondern nur die Paket-ID des DPI ARE_YOU_THERE Paketes übernommen wurde.

Kapitel 5

Beschreibung der Funktionen

Fast alle hier beschriebenen Funktionen befinden sich in dem File 'dpi.c'. Ausgenommen davon sind die Funktionen

- 'parse_var_op_list()' und 'bulk_var_op_list()', welche sich in dem File 'snmp_agent.c' befinden. Diese Funktionen wurden nur modifiziert, um auch die DPI-Schnittstelle anzusprechen.
- 'my_getStatPtr()' und 'getStatPtr()', welche die alte Funktion 'getStatPtr()' (neuer Name 'old_getStatPtr()') ersetzen, und die 'interne_teilbaeume_registrieren()' Funktion. Diese Funktionen finden sich im File 'snmp_vars.c'.

Hier nicht aufgeführte Funktionen stammen vom Beispielsubagenten von IBM, diese Funktionen befinden sich im File 'snmp_mDPI.c'. Die Beschreibung dieser Funktionen lese man bitte in [3] nach.

Bei jeder Bearbeitung eines SNMP GET, GETNEXT oder SET Paketes wird unteranderem die Funktion 'parse_var_op_list()' aufgerufen. Diese Funktion dient zur Bearbeitung der Variablenliste des Paketes. Welche weiteren Funktionen von hier aufgerufen werden, um sowohl die im Agenten implementierten Variablen als auch jene in Subagenten zu berücksichtigen, sieht man in der Abbildung 5.1. In dieser Abbildung sind die wichtigsten Funktionen in ihrer Aufrufreihenfolge und den möglichen Verzweigungen gezeigt.

Für die Bearbeitung eines SNMP GETBULK Paketes wird die Funktion 'bulk_var_op_list()' aufgerufen, welche sich entweder an die Funktion 'my_getStatPtr()' wendet, um Werte interner Variablen zu erhalten oder an die Funktion 'next_anfrage()', um die Werte von den Subagenten zu holen.

Außerdem muß mit Hilfe der Funktionen 'reading_data_from_udpsockets()' und 'reading_data_from_tcpsockets()' regelmäßig geprüft werden, ob Pakete der Subagenten angekommen sind. Diese Pakete dienen zur An- oder Abmeldung von Subagenten und Teilbäumen,

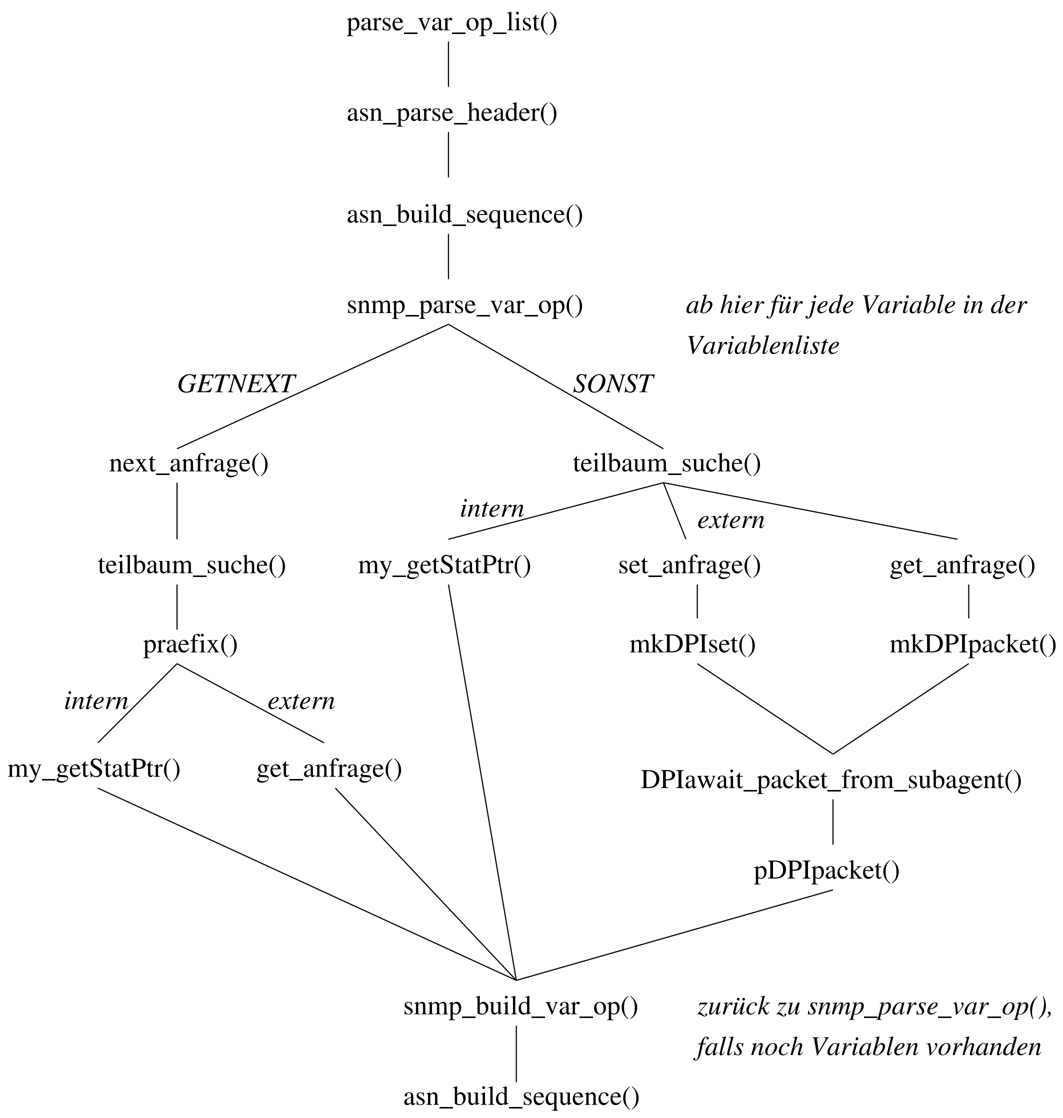

Abbildung 5.1: Funktionsaufrufe bei SNMP Paketbearbeitung

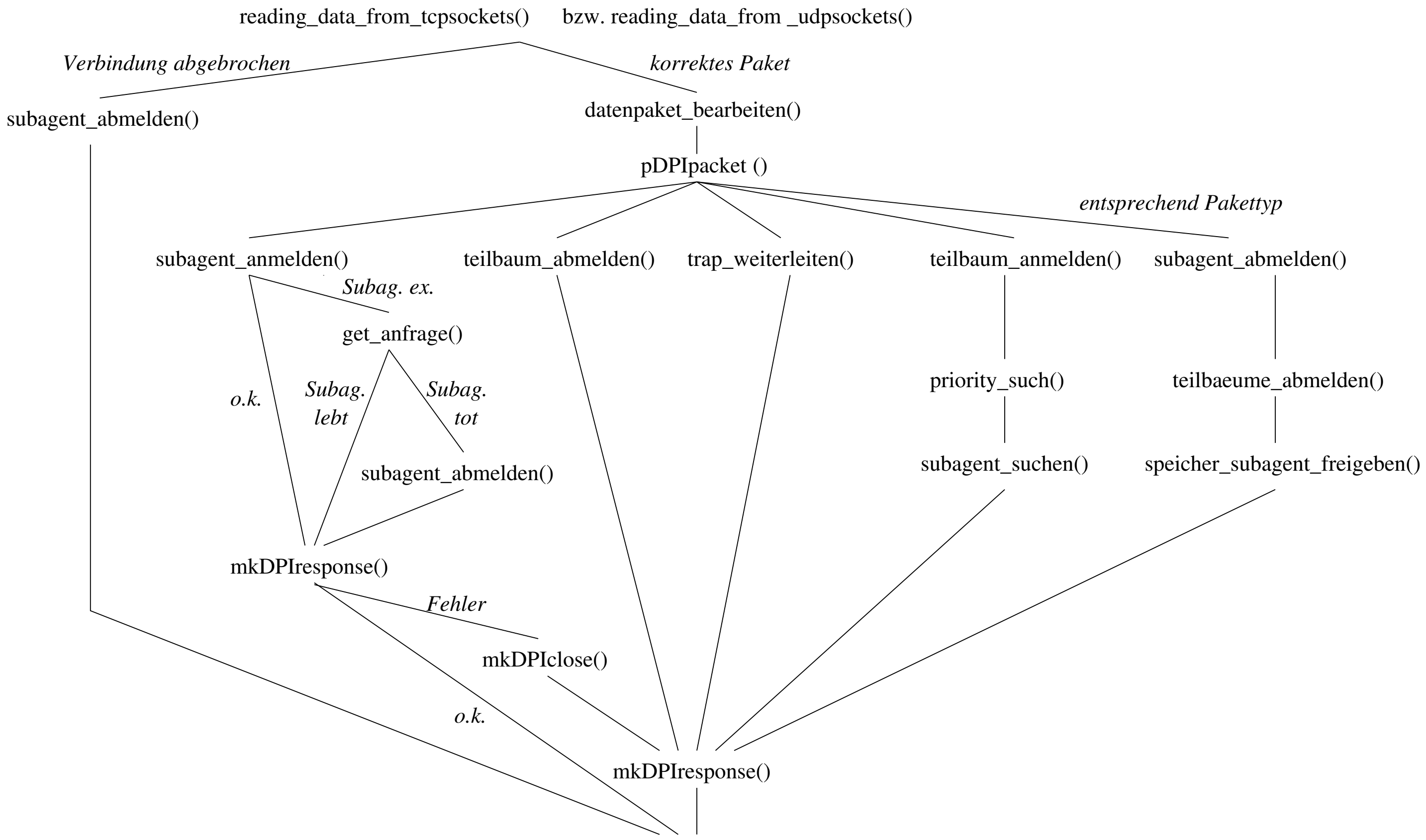

Abbildung 5.2: Funktionsaufrufe bei DPI Kommunikation

dem Weiterleiten von Traps oder zur Überprüfung der Existenz des Agenten. RESPONSE Pakete werden an dieser Stelle nicht erwartet, denn RESPONSE Pakete erfolgen auf eine Anfrage des Agenten und werden dort auch anschließend erwartet. Die Abbildung 5.2 soll auch hier den Aufruf und die Verzweigung der wichtigsten Funktionen zur Erledigung dieser Aufgabe zeigen.

5.1 Konstanten und globale Variablen

Alle Konstanten und Typevereinbarungen, die hier neu definiert wurden, finden sich im File 'dpi.h', die globalen Variablen im File 'dpi.c'.

Alle weiteren Konstanten wurden von der RFC1592 übernommen und befinden sich im File 'snmp_dpi.h'. Desweiteren befinden sich noch Definitionen im File 'snmp_lDPI.h' und globale Variablen im File 'snmp_mDPI.c'. Diese Files wurden dem Beispielsubagenten von IBM entnommen. Die Beschreibung dieser Files findet man in der Dokumentation des Beispielsubagenten von IBM.

5.1.1 Die Struktur 'dpi_subagent'

```
struct dpi_subagent{
      char                    *oid_p;
      char                    *description_p;
      unsigned short          timeout;
      unsigned short          max_varBinds;
      char                    character_set;
      int                     sock_desk;
      int                     versuche;
      struct sockaddr_in      *socketaddr;
      struct dpi_subagent     *next};
```

VARIABLEN

oid_p	Ein Zeiger auf den Namen des Subagenten, der als nullterminierter String abgelegt ist. Der Name kann z.B. "1.3.6.1.4.2" lauten.
descripton_p	Ein Zeiger auf einen nullterminierten String, der den Subagenten näher beschreibt.
timeout	Die Zeit, die der Agent auf eine Antwort vom Subagenten warten soll.
max_varBinds	Die maximale Anzahl von Variablen, die in einem DPI Paket an den betreffenden Subagenten geschickt werden können, sodaß der Subagent es noch bearbeiten kann.

character_set	Der vom Subagenten gewünschte Zeichensatz. Alle Strings müssen in diesem Zeichensatz übertragen werden.
sock_desk	Der Socketdeskriptor, falls der Subagent über eine TCP-Verbindung mit dem Agenten kommuniziert, sonst -1.
versuche	Die Anzahl der erfolglosen Versuche von einem Subagenten eine Antwort zu erhalten. Erhält man eine Antwort, wird dieser Zähler wieder auf 0 gesetzt.
socketaddr	Ein Zeiger auf die Adresse des Subagenten, falls der Subagent über eine UDP-Verbindung mit dem Agenten kommuniziert , sonst NULL.
next	Ein Zeiger auf den nächsten Subagenten in der Subagentenliste.

BESCHREIBUNG
Diese Struktur dient zur Aufnahme und Speicherung der Daten der angemeldeten Subagenten. Diese Subagenten stehen in einer einfach verketteten Liste, der 'subagenten_liste', wobei neue Subagenten einfach an die Liste angehängt und abgemeldete Subagenten aus der Liste entfernt werden.

5.1.2 Die Struktur 'dpi_teilbaum'

```
struct dpi_teilbaum{
    unsigned short          timeout;
    long int                priority;
    char                    *group_p;
    oid                     asn1_oid[ ];
    char                    bulk;
    char                    view;
    snmp_dpi_subagent       *subagent
    int                     oid_len;
    struct subtree          *tp;
    struct dpi_teilbaum     *next};
```

VARIABLEN

timeout	Die Zeit, die der Agent auf eine Antwort vom Subagenten, der diesen Teilbaum enthält, warten soll.
priority	Die Priorität, die der Teilbaum bei der Registrierung erhielt.
group_p	Ein Zeiger auf den Namen des Teilbaumes, abgelegt als nullterminierter String. Der Name ist das gemeinsame Präfix aller in diesem Teilbaum registrierten Variablen. So sind in dem Teilbaum "1.3.6.1.4.1." z.B. die Variablen '1.3.6.1.4.1.1.0', '1.3.6.1.4.1.2.3.0', '1.3.6.1.4.1.20.34.2.0', usw. registriert.

asn1_oid	Der Name des Teilbaumes, diesmal jedoch in der Form wie der Agent OID's speichert.
bulk	Eine Anzeige, ob der in einem Subagenten implementierte Teilbaum GETBULK Anfragen verarbeiten kann.
view	Eine Anzeige, ob der in einem Subagenten implementierte Teilbaum den Communitystring haben möchte.
subagent	Ein Zeiger auf den Subagenten, bei dem der Teilbaum implementiert ist, falls es ein externer Teilbaum ist, sonst NULL.
oid_len	Die Länge des Teilbaumnamens bzw. die Anzahl der Zahlen im Teilbaumnamen, so besteht z.B. der Teilbaumname "1.3.6.1.4.2." aus 6 Zahlen.
tp	Ein Zeiger auf den Teilbaum, falls es ein interner Teilbaum ist, sonst NULL.
next	Ein Zeiger auf den nächsten Teilbaum in der Liste.

BESCHREIBUNG
Diese Struktur dient zur Aufnahme und Speicherung der Daten der registrierten Teilbäume. Diese Teilbäume stehen in einer einfach verketteten Liste, der 'teilbaum_liste', wobei neue Teilbäume an der richtigen Stelle in die Liste eingefügt und abgemeldete Teilbäume aus der Liste entfernt werden müssen.

Die Liste ist sortiert nach der Priorität der Teilbäume, beginnend bei der höchsten Priorität. Bei mehreren Teilbäumen mit gleicher Priorität, kommen zuerst die längeren Teilbäume (Länge = Anzahl der Zahlen in der Variablen-OID). Ist auch die Länge der Teilbäum gleich, entscheidet die Reihenfolge der Anmeldung, dabei sind jüngere bevorzugt.

5.1.3 Die Struktur 'traps'

```
struct traps{
    char              hostname[ ];
    unsigned long     addr;
    char              community[20];
    struct traps      *next};
```

hostname	Der Name des Zielrechners.
addr	Die IP-Adresse des Zielrechners.
community	Der Communitystring für die Zielmanagementstation.
next	Ein Zeiger auf den nächsten Zielrechner.

BESCHREIBUNG
Diese Struktur dient zur Aufnahme und Speicherung der Daten der Zielrechner für SNMPv1

TRAPs. Diese Zielrechner stehen in einer einfach verketteten Liste, der 'trap_ziele' Liste, wobei die Zielrechner aus der 'trap.conf' Datei ausgelesen wurden. Der erste Eintrag in der Liste ist der eigene Rechner, dessen Daten als Absenderangabe benötigt werden.

5.1.4 Konstanten

Es wurden folgende Konstanten vereinbart, die Werte lese man im File 'dpi.h' nach.

- MAXHOSTNAME, gibt die maximale Länge eines Rechnernamens an.
- MAXSOCKETS, gibt die maximale Anzahl der TCP-Verbindungen an.
- BUFLAENGE, ist die Größe des Puffers für die Pakete.
- MAXPRIORITAET, ist die schlechteste Priorität, die ein Teilbaum bekommen kann.
- MAXTIMEOUT, ist die maximale Zeit, auf die auf eine Antwort von einem Subagenten gewartet wird.
- ASN1_OID_LEN, ist die maximale Länge, die eine Variablen-OID im Agenten haben darf.
- STRING_OID_LEN, gibt die maximale Länge der Variablen-OID, als String abgelegt, an.
- COMMUNITY_LEN, ist die maximale Länge des Communitystrings.
- SUBAGENT_VERSUCHE, gibt an, ab wieviel nacheinander folgenden Fehlversuchen, eine Antwort vom Subagenten zu bekommen, der Subagent abgemeldet wird.

5.1.5 Globale Variablen

Alle für die DPI-Schnittstelle global definierten Variablen befinden sich im File 'dpi.c'.

- dpiportfortcp_wert, ist die Variable der MIB, in welcher der Port für TCP Verbindungen abgelegt ist.
- dpiportforudp_wert, ist die Variable der MIB, in welcher der Port für UDP Verbindungen abgelegt ist.
- sock_desk[ ], ist ein Array, in dem alle gerade bestehenden TCP-Verbindungen gespeichert sind.
- string[ ], ist ein Puffer für die Variablen-OID-Umwandlung.

- puffer_in[ ], ist ein Puffer für empfangene DPI Pakete.
- teilbaum_liste, zeigt auf den Beginn der Teilbaumliste.
- subagenten_liste, zeigt auf den Beginn der Subagentenliste.
- trap_ziele, zeigt auf den Beginn der Trapzieleliste.
- internes_passwort, ist das Paßwort mit dem sich alle Subagenten anmelden müssen, ist allerdings noch nicht aktiviert.

5.2 Die var_dpiport() Funktion

BEREICH

Agenten MIB lesen

SYNTAX

```
unsigned char var_dpiport(
    struct variable        *vp,
    oid                    *name,
    int                    *length,
    int                    exact,
    int                    *var_len,
    int                    (**write_methode)());
```

PARAMETER

vp	Ein Zeiger auf eine Struktur, in der alle Variablen, die in diesem Teilbaum liegen, beschrieben sind. In dieser Struktur ist für jede Variable der Bezeichner, der Type, das Zugriffsrecht, der Name der Funktion zum Lesen dieser Variablen (in diesem Fall der Name dieser Funktion), die Länge der Instance und die Instance eingetragen.
name	Ein Zeiger auf die OID der gesuchten Variablen. Als Ergebnis erhält man einen Zeiger auf die OID der gefundenen Variablen.
length	Ein Zeiger auf die Länge der gesuchten Variablen-OID. Als Ergebnis erhält man einen Zeiger auf die Länge der gefundenen Variablen-OID.
exact	Diese Variable dient zur Unterscheidung, ob es sich um eine GET bzw. SET oder um eine GETNEXT Anfrage handelt. Falls sie den Wert 1 besitzt ist es eine GET bzw. SET Anfrage, bei 0 eine GETNEXT Anfrage.
var_len	Als Ergebnis erhält man einen Zeiger auf die Länge des gefundenen Wertes der gesuchten Variablen.

write_methode — Als Ergebnis normalerweise ein Zeiger auf eine Funktion mit der es möglich ist in die Variable einen Wert zu schreiben. Da aber in diesem Teilbaum alle Variablen nur gelesen werden können, ist dies ein NULL–Zeiger.

RETURNWERT

Bei Erfolg erhält man einen Zeiger auf den Wert der gesuchten Variablen. Trat ein Fehler auf oder existiert die gesuchte Variable in diesem Teilbaum nicht wird NULL zurückgegeben.

BESCHREIBUNG

Diese Funktion dient zum Lesen der beiden MIB-Variablen 'dpiportfortcp_wert (1.3.6.1.4.2.2.1.1.1.0)' und 'dpiportforudp_wert (1.3.6.1.4.2.2.1.1.2.0)'. In diesen Variablen sind die Ports für eine TCP bzw. UDP Verbindung abgelegt, an denen sich die Subagenten anmelden können.

5.3 Die komm_init() Funktion

BEREICH

Initialisierung

SYNTAX

```
unsigned int komm_init()
```

RETURNWERT

Folgende Rückgabewerte sind möglich:

- 0, erfolgreiche Ausführung
- -1, Socket initialisieren schlug fehl
- -2, Socket an Port binden schlug fehl

BESCHREIBUNG

Ein Kommunikationsport für UDP-Verbindungen und einer für TCP-Verbindungen werden angelegt. Die Portnummern werden in die MIB des Agenten eingetragen und ein Array, in dem die TCP-Verbindungen gespeichert werden, wird initialisiert. Dabei ist der erste Eintrag im Array der Socketdeskriptor des gerade erzeugten TCP-Kommunikationsports.

5.4 Die read_trap_database() Funktion

BEREICH

Initialisierung

SYNTAX

```
int read_trap_database(
      char                          *filename);
```

PARAMETER

filename	Ein Zeiger auf den Namen des Files in dem alle Rechner angegeben sind die ein SNMP TRAPv1 Paket an den UDP Port 162 geschickt bekommen sollen. Der Name des Files muß dabei mit dem kompletten Pfad angegeben sein.

RETURNWERT

Bei Erfolg erhält man den Wert 1, sonst 0.

BESCHREIBUNG

Alle Rechner die ein SNMP TRAPv1 Paket erhalten sollen, müssen im Konfigurationsfile eingetragen sein. Dieses File wird mit dieser Funktion gelesen und die gefundenen Rechner, mitsamt den noch nötigen Daten, werden in die Struktur 'traps' eingetragen.

5.5 Die lookup_host() Funktion

BEREICH

Initialisierung

SYNTAX

```
unsigned long lookup_host(
      char                          *hostname_p);
```

PARAMETER

hostname_p	Ein Zeiger auf eine IP-Adresse in Punktnotation oder der Rechnername.

RETURNWERT

Die IP-Adresse wird zurückgegeben.

BESCHREIBUNG

Die Funktion liefert zu einer IP-Adresse in Punktnotation bzw. dem Rechnernamen die IP-Adresse.

5.6 Die reading_data_from_tcpsockets() Funktion

BEREICH

Pakete empfangen

SYNTAX

```
int reading_data_from_tcpsockets()
```

RETURNWERT

Bei Erfolg erhält man den Wert 0, sonst -1.

BESCHREIBUNG

An allen im Array 'sock_desk[ ]' eingetragenen Socketdeskriptoren wird nachgesehen, ob Daten anliegen, dabei dürfen maximal 32 Deskriptoren existieren. Der erste Eintrag im Array 'sock_desk[ ]' ist dabei der Socketdeskriptor, an dem sich alle neuen Verbindungsaufbauwünsche anmelden müssen. Liegen hier Daten an wird versucht eine neue Verbindung aufzubauen. Die anderen Einträge im Array sind schon existierende Verbindungen, von denen die Daten gelesen werden und der 'datenpaket_bearbeiten()' Funktion übergeben werden. Bricht die Gegenseite die Verbindung ab oder kann der Subagent nicht angemeldet werden, wird der Socketdeskriptor geschlossen.

5.7 Die reading_data_from_udpsockets() Funktion

BEREICH

Pakete empfangen

SYNTAX

```
int reading_data_from_udpsockets()
```

RETURNWERT

Bei Erfolg erhält man den Wert 0, sonst -1.

BESCHREIBUNG

Am UDP Port wird geprüft ob Daten anliegen. Anliegende Daten werden gelesen und der Funktion 'datenpaket_bearbeiten()' weitergereicht.

5.8 Die datenpaket_bearbeiten() Funktion

BEREICH

DPI Paket bearbeiten

SYNTAX

```
int datenpaket_bearbeiten(
    char                    *puffer,
    int                     sd,
    struct sockaddr_in      *socketaddr);
```

PARAMETER

puffer	Ein Zeiger auf das Paket, das gerade am Port empfangen wurde.
sd	Der Socketdeskriptor, falls das empfangene Paket ein TCP-Paket ist, sonst -1.
socketaddr	Ein Zeiger auf die Adresse des Absenders des Paketes, falls es ein UDP-Paket ist, sonst NULL.

RETURNWERT

Folgende Werte sind möglich:

- 0, bei erfolgreicher Ausführung
- -1, bei einem fehlerhaften Paket
- -2, bei einem falschen Pakettyp
- -3, bei einer erfolglosen Anmeldung des Subagenten (bei einer TCP-Verbindung muß die Verbindung noch getrennt werden)

BESCHREIBUNG

Das angekommene DPI Datenpaket im 'puffer' wird bearbeitet. Je nachdem, ob es sich um ein OPEN, REGISTER, TRAP, ARE_YOU_THERE, UNREGISTER oder CLOSE DPI Paket handelt werden die entsprechenden Aktionen aufgerufen. Hat das angekommene DPI Paket einen anderen Typ wird eine Fehlermeldung ausgegeben. Abschließend wird ein RESPONSE Paket an den Absender des DPI Datenpaketes geschickt.

5.9 Die subagent_anmelden() Funktion

BEREICH

Subagent

SYNTAX

```
int subagent_anmelden(
    snmp_dpi_open_packet    *neuer_subagent,
    int                     sd,
    struct sockaddr_in      *socketaddr);
```

PARAMETER

*neuer_subagent	Ein Zeiger auf die Struktur 'snmp_dpi_open_packet', in der alle nötigen Informationen für eine Subagentenanmeldung enthalten sind.
sd	Der Socketdeskriptor, falls das empfangene Paket ein TCP-Paket ist, sonst -1.

socketaddr	Ein Zeiger auf die Adresse des Absenders des Paketes, falls es ein UDP-Paket ist, sonst NULL.

RETURNWERT

Folgende Werte sind möglich:

- SNMP_ERROR_DPI_noError, bei erfolgreicher Ausführung
- SNMP_ERROR_DPI_duplicateSubAgentIdentifier, falls ein Subagent mit gleichem Identifier bereits registriert ist
- SNMP_ERROR_DPI_characterSetSelectionNotSupported, falls der Subagent nicht den ASCII Zeichensatz wünscht.
- SNMP_ERROR_DPI_otherError, falls ein anderer Fehler auftrat
- -2, ein alter Subagent besetzt noch den Socketdeskriptor

BESCHREIBUNG

Ein Subagent, der sich anmeldet, wird in die Liste der Subagenten aufgenommen. Doch zuvor wird überprüft, ob er die gestellten Kriterien erfüllt. Sollte bereits ein UDP-Subagent mit gleichem Identifier existieren, wird überprüft, ob dieser noch läuft.

5.10 Die subagent_abmelden() Funktion

BEREICH

Subagent

SYNTAX

```
int subagent_abmelden(
    snmp_dpi_close_packet    *subagent,
    int                      sd,
    struct sockaddr_in       *socketaddr);
```

PARAMETER

*subagent	Ein Zeiger auf die Struktur 'snmp_dpi_close_packet', in der der Grund für die Subagentenabmeldung enthalten ist (wird nicht beachtet).
sd	Der Socketdeskriptor, falls das empfangene Paket ein TCP-Paket ist, sonst -1.
socketaddr	Ein Zeiger auf die Adresse des Absenders des Paketes, falls es ein UDP-Paket ist, sonst NULL.

RETURNWERT

Bei Erfolg erhält man den Wert 0, bei -1 wurde der Subagent nicht in der 'subagenten_liste' gefunden.

BESCHREIBUNG

Der TCP-Subagent, mit dem eine Verbindung am Socketdeskriptor 'sd' besteht bzw. der UDP-Subagent, mit der Adresse 'socketaddr', wird aus der 'subagenten_liste' entfernt. Davor werden jedoch alle seine Teilbäume aus der 'teilbaum_liste' entfernt.

5.11 Die subagent_suchen() Funktion

BEREICH

Subagent

SYNTAX

```
snmp_dpi_subagent *subagent_suchen(
    char                    vorgaenger_gesucht,
    int                     sd,
    struct sockaddr_in      *socketaddr);
```

PARAMETER

vorgaenger_gesucht	Falls 'vorgaenger_gesucht' = 0 ist, wird ein Zeiger auf den Subagenten in der 'subagenten_liste' gesucht. Bei 'vorgaenger_gesucht' = 1, wird ein Zeiger auf den Vorgänger des Subagenten in der 'subagenten_liste' gesucht.
sd	Der Socketdeskriptor, falls das empfangene Paket ein TCP-Paket ist, sonst -1.
socketaddr	Ein Zeiger auf die Adresse des Absenders des Paketes, falls es ein UDP-Paket ist, sonst NULL.

RETURNWERT

Einen Zeiger auf den gesuchten Subagenten in der 'subagenten_liste' bzw. auf den Vorgänger. Falls er nicht gefunden wurde einen NULL-Zeiger.

BESCHREIBUNG

Der TCP-Subagent, mit dem eine Verbindung am Socketdeskriptor 'sd' besteht bzw. der UDP-Subagent, mit der Adresse 'socketaddr', wird in der 'subagenten_liste' gesucht. Dabei kann man sich entweder einen Zeiger auf den Subagenten oder auf seinen Vorgänger zurückgeben lassen.

5.12 Die speicher_subagent_freigeben() Funktion

BEREICH
Subagent

SYNTAX

```
void speicher_subagent_freigeben(
    snmp_dpi_subagent        *subagent);
```

PARAMETER

subagent	Ein Zeiger auf die Struktur des Subagenten, dessen Speicher freigegeben werden soll.

RETURNWERT
void

BESCHREIBUNG
Der Speicher, der von dem Subagenten 'subagent' in der 'subagenten_liste' belegt wird, wird freigegeben.

5.13 Die interne_teilbaeume_registrieren() Funktion

BEREICH
Teilbaum

SYNTAX

```
void interne_teilbaeume_registrieren();
```

RETURNWERT
void

BESCHREIBUNG
Alle Teilbäume, die im Agenten registriert sind, müssen in die 'teilbaum_liste' aufgenommen werden. Interne Teilbäume haben keinen Subagenten, deshalb ist 'teilbaum→subagent' mit dem Wert NULL belegt. Außerdem erhalten die Teilbäume die schlechteste Priorität. Da die im Agenten registrierten Teilbäume disjunkt sind und auch keine Teilbaum-OID Präfix einer anderen Teilbaum-OID ist, ist es nicht nötig die Priorität zu überprüfen, es muß nur die richtige Stelle in der Liste gefunden werden.

5.14 Die teilbaum_anmelden() Funktion

BEREICH
Teilbaum

SYNTAX

```
int teilbaum_anmelden(
    snmp_dpi_reg_packet      *teilbaum,
    int                      sd,
    struct sockaddr_in       *socketaddr,
    int                      *prio);
```

PARAMETER

teilbaum	Ein Zeiger auf die Struktur 'snmp_dpi_reg_packet', in der alle Daten für die Teilbaumanmeldung enthalten sind.
sd	Der Socketdeskriptor, falls das empfangene Paket ein TCP-Paket ist, sonst -1.
socketaddr	Ein Zeiger auf die Adresse des Absenders des Paketes, falls es ein UDP-Paket ist, sonst NULL.
prio	Ein Zeiger auf die Priorität, welche der Teilbaum in der 'teilbaum_liste' erhalten hat.

RETURNWERT

Folgende Werte sind möglich:

- 0, bei erfolgreicher Ausführung
- SNMP_ERROR_DPI_alreadyRegistered, falls der Teilbaum nicht angemeldet werden konnte, da er bereits registriert ist
- SNMP_ERROR_DPI_higherPriorityRegistered, falls der Teilbaum zwar angemeldet wurde, aber bereits eine bessere Priorität existiert
- SNMP_ERROR_DPI_viewSelectionNotSupported, falls der Teilbaum nicht angemeldet wurde, weil der Subagent bei SNMPv1 Paketen auch den Communitystring haben möchte
- SNMP_ERROR_DPI_mustOpenFirst, falls der Teilbaum nicht angemeldet wurde, da der zugehörige Subagent nicht gefunden wurde
- SNMP_ERROR_DPI_otherError, falls der Teilbaum wegen eines anderen Fehlers nicht angemeldet werden konnte

BESCHREIBUNG

Der Teilbaum 'teilbaum' wird in die 'teilbaum_liste' an der richtigen Stelle eingefügt. Zuvor wird jedoch überprüft, ob er die gestellten Kriterien erfüllt, welche Priorität er erhält, zu welchem Subagenten er gehört und ob bereits Teilbäume mit besserer Priorität existieren, die ihn überdecken. Nachdem er in die 'teilbaum_liste' aufgenommen wurde wird noch die ihm zugewiesene Priorität zurückgeschickt.

5.15 Die priority_such() Funktion

BEREICH
Teilbaum

SYNTAX

```
int priority_such(
    snmp_dpi_reg_packet      *teilbaum,
    int                      maxprio);
```

PARAMETER

teilbaum	Ein Zeiger auf die Struktur 'snmp_dpi_reg_packet', in der alle Daten für die Teilbaumanmeldung enthalten sind.
maxprio	Eine natürliche Zahl, die die Priorität angibt, bis zu welcher in der 'teilbaum_liste' gesucht werden soll. Bei der Suche wird bei der besten Priorität begonnen.

RETURNWERT
Folgende Werte sind möglich:

- 0, bei erfolgreicher Ausführung
- SNMP_ERROR_DPI_alreadyRegistered, falls der Teilbaum mit der gewünschten Priorität bereits in der 'teilbaum_liste' enthalten ist
- SNMP_ERROR_DPI_higherPriorityRegistered, falls der Teilbaum bereits mit einer besseren (niedrigereren) Priorität in der 'teilbaum_liste' enthalten ist

BESCHREIBUNG
Für den Teilbaum 'teilbaum' wird eine Priorität gesucht. Dabei kann der Teilbaum sich eine bestimmte Priorität wünschen, die nächst bessere wollen oder die beste anfordern. Wurde eine Priorität gefunden wird sie in seiner Datenstruktur '*teilbaum' abgelegt und die Funktion beendet.

5.16 Die teilbaeume_abmelden() Funktion

BEREICH
Teilbaum

SYNTAX

```
void teilbaeume_abmelden(
    snmp_dpi_subagent        *subagent);
```

PARAMETER

subagent	Ein Zeiger auf einen Subagenten in der 'subagenten_liste'.

RETURNWERT

void

BESCHREIBUNG

Es werden alle Teilbäume aus der 'teilbaum_liste' entfernt, die in dem Subagenten 'subagent' registriert sind. Das ist nötig, wenn sich ein Subagent abmeldet oder nicht mehr erreicht wird und noch Teilbäume von ihm registriert sind.

5.17 Die teilbaum_abmelden() Funktion

BEREICH

Registrierung

SYNTAX

```
int teilbaum_abmelden(
      snmp_dpi_ureg_packet      *teilbaum,
      int                       sd,
      struct sockaddr_in        *socketaddr);
```

PARAMETER

teilbaum	Ein Zeiger auf die Struktur 'snmp_dpi_ureg_packet', in der alle Daten für die Teilbaumabmeldung enthalten sind.
sd	Der Socketdeskriptor, falls das empfangene Paket ein TCP-Paket ist, sonst -1.
socketaddr	Ein Zeiger auf die Adresse des Absenders des Paketes, falls es ein UDP-Paket ist, sonst NULL.

RETURNWERT

Bei einer erfolgreichen Ausführung den Wert 0, sonst SNMP_ERROR_DPI_notFound, wenn der Teilbaum in der 'teilbaum_liste' nicht gefunden wurde.

BESCHREIBUNG

Der Teilbaum 'teilbaum' wird aus der 'teilbaum_liste' entfernt. Bevor man jedoch den gefundenen Teilbaum aus der 'teilbaum_liste löscht, ist es nötig zu überprüfen, ob dieser Teilbaum auch wirklich beim richtigen Subagenten angemeldet ist. Denn es ist auch möglich, daß ein Teilbaum mit gleicher OID bei einem anderen Subagenten registriert ist.

5.18 Die teilbaum_suche() Funktion

BEREICH
Teilbaum

SYNTAX

```
snmp_dpi_teilbaum *teilbaum_suche(
    snmp_dpi_teilbaum        *alter_teilbaum,
    oid                      *asn_oid,
    int                      *laenge,
    int                      exact);
```

PARAMETER

alter_teilbaum	Ein Zeiger auf einen Teilbaum, der bei der letzten Suche gefunden wurde bzw. NULL, wenn es die erste Suche ist.
asn_oid	Ein Zeiger auf die gesuchte Variablen-OID.
laenge	Ein Zeiger auf die Länge der gesuchten Variablen-OID.
exact	Dieser Wert gibt an, ob sich die Teilbaumsuche auf ein GET bzw. SET (1) oder GETNEXT (0) bezieht.

RETURNWERT
Konnte ein Teilbaum gefunden werden, wird ein Zeiger auf diesen zurückgegeben, sonst NULL.

BESCHREIBUNG
Es wird ein Teilbaum gesucht, der bei einem GET bzw. SET oder GETNEXT als nächstes in Frage kommt. Dabei müssen die Prioritäten berücksichtigt werden, was bei einer GET-Anfrage keine größeren Probleme bereitet. Mit Hilfe der Prioritäten ist es auch möglich in einem Teilbaum einen Unterteilbaum durch einen anderen zu ersetzen. Diese Möglichkei führt dazu, daß die Suche nach dem richtigen Teilbaum bei einer GETNEXT-Anfrage nicht mit einem Aufruf dieser Funktion abgetan ist. Erst durch mehrere Aufrufe dieser Funktion und das Schicken von DPI GETNEXT Paketen an die gefundenen Teilbäume ist die Bestimmung der richtigen Variable im richtigen Teilbaum möglich. Diese Funktion liefert also immer den als nächstbesten infragekommenden Teilbaum unter Berücksichtigung der Prioritäten, der Anfrageart und dem vorher gefundenen Teilbaum.

5.19 Die parse_var_op_list() Funktion

BEREICH
SNMP Paket bearbeiten

SYNTAX

```
int parse_var_op_list(
    unsigned char          *data,
    int                    length,
    unsigned char          *out_data,
    int                    *out_length,
    long                   *index,
    struct packet_info     *pi,
    int                    action);
```

PARAMETER

data	Ein Zeiger auf den Anfang der Variablenliste im SNMP Paket.
length	Die Länge des restlichen SNMP Paketes.
out_data	Beim Verlassen der Funktion ein Zeiger auf die Variablenliste des SNMP RESPONSE Paketes.
out_length	Beim Verlassen der Funktion ein Zeiger auf die Länge der Variablenliste des SNMP RESPONSE Paketes.
index	Beim Verlassen der Funktion ein Zeiger auf die Position der Variablen in der Variablenliste, welche einen Fehler verursachte.
pi	Ein Zeiger auf eine Struktur, in der alle wichtigen Informationen, die die Abarbeitung des aktuellen SNMP Paketes betreffen, enthalten sind.
action	Gibt an welcher Schritt der SET Aktion durchgeführt werden soll. Mögliche Werte sind RESERVE1, RESERVE2 wird ignoriert, FREE und COMMIT.

RETURNWERT

Konnten alle Aktionen erfolgreich durchgeführt werden, erhält man den Wert 'SNMP_ERR_NOERROR', ansonsten erhält man einen Wert der anzeigt, an welcher Stelle die Aktion scheiterte.

BESCHREIBUNG

Die Variablenliste aus dem SNMP Paket wird geparsed, die entsprechenden Werte zu jeder Variablen-OID werden geholt bzw. gesetzt und gleichzeitig wird ein SNMP RESPONSE Paket aufgebaut, mit den gefundenen Werten oder den entsprechenden Fehlermeldungen.

5.20 Die bulk_var_op_list() Funktion

BEREICH

SNMP Paket bearbeiten

SYNTAX

```
int bulk_var_op_list(
    unsigned char          *data,
    int                    length,
    unsigned char          *out_data,
    int                    *out_length,
    int                    non_repeaters,
    int                    max_repetitions,
    long                   *index,
    struct packet_info     *pi);
```

PARAMETER

data	Ein Zeiger auf den Anfang der Variablenliste im SNMP Paket.
length	Die Länge des restlichen SNMP Paketes.
out_data	Beim Verlassen der Funktion ein Zeiger auf die Variablenliste des SNMP RESPONSE Paketes.
out_length	Beim Verlassen der Funktion ein Zeiger auf die Länge der Variablenliste des SNMP RESPONSE Paketes.
non_repeaters	Die Anzahl der Variblen in der Variablenliste vom Start weg, von welchen nur ein Nachfolger gesucht ist.
max_repetitions	Die Anzahl der Nachfolger einer Variablen in der Variablenliste, die nicht zu den ersten 'non_repeaters' Variablen in der Variablenliste gehört.
index	Beim Verlassen der Funktion ein Zeiger auf die Position der Variablen in der Variablenliste, welche einen Fehler verursachte.
pi	Ein Zeiger auf eine Struktur, in der alle wichtigen Informationen, die die Abarbeitung des aktuellen SNMP Paketes betreffen, enthalten sind.

RETURNWERT

Konnten alle Aktionen erfolgreich durchgeführt werden erhält man den Wert 'SNMP_ERR_NOERROR', ansonsten erhält man einen Wert der anzeigt, an welcher Stelle die Aktion scheiterte.

BESCHREIBUNG

Die Variablenliste aus dem SNMP GETBULK Paket wird geparsed, die entsprechenden Werte zu jeder Variablen-OID werden geholt bzw. gesetzt und gleichzeitig wird ein SNMP RESPONSE Paket aufgebaut mit den gefundenen Werten oder den entsprechenden Fehlermeldungen.

5.21 Die get_anfrage() Funktion

BEREICH

SNMP Paket verarbeitung

SYNTAX

```
unsigned char *get_anfrage(
    oid                     *var_name,
    int                     *var_name_len,
    unsigned char           *statType,
    int                     *statLen,
    snmp_dpi_teilbaum       *teilbaum,
    int                     exact,
    char                    *fehlerart,
    unsigned long int       *fehlerindex);
```

PARAMETER

var_name	Ein Zeiger auf die gesuchte Variablen-OID und beim Verlassen der Funktion auf die gefundene Variablen-OID.
var_name_len	Ein Zeiger auf die Länge der gesuchten Variablen-OID und beim Verlassen der Funktion auf die Länge der gefundenen Variablen-OID.
statType	Beim Verlassen der Funktion ein Zeiger auf den Type des gefundenen Wertes.
statLen	Beim Verlassen der Funktion ein Zeiger auf die Länge des gefundenen Wertes.
teilbaum	Ein Zeiger auf den Teilbaum, in dem gesucht werden soll.
exact	Dieser Wert gibt an, ob es sich um eine GET (Wert = 1) oder um eine GETNEXT (Wert = 0) Anfrage handelt.
fehlerart	Beim Verlassen der Funktion ein Zeiger auf die Fehlerart, die beim Versuch des Lesens der Variablen auftrat.
fehlerindex	Beim Verlassen der Funktion ein Zeiger auf den Index der Variable, die beim Lesen scheiterte. Der Index ist 0, wenn kein Fehler auftrat, sonst 1 (da nur eine Variable pro Paket).

RETURNWERT

Folgende Werte sind möglich:

- ein Zeiger auf den Wert der Variable, bei erfolgreicher Ausführung
- NULL und statType = endOfMibView, falls bei GETNEXT das Ende des Teilbaumes erreicht wurde
- NULL, wenn ein Fehler auftrat

BESCHREIBUNG

Der Wert der Variablen 'var_name' wird vom Subagenten, in dem der Teilbaum 'teilbaum' registriert ist, geholt. Dabei wird entweder ein DPI GET oder DPI GETNEXT Paket geschickt, je nachdem wie 'exact' belegt ist. Anschließend wird auf das DPI RESPONSE Paket gewartet, aber höchstens die in 'teilbaum→timeout' angegebene Zeit. Von diesem RESPONSE Paket werden nun die vom Subagenten gelieferten Werte übernommen.

5.22 Die next_anfrage() Funktion

BEREICH

SNMP Paket verarbeitung

SYNTAX

```
unsigned char *next_anfrage(
    oid                    *var_name,
    int                    *var_name_len,
    unsigned char          *statType,
    int                    *statLen,
    struct packet_info     *pi);
```

PARAMETER

var_name	Ein Zeiger auf die gesuchte Variablen-OID und beim Verlassen der Funktion auf die gefundene Variablen-OID.
var_name_len	Ein Zeiger auf die Länge der gesuchten Variablen-OID und beim Verlassen der Funktion auf die Länge der gefundenen Variablen-OID.
statType	Beim Verlassen der Funktion ein Zeiger auf den Type des gefundenen Wertes.
statLen	Beim Verlassen der Funktion ein Zeiger auf die Länge des gefundenen Wertes.
pi	Ein Zeiger auf die Struktur, in der alle wichtigen Informationen zum SNMP Paket, das diese Aktion auslöste, stehen.

RETURNWERT

Konnte ein Wert gefunden werden, wird ein Zeiger auf diesen zurückgegeben, sonst NULL.

BESCHREIBUNG

Es wird der Nachfolger der Variablen 'var_name' in der MIB gesucht. Dabei müssen die Prioritäten der Teilbäume berücksichtigt werden. Mit Hilfe der Prioritäten ist es auch möglich Teilbäume teilweise zu überdecken oder in einem Teilbaum einen Unterteilbaum durch einen anderen zu ersetzen. Diese Möglichkeiten führen dazu, daß die Suche nach der richtigen Variablen bei einer GETNEXT-Anfrage nicht mit einem Aufruf der Funktion 'teilbaum_suche()' abgetan ist. Erst durch mehrere Aufrufe dieser Funktion und das Schicken von DPI GETNEXT Paketen an die gefundenen Teilbäume ist die Bestimmung der richtigen Variable möglich. Die von den verschiedenen Teilbäumen erhaltenen Variablen müssen dabei miteinander verglichen und die beste behalten werden. Man erhält schließlich eine Variable die der kleinste Nachfolger (in dieser MIB) von der gegebenen Variablen ist, unter Berücksichtigung der Prioritäten.

5.23 Die set_anfrage() Funktion

BEREICH

SNMP Paket verarbeitung

SYNTAX

```
int set_anfrage(
    oid                 *var_name,
    int                 *var_name_len,
    unsigned char       val_type,
    int                 val_len,
    unsigned char       *val_p,
    snmp_dpi_teilbaum   *teilbaum,
    int                 action,
    char                *fehlerart,
    unsigned long int   *fehlerindex);
```

PARAMETER

var_name	Ein Zeiger auf die zu setzende Variablen-OID.
var_name_len	Ein Zeiger auf die Länge der zu setzenden Variablen-OID.
val_type	Der Type des zu setzenden Wertes.
val_len	Die Länge des zu setzenden Wertes.
val_p	Ein Zeiger auf den zu setzenden Werte.
teilbaum	Ein Zeiger auf den Teilbaum, in dem gesucht werden soll.
action	Gibt an welcher Schritt der SET Aktion durchgeführt werden soll. Mögliche Werte sind RESERVE1, RESERVE2 wird ignoriert, FREE und COMMIT.

fehlerart	Beim Verlassen der Funktion ein Zeiger auf die Fehlerart, die beim Versuch des Setzens der Variable auftrat.
fehlerindex	Beim Verlassen der Funktion ein Zeiger auf den Index der Variable, die beim Beschreiben scheiterte. Der Index ist 0, wenn kein Fehler auftrat, sonst 1 (da nur eine Variable pro Paket).

RETURNWERT

Bei einer erfolgreichen Ausführung erhält man den Wert 0, sonst den Wert -1.

BESCHREIBUNG

Der Wert der Variablen 'var_name' wird beim Subagenten, in dem der Teilbaum 'teilbaum' registriert ist, gesetzt. Dabei wird ein DPI SET Paket geschickt, in dem die auszuführende Aktion (SET, UNDO, COMMIT) angegeben ist, je nachdem wie 'action' belegt ist. Anschließend wird auf das DPI RESPONSE Paket gewartet, aber höchstens die in 'teilbaum→timeout' angegebene Zeit. In diesem RESPONSE Paket wird nun kontrolliert, ob die Aktion erfolgreich verlief.

5.24 Die trap_weiterleiten() Funktion

BEREICH

SNMP Paket bearbeiten

SYNTAX

```
void trap_weiterleiten(
    snmp_dpi_trap_packet    *trap_packet,
    int                     sd,
    struct sockaddr_in      *socketaddr);
```

PARAMETER

trap_packet	Ein Zeiger auf eine Struktur, in der die Daten für den Trap abgelegt sind. Darin enthalten sind auch die Variablen, die mitgeschickt werden sollen.
sd	Der Socketdeskriptor, falls das empfangene Paket ein TCP Paket ist, sonst -1.
socketaddr	Ein Zeiger auf die Adresse des Absenders des Paketes, falls es ein UDP Paket ist, sonst NULL.

RETURNWERT

void

BESCHREIBUNG

Wenn der Agent ein DPI TRAP Paket von einem seiner Subagenten erhält, muß der Agent dieses Paket in ein SNMPv1 TRAP Paket verwandeln und an alle ihm bekannten TRAP-Ziele schicken. Die Trap-Ziele findet er in der Liste 'trap_ziele', in der alle Trapziele aus der Datei 'trap.conf' eingetragen sind. Der erste Eintrag in dieser Liste ist dabei der eigene Rechner, diese Daten werden gebraucht um im TRAP Paket den Absender anzugeben. An den ersten Eintrag in der Liste wird deshalb kein TRAP Paket geschickt.

5.25 Die my_getStatPtr() Funktion

BEREICH

SNMP Paket verarbeiten

SYNTAX

```
unsigned char *my_getStatPtr(
    struct subtree          *tp,
    oid                     *name,
    int                     *namelen,
    unsigned char           *type,
    int                     *len,
    unsigned short          *acl,
    int                     exact,
    int                     (**write_method)(),
    struct packet_info      *pi,
    int                     *noSuchObject);
```

PARAMETER

tp	Ein Zeiger auf einen internen Teilbaum, in dem die gesuchte Variable sein soll.
name	Ein Zeiger auf die gesuchte Variablen-OID und beim Verlassen der Funktion auf die gefundene Variablen-OID.
namelen	Ein Zeiger auf die Länge der gesuchten Variablen-OID und beim Verlassen der Funktion auf die Länge der gefundenen Variablen-OID.
type	Beim Verlassen der Funktion ein Zeiger auf den Type des gefundenen Wertes.
len	Beim Verlassen der Funktion ein Zeiger auf die Länge des gefundenen Wertes.
acl	Beim Verlassen der Funktion ein Zeiger auf einen Eintrag, der die Zugriffsrechte auf die Variable anzeigt.
exact	Dieser Wert gibt an, ob es sich um eine GET (Wert = 1) oder um eine GETNEXT (Wert = 0) Anfrage handelt.

write_method()	Beim Verlassen der Funktion ein Zeiger auf eine Funktion, die das Schreiben der Variablen ermöglicht. Allerdings nur wenn das Schreiben erlaubt ist.
pi	Ein Zeiger auf eine Struktur, in der alle wichtigen Informationen, die die Abarbeitung des aktuellen SNMP Paketes betreffen, enthalten sind.
noSuchObject	Beim Verlassen der Funktion ein Zeiger auf einen Eintrag, der anzeigt, ob die gesuchte Variable überhaupt in diesem Teilbaum existiert.

RETURNWERT

Bei Erfolg ein Zeiger auf den Wert der Variablen, sonst ein NULL-Zeiger.

BESCHREIBUNG

Diese Funktion ersetzt die 'old_getStatPtr()' Funktion im Bereich der SNMP Paketverarbeitung, in allen anderen Fällen wird auf die Funktion 'getStatPtr()' zurückgegriffen. Der Wert der Variablen 'name' wird von dem angegebenen internen Teilbaum geholt. Dabei wird auch die Zugriffsberechtigung überprüft und falls vorhanden die Schreibfunktion geliefert.

5.26 Die getStatPtr() Funktion

BEREICH

Agenteninterne MIB-Variable lesen

SYNTAX

```
unsigned char *getStatPtr(
    oid                 *name,
    int                 *namelen,
    unsigned char       *type,
    int                 *len,
    unsigned short      *acl,
    int                 exact,
    int                 (**write_method)(),
    struct packet_info  *pi,
    int                 *noSuchObject);
```

PARAMETER

name	Ein Zeiger auf die gesuchte Variablen-OID und beim Verlassen der Funktion auf die gefundene Variablen-OID.
namelen	Ein Zeiger auf die Länge der gesuchten Variablen-OID und beim Verlassen der Funktion auf die Länge der gefundenen Variablen-OID.

type	Beim Verlassen der Funktion ein Zeiger auf den Type des gefundenen Wertes.
len	Beim Verlassen der Funktion ein Zeiger auf die Länge des gefundenen Wertes.
acl	Beim Verlassen der Funktion ein Zeiger auf einen Eintrag, der die Zugriffsrechte auf die Variable anzeigt.
exact	Dieser Wert gibt an, ob es sich um eine GET (Wert = 1) oder um eine GETNEXT (Wert = 0) Anfrage handelt.
write_method()	Beim Verlassen der Funktion ein Zeiger auf eine Funktion, die das Schreiben der Variablen ermöglicht. Allerdings nur wenn das Schreiben erlaubt ist.
pi	Ein Zeiger auf eine Struktur, in der alle wichtigen Informationen, die die Abarbeitung des aktuellen SNMP Paketes betreffen, enthalten sind.
noSuchObject	Beim Verlassen der Funktion ein Zeiger auf einen Eintrag, der anzeigt, ob die gesuchte Variable überhaupt in diesem Teilbaum existiert.

RETURNWERT

Bei Erfolg ein Zeiger auf den Wert der Variablen, sonst ein NULL-Zeiger.

BESCHREIBUNG

Diese Funktion ersetzt die 'old_getStatPtr()' Funktion beim Lesen von MIB-Variablen. Der Wert der Variablen 'name' wird aus der MIB (sowohl Subagent als auch Agent) geholt. Diese Funktion wird jedoch nicht zur Berarbeitung von SNMP GET Paketen verwendet, sie dient nur zum agenteninternen Gebrauch.

5.27 Die antwort_abwarten() Funktion

BEREICH

Prüfen

SYNTAX

```
int antwort_abwarten(
    int             fd,
    int             time_out);
```

PARAMETER

fd	Der Socketdeskriptor, an dem auf Daten gewartet werden soll.
time_out	Die Zeit, wielange auf die Daten an dem Socketdeskriptor gewartet werden soll.

RETURNWERT

Folgende Werte sind möglich:

- DPI_RC_OK, wenn Daten am Socketdeskriptor anliegen
- 1, wenn innerhalb der angegeben Zeit keine Daten ankamen
- -1, wenn ein anderer Fehler auftrat

BESCHREIBUNG

Es wird am Socketdeskriptor auf ankommende Daten gewartet. Allerdings nur eine begrenzte Zeit, die festgelegt werden kann.

5.28 Die DPIawait_packet_from_subagent() Funktion

BEREICH

SNMP Paket bearbeiten

SYNTAX

```
int DPIawait_packet_from_subagent(
    snmp_dpi_subagent        *subagent,
    int                      timeout,
    char                     **message_p,
    int                      *length);
```

PARAMETER

subagent	Ein Zeiger auf den Subagenten, von dem ein Paket erwartet wird.
timeout	Die maximale Zeit, auf die ein Paket gewartet wird.
message_p	Beim Verlassen der Funktion ein Zeiger auf einen Zeiger, wo das empfangene Paket zu finden ist.
length	Beim Verlassen der Funktion ein Zeiger auf die Länge des empfangenen Paketes.

RETURNWERT

- DPI_RC_OK, bei erfolgreicher Ausführung
- DPI_RC_NOK, wenn ein undefinierter Fehler auftrat
- DPI_RC_INVALID_HANDLE, wenn ein ungültiger Socketdeskriptor übergeben wurde
- DPI_RC_TIMEOUT, wenn die Zeit überschritten wurde

BESCHREIBUNG

Es wird vom angegebenen Subagenten ein Paket erwartet. Das Paket wird in den Puffer abgelegt und die Länge des Paketes in der Variablen 'length'. Sollte jedoch die Zeit 'timeout' überschritten werden ohne das ein Paket empfangen wurde beendet sich die Funktion mit einer entsprechenden Fehlermeldung.

5.29 Die trans_type_2_agent() Funktion

BEREICH

Konvertierung

SYNTAX

```
unsigned char trans_type_2_agent(
    unsigned char              subagent_type);
```

PARAMETER

subagent_type	Die DPI-Typ-Konstante, der Typ einer Variablen, so wie er in DPI definiert ist.

RETURNWERT

Bei erfolgreicher Ausführung erhält man die SNMP-Typ-Konstante. Konnte der DPI Typ nicht in den SNMP Typ verwandelt werden, erhält man die SNMP-Typ-Konstante 'ASN_NULL'.

BESCHREIBUNG

Die Typen der Variablen sind in DPI und SNMP unterschiedlich definiert, deshalb ist es nötig die Typendefinitionen zu konvertieren. In dieser Funktion von DPI zu SNMP.

5.30 Die trans_type_2_subagent() Funktion

BEREICH

Konvertierung

SYNTAX

```
unsigned char trans_type_2_subagent(
    unsigned char              agent_type);
```

PARAMETER

agent_type	Die SNMP-Typ-Konstante, der Typ einer Variablen, so wie er in SNMP definiert ist.

RETURNWERT

Bei erfolgreicher Ausführung erhält man die DPI-Typ-Konstante. Konnte der SNMP Typ nicht in den DPI Typ verwandelt werden, erhält man die DPI-Typ-Konstante 'SNMP_TYPE_NULL'.

BESCHREIBUNG

Die Typen der Variablen sind in DPI (Subagenten) und SNMP (CMU-Agent) unterschiedlich definiert, deshalb ist es nötig die Typendefinitionen zu konvertieren. In dieser Funktion von DPI zu SNMP.

5.31 Die praefix() Funktion

BEREICH

Prüfen

SYNTAX

```
int praefix(
    oid     *name1,
    int     len1,
    oid     *name2,
    int     len2);
```

PARAMETER

name1	Ein Zeiger auf die Variablen-OID der ersten Variablen.
len1	Die Länge der ersten Variablen-OID.
name2	Ein Zeiger auf die Variablen-OID der zweiten Variablen.
len2	Die Länge der zweiten Variablen-OID.

RETURNWERT

Folgende Werte sind möglich:

- 0, wenn die erste Variablen-OID kein Präfix der zweiten ist
- 1, wenn die erste Variablen-OID ein echtes Präfix der zweiten ist
- 2, wenn die erste Variablen-OID Präfix der zweiten ist, d. h. die beiden Variablen-OIDs können auch gleich sein.

BESCHREIBUNG

Es wird geprüft, ob die erste Variablen-OID 'name1' ein Präfix von der zweiten 'name2' ist. Dabei wird noch zwischen einfachem Präfix und echtem Präfix unterschieden.

5.32 Die asn1_oid_2_string_oid() Funktion

BEREICH

Konvertierung

SYNTAX

```
char *asn1_oid_2_string_oid(
    oid                     *asn1_oid,
    int                     laenge);
```

PARAMETER

asn1_oid	Ein Zeiger auf eine Variablen-OID, so wie sie im Agenten gespeichert ist.
laenge	Die Länge der Variablen-OID.

RETURNWERT

Ein Zeiger auf den statischen Puffer 'string', in dem die Variablen-OID als nullterminierter String mit abschließendem Punkt abgelegt ist.

BESCHREIBUNG

Die Variablen-OID, so wie sie im Agenten gespeichert ist z. B. {1,3,6,1,0} mit der Länge 5, wird in einen String verwandelt "1.3.6.1.0.". Der abschließende Punkt wird nicht immer benötigt und muß deshalb bei Bedarf noch entfernt werden.

5.33 Die string_oid_2_asn1_oid() Funktion

BEREICH

Konvertierung

SYNTAX

```
void string_oid_2_asn1_oid(
    char                    *string_oid,
    oid                     *asn1_oid,
    int                     *laenge);
```

PARAMETER

string_oid	Ein Zeiger auf die Variablen-OID, welcher als String abgelegt ist.
asn1_oid	Beim Verlassen der Funktion ein Zeiger auf eine Variablen-OID, so wie sie im Agenten gespeichtert wird.
laenge	Die Länge der Variablen-OID.

RETURNWERT

void

BESCHREIBUNG

Die Variablen-OID, welche als String gespeichert ist, z. B. "1.3.6.1.0", wird in die für den Agenten akzepierte Form, {1,3,6,1,0} mit Länge = 5, konvertiert.

Kapitel 6

Kurzbeschreibung der DPI-Schnittstelle

Durch die Erweiterung des CMU-Agenten um die DPI-Schnittstelle hat sich der Funktionsumfang des Agenten um folgendes erweitert:

- Es können sich maximal 31 Subagenten über eine TCP-Verbindung beim Agenten registrieren.
- Über eine UDP-Verbindung können sich beliebig viele Subagenten registrieren.
- Die Subagenten können sogar auf einem anderen Rechner laufen.
- Die registrierten Teilbäume dürfen sich ganz oder auch nur teilweise überdecken, wird aber wegen schlechterer Laufzeit nicht empfohlen.
- Bei allen Anfragen wird die Priorität der Teilbäume berücksichtigt.
- SNMP GETBULK Pakete werden in DPI GETNEXT Pakete umgewandelt.
- Jeder Subagent muß sich anmelden, wobei er sich nicht zweimal anmelden kann.
- Jeder Teilbaum muß sich durch ein DPI REGISTER bzw. DPI UNREGISTER Paket beim Agenten an- bzw. abmelden.
- Beendet sich ein Subagent ohne sich vorher abzumelden wird er automatisch aus der Subagentliste entfernt, genauso seine Teilbäume in der Teilbaumliste.
- Bei einem SNNP Paket wird für jede Variable ein entsprechendes DPI Paket an den richtigen Subagenten gesendet, sofern die Variable in einem Subagenten implementiert ist.

- Ein DPI TRAP Paket, das auch mehrere Variablen mitliefern kann, wird an alle bekannten Managementstationen als SNMPv1 TRAP Paket weitergeleitet.
- Wird beim Starten des Agenten der DUMP-MODUS eingeschaltet, erhält man auch Meldungen von der DPI-Schnittstelle.
- Es wurde versucht die Implementierung der DPI-Schnittstelle möglichst modular zu halten und eine einigermaßen klare Schnittstelle zwischen der DPI-Schnittstelle und dem Agenten zu finden.

Kapitel 7

Schlußbemerkung

Diese Implementierung der DPI-Schnittstelle ist voll funktionsfähig, ausgenommen von INFORM-PDUs und TRAPv2 PDUs, welche auch in der RFC als zur Zeit noch nicht benützt gekennzeichnet sind.

7.1 Einbinden der DPI–Schnittstelle im CMU–Agenten

Es müssen folgende Files in das Verzeichnis "agent" des CMU–Agenten kopiert werden:

- dpi.h
- snmp_dpi.h
- snmp_lDPI.h
- dpi.c
- snmp_mDPI.c
- mib2.c

Außerdem müssen noch die Files

- snmpd.c
- snmp_vars.c
- snmp_agent.c

- snmp_vars.h
- Makefile

im Verzeichnis "agent" des CMU–Agenten durch die entsprechenden Files der DPI–Schnittstelle ersetzt werden. In diesen Files wurden kleine Veränderungen vorgenommen.

Abschließend sollte man noch sein 'mib.txt' File ersetzen, da einige Definitionen neu in dieses File aufgenommen wurden.

In manchen dieser Files befinden sich bereits Teile des Loggingmechanismus, auch das Makefile ist davon betroffen. Welche Files für den Loggingmechanismus noch benötigt werden lese man bitte in [4] nach. Es sollten unteranderem die Files 'loging.c', 'loging.h' und 'logfile.c' sein.

Nachdem nun alle nötigen Files in das Verzeichnis kopiert wurden, kann mit Hilfe von 'make' die Compilierung und das Linken gestartet werden und man erhält schließlich den ausführbaren CMU–Agenten 'snmpd' mit DPI–Schnittstelle und Loggingmechanismus.

7.2 Erweiterungsmöglichkeiten

Obwohl die Implementierung der DPI–Schnittstelle voll funktionsfähig ist, könnte man sie noch erweitern und verbessern. Zum einen um die Schnittstelle an neue Entwicklungen des DPI–Protokolls anzupassen und zum anderen um die Laufzeit des Agenten zu verkürzen und die Betriebssicherheit zu erhöhen.

- INFORM-PDUs
- TRAPv2 Pakete
- mehrere Variablen in einem DPI Paket
- Zeichensatzwahl
- Paßwort
- GETBULK weitergeben
- Communitystring an Subagenten mitgeben

Desweiteren wäre noch denkbar die DPI-Schnittstelle so zu erweitern, daß ein Subagent in der Lage wäre eine Variable beim Agenten zu lesen.

7.3 Lauffähige Subagenten

Die DPI–Schnittstelle des CMU–Agenten wurde mit dem Beispielsubagenten von IBM entwickelt und getestet. Sowohl der Orginalbeispielsubagent als auch verschiedene veränderte Formen dieses Subagenten liefen mit dem CMU–Agenten einwandfrei.

Ein zweiter, größerer Subagent, der als Fopra für das LRZ von Hauck und Haubelt entwickelt wird, arbeitet mit der DPI–Schnittstelle des CMU–Agenten problemlos zusammen.

Literaturverzeichnis

[1] William Stallings, *SNMP, SNMPv2 and CMIP - The Practical Guide to Network-Management Standards*, Addison–Wesley Co., Inc., Reading, MA, 1993

[2] B. Wijnen, G. Carpenter, K. Curran, A. Sehgal, und G. Waters, *Simple Network Management Protocol Distributed Protocol Interface Version 2.0*, RFC 1592, T.J. Watson Research Center, IBM Corp., Bell Northern Research, Ltd., März 1994

[3] Bert Wijnen, *DPI Version 2.0 API – Programmers Reference*, IBM International Operations, 1994

[4] Bastian Pusch, *Erweiterung eines SNMPv2-Agenten für Systemmanagementaufgaben*, Fortgeschrittenenpraktikum, TU–München, 1994

[5] J. Case, K. McCloghrie, M. RosE, und S. Waldbusser, *Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1448, SNMP Research, Inc., Hughes LAN Systems, Dover Beach Consulting, Inc., Carnegie Mellon University, April 1993

[6] J. Case, M. Fedor, M. Schoffstall, und J. Davin, *A Simple Network Management Protocol (SNMP)*, RFC 1157, SNMP Research, Performance Systems International, MIT Laboratory for Computer Science, Mai 1990

[7] J. Case, K. McCloghrie, M. RosE, und S. Waldbusser, *SNMPv2 RFCs (RFC 1441 bis RFC 1452)*, SNMP Research Inc, Hughes LAN Systems, Dover Beach Consulting Inc, Carnegie Mellon University, Trusted Information Systems, April 1993

[8] M. Rose, *SNMP MUX Protocol and MIB*, RFC 1227, Performace Systems International, Inc., Mai 1991