Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik IV
Prof. Dr. rer. nat. Otto Spaniol

SNMP - Simple Network Management Protocol

Seminar: Kommunikationsprotokolle Sommersemester 2003

Matthias Bätzold

Matrikelnummer: 235001

Betreuung: Karl-Heinz Krempels
Lehrstuhl für Informatik IV, RWTH Aachen

Inhaltsverzeichnis

1 Einführung

1.1 Entstehung und Entwicklung

Die Geschichte des SNMP hängt stark mit der Geschichte des Internets zusammen. In den achtziger Jahren boomte das Internet. Es wuchs mit immer schnellerer Geschwindigkeit und immer mehr private Haushalte und öffentliche Organisation waren mit dem Internet verbunden. Auch wurde die Anzahl der unterschiedlichen angeschlossen Geräte immer größer, je mehr Teilnehmer hinzukamen. Daraufhin setzten sich einige Leute zusammen, um ein Protokoll zu entwickeln, um das Internet effektiv zu nutzen und Ressourcen geschickt zu verteilen.
So entstand im Jahre 1987 das SGMP (Simple Gateway Monitoring Protocol), da zu damaliger Zeit das Verbinden der Gateways das Hauptproblem war. Gleichzeitig entstand noch ein anderes Protokoll mit dem Namen HEMS (High Level Managment Entity System), dessen Entwicklung schon länger zurückreichte. Allerdings fand dieses keine breite Unterstützung im Gegensatz zu dem SGMP, das schon zu dieser Zeit anfing, sich durchzusetzen. Ein weiterer Ansatz lag in einem OSI basierten Protokoll CMIP (Common Managment Information Protocol) das auf TCP Protokoll aufgesetzt werden sollte und somit den Namen CMOT erhielt (CMIP over TCP). Aufgrund der geringen Durchsetzung von HEMS wurden nur SGMP und CMOT weiterentwickelt, ersteres, weil es schon weit verbreitet war und zweiteres, weil es auf einem langen ISO-standardisierten Untergrund aufbaute. Später sollten beide zu einem Protkoll verschmelzen.
1988 brachten die Entwickler um SGMP das RFC 1065 *Structure of Managment Information* [MR88b], RFC 1066 *Managment Information Base* [MR88a] und RFC 1067 *Simple Network Managment Protocol* [CFSD88] heraus, was dann 1989 zu *recommended* erklärt wurde, welches einem quasi-Standard entspricht und ab hier auch schon den Namen SNMP trägt.
Die Entwickler von CMOT waren nicht so erfolgreich, denn sie schafften zwar 1989 den RFC 1095 *Common Managment Information Services and Protocol over TCP/IP* [WB89] zu veröffentlichen, allerdings gelang es ihnen nicht bis Mitte 1989 eine Implementierung zu erstellen.
Da die Entwickler beider Gruppen sich im Juni 1989 nicht auf eine gemeinsame Weiterentwicklung einigen konnten, entwickelten beide Gruppen weiter, wobei sich SNMP letztendlich durchsetzte und CMOT heute keine Bedeutung mehr hat. [uWS93]

Im Jahre 1990 wurde dann von den Entwicklern von SNMP das RFC 1155 *Structure of Managment Information* [RM90], RFC 1156 *Managment Information Base, MIB-I* [MR90] und RFC 1157 *Simple Network Management Protocol* [CFSD90] veröffentlicht. 1990/1991 ging die Verbreitung von SNMP kontinuierlich weiter und so wurde 1991 das RFC 1213 [MR91] herausgebracht, was die MIB-I auf MIB-II erweiterte und das RFC 1271 *Remote Network Monitoring MIB (RMON)* [Wal91], das die Funktionen im lokalen Netzwerk erweiterte. 1992 wurde dann eine neue Arbeitsgruppe gegründet, die sich mit der Entwicklung von SNMPv2 beschäftigte, da vor allen Dingen das Sicherheitskonzept Verbesserungen benötigte.

SNMPv2 war allerdings von seinem Sicherheitsstandard her zu komplex, so dass dies keine breite Zustimmung fand und so wurde 1996 SNMPv2 nur mit dem Sicherheitsmanagment aus SNMPv1 noch mal als RFC eingereicht, was dann SNMPv2c genannt wurde. Doch auch diese Lösung wurde nicht

als zufriedenstellend empfunden und so entstanden die beiden Standards SNMPv2u und SNMPv2*, die das Sicherheitsproblem lösen sollten.

Als letztes ist noch SNMPv3, RFC 2272-2275, zu erwähnen, das als Nachfolger von SNMPv2 zu verstehen ist, aber zusätzlich noch die Vereinigung von SNMPv2u und SNMPv2* bewirken soll [Bla97]. Diese wurde 1998 zum *Proposed Internet Standard* und dann 1999 zum *Draft Internet Standard*.

1.2 Ziele

Bei der SNMP-Entwicklung wurde sich an folgenden Zielen orientiert:

- es sollte leicht zu implementieren sein (*SIMPLE* Network Management Protocol), so dass auch kleinere Geräte in der Lage sind, dieses Protokoll zu benutzen ohne die Hardware großartig zu erweitern.
- es sollte systemunabhänging sein, da es ja auch in heterogenen Netzen, wie dem Internet, seine Anwendung finden sollte.
- auch sollte der Netzwerkdurchsatz durch die Protokoll-Architektur nicht zu sehr belastet werden, da dieses Protokoll ja nur zu Managementzwecken gedacht ist. Die verfügbare Netzwerkbandbreite sollte durch dieses Protokoll nicht wesentlich gemindert werden.

Als erstes wird auf allgemeine Grundlagen des Netzwerkmanagements eingegangen, damit ein grundlegendes Verständnis aufgebaut wird. Darauf folgen die Kapitel ASN.1 und SMI, in denen die Grundlagen der Datenbeschreibung und der Zugriff darauf unter SNMP erklärt werden. In Kapitel 5 wird die Struktur von SNMP dann genau beschrieben, in Kapitel 6 die Weiterentwicklungen in Form von SNMPv2 aufgezeigt und abschließend in Kapitel 7 die aktuellen Entwicklungen vorgestellt.

2 Netzwerk-Managment

2.1 Modell Manager/Agent

Als erstes wird hier auf die Grundlagen eines Netzwerk Managements eingegangen. Bei dem Netzwerk Management geht es darum, dass alle *managebaren* (was das heisst folgt später) und an dem Netzwerk angeschlossenen Ressourcen verwaltet und überwacht werden müssen. Dabei gibt es die Hardware Ressourcen wie Switches, Hubs, oder Repeater, so wie die Software Ressourcen, die gewisse Dienste bereitstellen. Zum Beispiel ein Gateway oder ein zentraler E-Mail-Server. Somit müssen alle Ressourcen den Netzwerk Manager mit Informationen versorgen und dieser dann, entsprechend den Informationen, reagieren. Somit lassen sich die übertragenen Informationen in Meldungen und

Kommandos unterteilen. Einmal Meldungen, die die Ressource über ihren Zustand oder über ein eingetretendes Ereignis an das Manager-System weiter gibt und die Kommandos, die ausgehend von dem Manager-System den Ressourcen Befehle erteilt.

Wie diese Kommunikation genau stattfindet, wird durch das Managementprotokoll definiert.
Dabei gibt es noch zwei Möglichkeiten, einmal das *Inband*- und das *Outband*verfahren. Beim ersteren werden die Management Informationen über das ganz normale Nutz-Datennetz ausgetauscht, beim zweiten Verfahren wird für die Management Informationen ein extra Netzwerk aufgebaut. Der Unterschied besteht also in der physikalischen Trennung von Nutz- und Management-Netzwerk. Im Regelfall wird allerdings das *Inband*verfahren benutzt, da keine Mehrkosten für die Installationen eines zweiten Netzwerkes entstehen. Das Managementprotokoll an sich baut in der Regel auf anderen Protokollen auf, wie zum Beispiel SNMP (bzw. SNMPv1) über UDP versendet wird, in dem eigentlichen Datenbereich des UDP-Paketes ist das SNMP-Paket eingebettet. Allerdings ist diese Festlegung nicht notwendig für ein Managementprotokoll, denn auch TCP wäre als Transportprotokoll für SNMP geeignet, doch man hat sich für die nicht-verbindungsorientierte Version entschieden.

Beim Netzwerk-Management ist das Manager/Agent Modell verbreitet. Die Agenten sind logische Einheiten, die mit dem Manager kommunizieren, wobei zu jeder *managebaren* Ressource ein Agent gehört. Somit erklärt sich auch der Begriff *managebar*, da in der Regel nur Ressourcen mit einem Agent gemanaget werden können. Auf der anderen Seite existiert dann der Manager, der alle Informationen von den Agenten sammelt und dementsprechend reagiert. Doch da nicht unbedingt alle Informationen, die dem Agenten zur Verfügung stehen, für den Manager von Interesse sind, bekommt dieser auch nur Informationen über den Teil der Ressourcen, die für ihn wichtig sind. Dieser Teil wird als *Managed Object* bezeichnet.

2.2 Managed Objects

Ein *Managed Object* besteht aus mehreren Eigenschaften. Einmal die Attribute, die für die Verwaltung der Ressource wichtig sind, dann die Operationen, die auf dem Object angewendet werden können und die Reaktionen darauf, so wie die Eigenschaft, von sich aus Meldung zu geben, wenn ein vorher definiertes Ereignis eintritt, dass so eine Meldung erfordert.
Ein Beispiel ist eine Zustandsänderung der Ressource oder eine Warnmeldung über aufgetretende Fehler. Irgendwie muss jetzt noch sichergestellt werden, dass alle Kommunikationsteilnehmer (also der Manager und alle Agenten) auf den gleichen *Managed Objetcs* arbeiten. Um das zu gewährleisten, bedienen sich bei Seiten aus einem Pool von Objekten, der *Management Information Base*, *MIB*. In einem Agenten können keine neuen Objekte erzeugt werden, es stehen nur die zur Verfügung, die durch die *MIB* in der Ressource des Netzwerkteilnehmers implementiert sind. Man beachte, dass der Agent ja nur ein kleines Programm ist, dass in der Netzwerk Ressource noch mitlaufen muss und somit nicht sehr komplex sein kann.

2.3 Management Information Base

Die *Management Information Base*, beschrieben in RFC 1066 [MR88a], ist eine komplette Definition von Objekten, die baumartig aufgebaut ist. Diese ist mitlerweile um die *MIB-II*, RFC 1213 [MR91] erweitert worden, die als Ergänzung zu *MIB-I* zu verstehen ist. Die *MIB* ist, eine baumartige Struktur von Objekten, die in einem anderen Teilbaum verwurzelt ist. Abgesehen von der Wurzel ist jeder Zweig mit einem eindeutigen Namen und mit einer Nummer versehen. Aus diesem Grund lässt sich jeder beliebige Ast auch eindeutig ansprechen. Entweder über die Nummer oder über die Namen, wobei die Äste durch Punkte voneinander getrennt werden. So ist zum Beispiel die Schreibweise 1.3.6.1 der Knoten mit dem Namen Internet der sich auch unter dem Namen iso(1).org(3).dod(6).internet(1) adressieren lässt. Uns interessieren nur die Objekte, die unter dem genannten Beispiel Ast "hängen". Denn unter internet.mgmt(2).mib-2(1) sind die Objekte der *MIB-II* zu finden. Ein weiterer interessanter Teilbaum ist unter internet.private(4).enterprises(1) zu finden, denn darunter können Unternehmen, an einem Unter-Knoten für sich ihre eigene private *MIB* erstellen. So findet sich beispielweise unter enterprises.SNI(231) die private *MIB* der Firma Siemens-Nixdorf. Die *MIB* hat nur die Funktion, eine Standardisierung aller benutzten Objekten zu geben. Denn so können alle Geräteherrsteller von Netzwerkressourcen diese mit einbinden und dann sicher sein, dass die anderen Netzwerkkomponenten mit den gleichen Objekten arbeiten.

3 ASN.1

3.1 Einleitung

Hier nun eine kleine Einführung zu ASN.1, da diese zur Beschreibung der *Managed Objects* benutzt wird. Allerdings wird hier nur soweit auf ASN.1 eingegangen, wie für SNMP notwendig.
ASN.1 *Abstract Syntax Notation One* ist eine Sprache zur Beschreibung von Daten. Das liegt daran, dass die Daten in verschiedenen Rechnern und auch in verschiedenen Progammiersprachen unterschiedlich dargestellt werden. Werden nun Daten von einer *Workstation* zu einem *Client* geschickt, so kann man diese nicht einfach senden, sondern muss sie in ein standardisiertes Format umwandeln. Der Empfänger kann dann mit Hilfe des Standards die Daten richtig zurückwandeln und dann interpretieren. Da über eine Leitung allerdings nur Bitfolgen übertragen werden können, müssen die ASN.1 Daten noch weiter umgewandelt werden. Dafür werden bei SNMP die *BER Basic Encoding Rules* benutzt, siehe Kapitel 3.3.
ASN.1 wird allerdings aus Gründen der Einfachheit nur mit einem abgespeckten Sprachumfang in SNMP eingesetzt. Wenn im Folgenden von ASN.1 gesprochen wird, so ist damit nur die für SNMP reduzierte Form gemeint.

3.2 Datentypen

ASN.1 kennt vier unterschiedliche *types*, nämlich *simple types*, *constructed types*, *tagged types* und *subtypes* (wohlgemerkt die von SNMP benutzte Form). Eine Zuweisung einer Variablen zu einem bestimmten Typ sieht so aus:

```
MyVar ::= INTEGER
```

Somit ist MyVar jetzt vom Typ INTEGER.

3.2.1 simple Types

ASN.1 benutzt folgende einfachen Datentypen:

INTEGER eine ganze Zahl ohne Angaben über die Genauigkeit

OCTET STRING ist eine Zeichenkette wobei jedes Zeichen aus genau 8 Bits besteht

OBJECT IDENTIFIER dient dazu, Objekte zu benennen

NULL ist nur ein Platzhalter, es zeigt das Fehlen jeglichen Typs an

3.2.2 constructed types

Von den *constructed types* werden aus ASN.1 nur SEQUENCE, SEQUENCE OF und CHOICE benutzt.

Mit SEQUENCE wird ein zusammengesetzter Datentyp definiert, ein Beispiel

```
Car ::= SEQUENCE {
        kW              INTEGER,
        farbe           OCTET STRING,
        besitzer        OBJECT IDENTIFIER
}
```

Mit SEQUENCE OF wird eine dynamische Liste von einem beliebigen Typ erstellt. Dieser Typ entspricht in anderen Programmiersprachen den Arrays. Beispiel:

```
Carport ::= SEQUENCE OF Car
```

Als letztes ist der Typ CHOICE anzugeben, der einen Variantentyp beschreibt:

```
CarColor ::= CHOICE {
        rgb             INTEGER,
        name            OCTET STRING
}
```

Hier kann man die Farbe des Autos entweder als RGB-Kodierung über eine ganze Zahl oder über den ausgeschrieben Farbennamen angeben.

3.2.3 tagged types

Unter Tag wird in ASN.1 eine Typkennung verstanden. Alle Tags sind entweder *universal, application, private* oder *context-specific*. Alle *simple types* und *constructed types* haben einen sogenannten Tag:

ASN.1-Typ	Tag
INTEGER	2
OCTET STRING	4
NULL	5
SEQUENCE, SEQUENCE OF	16

Diese vordefininierten Tags bezeichnet man als Tags der Klasse *UNIVERSAL*, es bedarf keiner expliziten Definition, damit diese gelten. Will man neue Typen erstellen, so kann diese von der Klasse *APPLICATION* ableiten zum Beispiel so:

```
MeinAuto ::= [APPLICATION 1] IMPLICIT OCTET STRING
```

Hiermit wird gesagt, dass MeinAuto eine andere Bezeichnung von *OCTET STRING* ist. Darauf zu achten ist, dass die Typenkennung *[APPLICATION 1]* nur einmal vorkommen darf! Eine weitere Klasse ist mit dem Namen *PRIVATE* bezeichnet, wird genauso benutzt wie *APPLICATION*, unterliegt aber nicht der gerade genannten Einschränkung von *APPLICATION*.

Als vierter *tagged type* ist die Typkennnung als *CHOICE* Aufbau, die dann folgendermassen aussieht:

```
Bauplan ::= CHOICE {
        [0] IMPLICIT NordSuedAusrichtung
        [1] IMPLICIT WestOstAusrichtung
}
```

Diese Tag-Klasse wird als *context-specific* bezeichnet und ist besonders wichtig bei den *Basic Encoding Rules*, ansonsten wird sie kaum benutzt.

3.2.4 subtypes

Diese Typen bestehen aus zwei Teilen, aus einem der drei erstgenannten Typen und einer Einschränkung. Hiermit lassen sich Typen bestimmen, die nur einen bestimmen Wertebereich haben. Zum Beispiel:

```
monat ::= INTEGER (1..12)
```

Somit können Variablen vom Typ Monat nur die Zahlen 1-12 annehmen.

3.3 ASN.1 Basic Encoding Rules

Die ASN.1 Datenbeschreibung dient dazu, Daten in eine wohldefinierte Form zu bringen. Die *Basic Encoding Rules BER* dienen jetzt dazu, die ASN.1 Daten als *Bytestring* darzustellen, so dass diese auch über das Netzwerk übertragen werden können. Auch hier wird nur kurz auf das Verfahren eingegangen, da dieses relativ schnell komplex wird und die Kodier- und Dekodierroutinen von heutigen ASN.1 Compilern automatisch miterzeugt werden. Der Aufbau von *BER* hat eine rekursive Struktur. Alle Objekte werden über nachstehende Regeln auf einfache Typen reduziert, so dass bei zusammengesetzten Typen die Regeln nur rekursiv anzuwenden sind.

Dabei werden Variablen über 3 Grössen definiert:

- -tag
- -length
- -value

weshalb auch von der TLV-Kodierung gesprochen wird. Das *tag*-Feld gibt die Typkennung der Variablen an, wodurch diese eindeutig bestimmt sind. *length* gibt die darauffolgende Länge des *value*-Feldes an und das *value*-Feld beinhaltet dann die eigentlichen Daten. Was noch anzumerken ist, ist in welcher Reihenfolge die Bits interpretiert werden und *BER* gibt an, dass das höchstwertige Bit mit der Nummer 8 bezeichnet und als erstes übertragen wird.

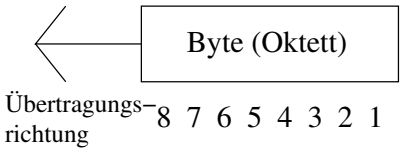

Abbildung 1: Die Übertragungsrichtung eines Bytes

Das erste Feld gibt den Tag-Typ und die Tag Klasse an, wobei die Definition dazu so aussieht, das Bit 7 und Bit 8 die Tag-Klasse angeben.

Tag-Klasse	Bit 8	Bit 7
UNIVERSAL	0	0
APPLICATION	0	1
Context-specific	1	0
PRIVATE	1	1

Bit 6 zeigt an, ob es sich um einen *simple typ* handelt (Bit 6 = 0) oder um einen *constructed type* (Bit 6 = 1) In Bit 1-5 werden dann die Typkennungswerte von 1-30 dargestellt, reicht dieses Feld nicht aus, so wird dieses um ein Byte erweitert.

Das *length*-Feld gibt an, wie lang das nachfolgende *value*-Feld in Bytes ist. Reicht ein Byte nicht aus, um das zu kodieren, dann gibt das erste Byte an, wie lang das Feld ist, das die Länge des *value*-Feldes kodiert. Die eigentliche Länge des *value*-Feldes steht damit dann in den Feldern 2 bis x (je nach der Kodierung im ersten Feld). Um diese Fälle zu unterscheiden, wird das Bit 8 des *length*-Feldes genutzt. Ist dieses auf 1 gesetzt, so gibt das erste Byte nur die Länge des *length*-Feldes an, ist dieses hingegen 0, so ist das erste Byte die Kodierung für die Länge des *value*-Feldes. Man beachte, dass bei dem Fall Bit 8 = 0 nur eine *value*-Feld Länge von 1-126 möglich ist, da die Kodierung für 127 reserviert ist und Bit 8 ja eine andere Funktion erfüllt.

In dem *value*-Feld steht dann der Wert der Variablen, entweder kein Wert, einer (bei primitven Datentypen) oder mehrere (bei *constructed types* zum Beispiel) Werte.

4 SMI

4.1 Einleitung

Im folgenden wird SMI beschrieben (RFC 1155 *Structure and Identification of Management Information* [RM90]). Ziel diese Standards bzw. Verfahrens ist es, allgemein eine gültige Definition der Objektklassen für die *MIBs* zu erstellen, ohne jedoch konkrete Objekte zu definieren.

Dazu schauen wir uns noch mal den Registrierbaum an, in dem die *MIB* (*MIB-II*) untergebracht ist (siehe Abbildung 2).
Zu beachten ist, dass der Knoten *MIB-II* selber kein *Managed Object* ist, sondern nur die Knoten darunter. Interessant sind noch die zu *mib2* parallelen Knoten *experimental* und *private*. Letzter wird vor allen Dingen durch seinen Unterknoten *enterprises*, wie schon in 2.3 erwähnt, interessant. Wir beschäftigen uns aber erst mal nur mit dem Teilbaum unter *mib2*. Dadurch haben alle Knoten den gleichen Namesanfang, nämlich 1.3.6.1.2.1 oder iso(1)org(3)dod(6)internet(1)mgmt(2)mib2(1) oder am einfachsten {mgmt 1}. Somit kann man über diese Adressierung auch auf die Objekte zugreifen. Allerdings gilt das nur für die Objekte und nicht für deren Instanzen. Wie man darauf zugreift wird jetzt gezeigt.

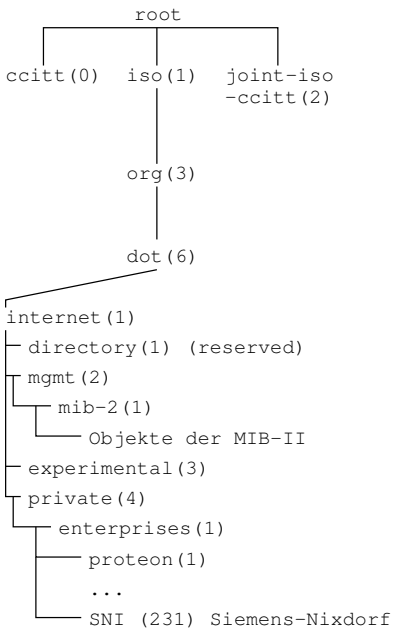

Abbildung 2: Der Registrierbaum

4.2 Datentypen

SMI kennt drei unterschiedliche Datentypengruppen: *primitive*, *constructor* und *definite types*. Primitive Typen sind die *simple types* aus ASN.1, also INTEGER, OCTET STRING, OBJECT IDENTIFIER und NULL. Unter *constructor types* versteht man die zusammengesetzten Typen SEQUENCE, SEQUENCE OF und CHOICE aus ASN.1. Die *definite types* sind vordefinierte Typen, deren gibt es 6:

NetworkAddress: Dies ist ein allgemeiner Typ, der eine Netzwerkadresse darstellt, eigentlich war er gedacht, alle möglichen Arten von Netzwerkadressen darzustellen, in SMI wir allerdings nur die IP Variante definiert.

IpAddress: 4-Byte-Internetadresse

Counter: 32-Bit Zähler, startet bei 0 und dann zählt dann *nur* hoch, am Ende seines Wertebereiches fängt er wieder bei 0 an.

Gauge: 32-Bit Zähler, der aber in beiden Richtungen laufen kann und am Ende seines Wertebereiches nicht wieder bei 0 anfängt.

TimeTicks: 32-Bit Wert, repräsentiert eine Zeitspanne, angegeben in 1/100 Sekunden

Opaque: Zur Verwendung privater, nicht genormter Daten, die auf einen OCTET STRING abgebildet werden.

4.3 Managed Objects

Nun geht es darum, wie man denn nun auf ein *Managed Object* zugreifen kann. Dafür muss man erstmal wissen, wie so ein *Managed Object* aufgebaut ist. Es besteht aus 5 Feldern: Object, Syntax, Definition, Access, Status.

Object gibt eine besser lesbare Beschreibung des Objektes ab. Beispiel: das Objekt 1.3.6.1.2.1.11.29 hätte hier in seinem Object Feld *snmpOutTraps* stehen, ein eindeutig besser lesbarer Namen. Die Namen im Registrierbaum (wir hier der Name snmpOutTraps) sind in dem Baum alle eindeutig, dafür sorgt die *Internet Assigned Number Authority*, die diese Namen vergibt.

Syntax beschreibt die Datenstruktur des Objektes unter zu Hilfe Name der ASN.1 Regeln.

Definition gibt eine textuelle Beschreibung des Objektes und Hinweise zu der Nutzung der Variabeln an.

Access gibt die Zugriffsrechte auf das Objekt an. Möglich sind *read-only, read-write, write-only* und *not-accessible*.

Status gibt den Status des Objektes an. Mögliche Werte sind *mandatory* für zwingend erforderlich, *optional* für nicht erforderlich, *obsolete* für veraltet und *deprecated* als Vorstufe von *obsolete* und somit als Hinweis, dass diese bald veraltet sein wird.

Als Beispiel das Object sysName:

OBJECT sysName {system 5}

Syntax DisplayString(SIZE(0..255))

Definition Dieses Objekt ist der von dem Administrator Name zugewiesene Name des Managed Node (Kapitel 5.1). Standardmässig wird hier der *fully-qualified domain name* eingetragen.

Access read-write

Status mandatory

Da diese Auflistung für die automatische Weiterverabeitung nicht unbedingt so geeigent ist, gibt es ein ASN.1 Makro, das OBJECT-TYPE Makro, das diese Form ”übersetzt”. Danach sieht das Beispiel von so aus:

```
sysName OBJECT-TYPE
        SYNTAX DisplayString(SIZE(0..255))
        ACCESS read-write
        STATUS mandatory
        ::= {system 5}.
```

Es stehen hier also die gleichen Informationen, nur das Feld Definition mit der Beschreibung fehlt. Die vollständige Beschreibung einer *MIB* in dieser Form wird auch als *MIB-Module* bezeichnet.
Um beide Beschreibungsformen anzugeben, sind in der *MIB-I*, RFC 1156, auch alle Beschreibungen doppelt angegeben.

Will man jetzt auf ein *Managed Object* zugreifen, so sind hier zwei Fälle zu unterscheiden. Einmal gibt es von einem Objekt nur eine Instanz, dann ist der Zugriff relativ einfach. In dem Fall erfolgt der Zugriff nämlich über den eindeutigen Objektnamen, an den dann einfach eine Null angehängt wird. Bei dem Beispiel mit dem *sysName* - Objekt von vorhin sieht die Adressierung dann so aus: 1.3.6.1.2.1.1.5.0 oder einfach nur {sysName 0}.
Gibt es mehrere Instanzen ein Objektes, so ist das Objekt als Tabelle definiert, wobei jede Spalte vom gleichen Typ ist. Da erfolgt der Zugriff dann über die Angabe des eindeutigen Objektnamens, dann die Nummer der Spalte und dann die Indexnummer des gewünschten Eintrages. Am besten lässt sich das an einem Beispiel zeigen:

Objekttype	ifIndex	...	ifSpeed	...	ifSpecific
Feld Nr.	1	...	5	...	22
Interface Nr.					
1	1		9600		0
2	2		64000		0
3	3		1200		0
.					
.					
.					

Diese Interface Tabelle ist unter 1.3.6.1.2.1.2.2. zu finden bzw. {interfaces 1}. Will man jetzt auf die Interface-Geschwindigkeit des 3 Interfaces zugreifen, so würde die Adressierung so aussehen:

1.3.6.1.2.1.2.2.2.1.5.3 und zwar gibt die vorletzte Zahl die Spalte an, in dem der Eintrag zu finden ist und die letzte Zahl den Index-*Wertes* des Tabellenindizes, wobei der Index dieser Tabelle die Spalte ifIndex ist.

5 Details über SNMP

5.1 NMS / MN

Das SNMP Protokoll basiert auch auf dem in Kapitel 2.1 beschriebenen Manager/Agent Modell. Auch hier gibt es Agenten, *Managed Node MN* genannt und den Manager, *Network Management Station NMS* genannt. *NMS* und *MN* tauschen Managementinformationen über das Managementprotokoll SNMP aus und zwar bezogen auf die Attribute ihrer *Managed Objects*.

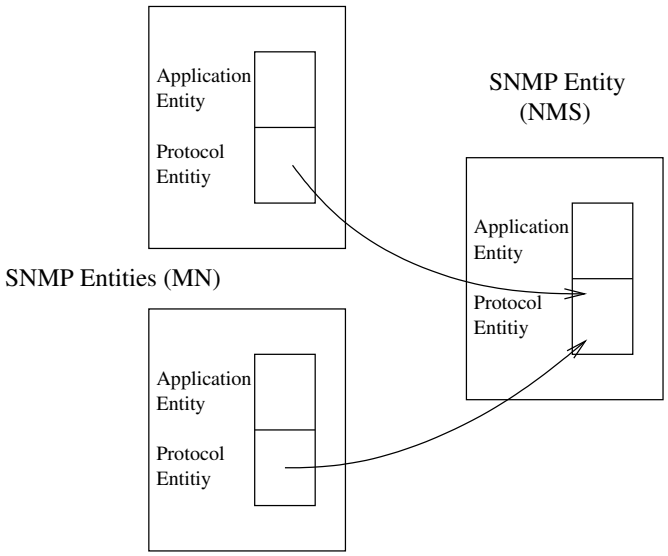

Abbildung 3: Die Kommunikation zwischen SNMP-Entities

Dabei bilden beide Einheiten eine sogenannte *SNMP-Entitiy*, bestehend aus einer *SNMP Application Entity* und einer *SNMP Protocol Entity*. Die Application Entity übernimmt die Aufgabe des Agenten bzw. des Managers, die *Protocol Entity* wickelt die Kommunikation über das SNMP Protokoll ab. Abbildung 3 verdeutlicht diesen Aufbau.
Zwischen den beiden *Entities* besteht ein Managementdienst, der in den RFCs aber nicht definiert ist und somit dem Programmierer überlassen bleibt.

5.1.1 Proxy-Agents

Das Netzwerk-Management mit SNMP kennt noch einen zusätzlichen Typ von Agenten, den Proxy-Agenten. Wie oben schon beschrieben gibt es auch nicht *managebare* Netzwerkressourcen, sei es, weil diese zu alt sind und das SNMP Protokoll deshalb noch nicht kennen oder weil sie ein anderes Protokoll unterstützen. Hier wird dann ein Proxy-Agent eingerichtet, der dann auf einer anderen Netzwerk-Ressource läuft und als Agent für die eigentlich nicht managebare Ressource agiert. Der Proxy-Agent kommuniziert dann direkt mit der Ressource in einem Protokoll, das diese versteht und gibt diese Informationen dann SNMP-konform an das *NMS* weiter. Auch empfängt er die Anweisungen von dem *NMS* und gibt diese an die Ressource so weiter, das diese die verarbeiten kann. Somit sind die nicht-managebaren Einheiten auch in das Netzwerk eingebunden und das *NMS* sieht noch nicht einmal einen Unterschied, da sich für ihn der Proxy-Agent nicht von dem normalen Agenten unterscheidet.

5.2 Managementkommunikation

5.2.1 Protokoll

Das Protokoll der Wahl ist UDP. Allerdings wird den in RFC 1157 ([CFSD90]) darauf hingewiesen, dass das Protokoll nicht festgelegt ist, es darf nur nicht verbindungsorientiert sein, wie zum Beispeil das TCP Protokoll. Allerdings ist SNMP bis jetzt hauptsächlich nur über UDP realisiert worden. Das Ganze sieht dann so aus:

IP-Header	UDP-Header	SNMP-Header	SNMP-PDU

In der PDU *Protocol Data Unit* stehen dann die eigentlichen Management Anweisungen.

5.2.2 Operationen

In seiner Grundversion (SNMPv1) gibt es nur 4 Befehle: GET, GETNEXT, SET und TRAP. Die ersten drei Befehle sind Befehle, die das NMS an den Agent schickt um entweder Informationen von ihm zu bekommen (GET,GETNEXT) oder Werte eines *Managed Object* zu setzen (SET). Der dritte Befehl (TRAP) wird spontan von dem Agenten initiiert, der dem *NMS* damit meldet, dass sich an der Netzwerkressource etwas geändert hat und das *NMS* darauf unter Umständen reagieren muss. Die Trap-Meldung beinhaltet nicht, was sich verändert hat, sondern nur, dass sich was verändert hat, das *NMS* muss den Zustand der Ressource erst wieder mit GET abfragen. Aussehen einer SNMP-PDU:

PDU-Typ
request-id INTEGER
errorstatus noError(0) tooBig(1) noSuchName(2) badValue(3) readonly(4) genErr(5)
error-index INTEGER
variable-bindings Liste von {Objektname, Wert }

In dem PDU-Typ wird der Typ der Nachricht angegeben, zum Beispiel 0 für die GET-Operation. *request-id* ist eine Id, anhand derer die Response-Nachricht nachher eindeutig zugeordnet werden kann. Das nächste Byte gibt den Errorstatus an, bei 0 ist alles in Ordnung, ansonsten einer der Fehler 1-4, Fehler 5 ist für aufgetretene Fehler, wo die Fehlerursache unbekannt ist. Error-index gibt an, wenn zum Beispiel ein *readonly* - Fehler vorliegt, also versucht wurde eine Variable zu schreiben, die nur *read-only* ist, bei welcher Variable dieser Fehler aufgetreten ist und zwar bezogen auf die Position der Variablen, die in der *variable-bindings* Liste mit übergeben werden. In der *variable-bindings* Liste stehen die übergebenen Variablen drin, bei der SET Operation die, die gesetzt werden sollen und bei der GET und GETNEXT Operation die, die ausgelesen werden sollen. Die *variable-bindings* Liste definiert mehrere Variablen, wovon jede einzelne durch den eindeutigen Objektnamen und den Wert angegeben ist. Der eindeutige Objektname wird am Ende von 4.3 erklärt.
Ich will hier noch kurz den Unterschied von GET und GETNEXT erklären. Die GET Operation fragt den Wert von Variablen ab. Dafür muss allerdings eine genaue Kenntniss der *MIB* vorhanden sein, wenn man sich das Beispiel von 4.3 anschaut und bedenkt, dass das nur eine kleiner Teil der ganzen *MIB* ist, dann ist einem klar, dass das ziemlich komplex ist. Um somit den Zugriff zu vereinfachen, wurde die GETNEXT Operation eingeführt, die einfach das nächste Objekt in der logischen Reihenfolge abfragt. Somit ist diese Operation auch besonders für Tabellen gut geeignet.
Die Trap-Nachricht hat einen etwas anderen Aufbau als die Nachrichten GET, GETNEXT und SET. Da dieser Unterschied allerdings in SNMPv2 aufgehoben wurde, wird hier nicht darauf eingegangen.

In Version SNMPv1 gilt noch die ”Alles oder Nix” Praxis. Wenn über eine SET Operation mehrere Variablen gesetzt werden sollen, und eine Variablenbesetzung davon einen Fehler zurückgibt, so wird die ganze SET Operation verworfen und *keine* Variable wird gesetzt.

Das SNMP-Protokoll ist ein *asynchrones symmetrisches Request-Response* Protokoll, das bedeutet:

- *Request* und *Response* sind nicht synchronisiert, es können mehrere *Requests* gestellt werden, ohne das ein *Response* zurückgekommen ist. Identifiziert werden die *Requests* über einen *request identifier*, der in der *Response* Nachricht auch vorhanden ist.
- Nachrichten können verloren gehen (zum Beipiel die Trap-Nachricht), es ist dann Aufgabe der *Application Entities*, dies zu erkennen und entsprechend zu handeln
- die Nachrichten werden ”spontan” gesendet, es muss nicht erst eine Kommunikationsverbindung aufgebaut werden, bevor Nachrichten ausgetauscht werden.

5.2.3 Sicherheit

Um eine gewisse Sicherheit zu gewährleisten werden die *Application Entities* in *Communities* unterteilt und jede *Communtity* hat einen eigenen Namen. Wird nun eine Nachricht von einer *SNMP-Entity* zu einer anderen geschickt, wird der *Community* Name mit übertragen und anhand dieses Namens kann der Empfänder dann prüfen, ob die Nachricht für ihn bestimmt ist. Ansonsten verwirft er diese. Ist der *Community* Name richtig, so spricht man auch von eine *authentic SNMP message*. Dies

ist auch der Grund, warum in SNMPv2 die Sicherheit verbessert werden sollte, da der *Community* Name unverschlüsselt als Teil der SNMP-Nachricht mit übertragen wurde, was eine missbräuchliche Anwendung recht einfach macht.

6 SNMP v2

6.1 Ziele

Die Entwicklung von SNMPv2 sollte eine kontinuierliche Weiterentwicklung von SNMPv1 darstellen, vor allen Dingen sollten die Probleme und Schwachstellen die bei SNMPv1 aufgetreten sind behoben werden. Dabei wurden folgende thematischen Neuerungen in SNMPv2 realisiert:

- Kommunikationsmodell für SNMP Protocol Entities
- Sicherheitskonzepte
- Kommunikation zwischen den Managerstationen
- Erweiterung des Agent-Konzeptes
- Einführung einer *MIB* für SNMP Protocol Entities
- Übertragung grösserer Informationsmengen in einer PDU
- Erweiterung der Fehlersignalisierung und der Traps
- Erweiterungen zum Structure of Management Information (SMI)
- Verwendung unterschiedlicher Transportdienste

6.2 Kommunikationsmodell für SNMP Protocol Entities

Das Kommunikationsmodell wurde dahingehend verändert, dass die miteinander kommunizierenden *SNMP Protocol Entities* (in SNMPv2 nur noch *SNMP Entities* genannt) nun in *parties* eingeteilt werden. Diese *parties* verständigen sich dann über eine verschlüsselte Übertragung, was vorraussetzt, dass die *parties* alle das gleiche Verschlüsselungsprotokoll benutzen. Damit muss jede *SNMP-Entity* eine lokale Datenbasis mit den *parties* führen, die bekannt sind. Nachrichten von unbekannten *parties* werden verworfen.
Die zweite Veränderung im Kommunikationsmodell ist die Einführung des Begriffes *context*. Diese definiert einen Teilausschnitt aus der *MIB, MIB View* genannt, die für die anderen *SNMP Entities* sichtbar sind. Der Kommunkationspartner kann so nur die Werte verändern und auslesen, die er auch ”sehen” kann.
Beides, *parties* und *context*, müssen im SNMPv2-Header bei jeder Nachricht mit angegeben werden.

6.3 Sicherheitskonzepte

Um das Sicherheitskonzept zu erweitern, werden nun DES und MD5 als Algorithmen benutzt, um zu verhindern, dass jemand die Nachricht liest, der dazu nicht bestimmt ist, oder jemand eine Nachricht schickt, der nicht dazu berechtigt ist.

6.4 Kommunikation zwischen den Managerstationen

In SNMPv2 wurde die Manager/Manager-Kommunikation, die in SNMPv1 nicht definiert war, eingeführt. Dabei sendet die Managerstation, bei Eintreten eines bestimmten Ereignisses, eine Nachricht an eine andere Managerstation. Die Funktionsweise ist wie bei einer TRAP-Nachricht. Allerdings hat die erzeugte Nachricht einen anderen Typ, der in SNMPv2 neu definiert wurde und *inform-request* heißt. Der adressierte Manager antwortet dann mit einem *response-request*.

6.5 Erweiterung des Agentenkonzeptes

Eine schon erwähnte Erweiterung liegt in der Manager/Manager Kommunikation, denn diese *SNMP Entity* arbeitet einmal als Manager und einmal als Agent, nämlich dann, wenn es den anderen Managern den *inform-request* sendet. Eine weitere Erweiterung sollte darin bestehen, dass die Agenten Informationen von den untergelagerten Ressourcen einholen sollten. Ein diesbezüglicher Ansatz, RFC 1227 SNMP SMUX Protocol and MIB [Ros91], ist nie über den experiementellen Status hinausgekommen.

6.6 Einführung einer MIB für SNMP Protocol Entities

Durch die Einführung von *parties* und *context* sind die Anforderungen an eine *SNMP Protocol Entity* gewachsen, so das in RFC 1450 *Management Information Base for SNMPv2* [CMRW93a] definiert wird, wie man diese Einheiten geeigent überwachen kann.

6.7 Übertragung grösserer Datenmengen

War am Anfang von SNMPv1 noch der getnext- Befehl als äußerst effektiv erachtet worden, so stellte sich mit der Zeit heraus, dass bei größeren Tabellen die Abfrage sehr aufwendig war. Aufgrund dessen, dass für jeden Tabelleneintrag ein neuer *getnext-request* auszuführen war, was auch eine erhebliche Netzlast erzeugte, die aber einem Grundziel von SNMP widerspricht (s. Kapitel 1.2). So wurde der *getbulk* Befehl eingeführt, der mehrere Daten auf einmal hintereinander abfragen kann. So konnte die Netzlast durch diesen Befehl stark verringert werden, es wurden Performanceverbesserungen um den Faktor 10 gegenüber getnext beobachtet [uWS93].

6.8 Erweiterung der Fehlersignalisierung

In SNMPv1 galt noch, wenn ein Fehler auftritt, beispielsweise bei einer SET Operation, dann wird das ganze SNMP Paket nicht gesetzt und eine Fehlerkennung zurückgeliefert. In SNMPv2 ist das dahingehend verändert worden, dass nur die fehlerverursachenden Variablen nicht bearbeitetet werden und alles was keine Fehler verursacht, ausgeführt wird.

6.9 Erweiterungen zum Structure of Management Information (SMI)

IM OBJECT-TYPE-MACRO können jetzt vier neue Datentypen benutzt werden:

- Standard-ASN.1-Datentyp BIT STRING
- ein 64-Bit langer Zähler (Counter64)
- ein vorzeichenloser INTEGER Typ UInteger32
- ein Datentyp für OSI-Netzadressen, sogenannte NSAPs

Das zusätzliche Schlüsselwort "UNITS" erlaubt jetzt eine textuelle Beschreibung der Einheiten, in denen die Objektvariable angegeben wird.
Das Schlüsselwort ACCESS wird jetzt durch MAX-ACCESS ersetzt und kann Werte in der folgenden Reihenfolge annehmen: *not-acessible, read-only, read-write, read-create*.
Das Schlüsselwort Status kann jetzt nur noch die Werte *current, deprecated* und *obsolete* annehmen, mandatory und optional entfallen.

6.10 Verwendung unterschiedlicher Transportdienste

In SNMPv1 wurde nur UDP als Transportdienst definiert, es wurde nur von der Möglichkeit gesprochen, weitere Dienste zu benutzen. In SNMPv2, RFC 1449 *Transport Mappings for SNMPv2* [CMRW93b], sind jetzt auch weitere Dienste spezifiziert worden. Diese sind:

- - UDP
- - OSI Connectionless Transport Service (CLTS) [Ros93]
- - Appletalk DDP [MR93]
- - Novell IPX [Bos93]

Allerdings wird SNMP über UDP weiter als "*preferred mapping*" hevorgehoben.

6.11 Koexistenz von SNMPv1 und SNMPv2

Da SNMPv1 und SNMPv2 nicht kompatibel sind, weil zum Beispiel in SNMPv1 die *parties* und der *context* nicht mitangegeben wird, kann man alte SNMPv1 Komponenten nur dadurch integrieren, dass man einen Proxy-Agenten benutzt. Ein Proxy-Agent läuft dann auf einer anderen Ressource, kommunziert mit dem Agenten via SNMPv1 und übersetzt dessen Pakete dann in SNMPv2 und schickt diese an die entsprechenden *SNMP-Entities* weiter. Somit können die alten Ressourcen weitergenutzt werden.

6.12 Versionen

Wie schon in der Einleitung beschrieben, entwickelten sich mehrere Versionen von SNMPv2. Das lag vor allem daran, dass sich das Sicherheitskonzept von SNMPv2 nicht durchsetzen konnte und so wurde SNMPv2c (c für *classic*) entwickelt, das die Neuerungen von SNMPv2 enthielt, aber noch auf dem alten Sicherheitskonzept von SNMPv1, den *Communities*, basierte. Das ursprünglich standardisierte SNMPv2 wurde auch SNMPv2p (p für *parties*) genannt. Als letztes entwickelte sich noch SNMPv2* und SNMPv2u, die beide auf SNMPv2c basieren, hinsichtlich *Remote Administration* und *Security* aber verbessert wurden.

7 Aktuellen Entwicklungen / Ausblick

Nach der Versionsteilung von SNMPv2 und dem damit verbundenen Misserfolg wurde 1996 eine neue Arbeitsgruppe gegründet, die SNMPv3 entwickelt und die vorhandenen Probleme beseitigen soll. Allerdings sollte die Kompatibilität zu den Vorgängerversionen gewährleistet werden.

7.1 Kommunikation

Die Befehle *get, getnext, getbulk, trap* und *inform* bleiben in SNMPv2 unverändert. Allerdings wird die ursprüngliche *SNMP Protocol Entitiy* in *SNMP Engine* umbenannt und hat jetzt mehr Aufgaben. Sie besteht aus dem *Dispatcher*, der für das eigentliche Versenden zuständig ist, dem *Message Processing Subsysytem* [CHPW98], das alle 3 SNMP Formate erkennt und in das SNMPv3 umwandeln kann, dem *Security Subsystem* und dem *Access Control Subsystem*, die sich um die Authentifizierung und die Zugriffskontrolle kümmern.
Wichtig sind hier auch die Begriffe der *Subsysteme*. Dies verdeutlicht den Aufbau, denn diese Subsysteme sind modular aufgebaut und alle Module müssen ein spezielles *Interface* implementieren. Dies hat den Vorteil, dass später entdeckte Sicherheitsmängel durch den Austauch eines Modules behoben werden können.

7.2 Sicherheit

Die Sicherheit wird durch mehrere Submodule gewährleistet.

7.2.1 User Base Security Model

[BW98] Das *User Based Security Model* basiert darauf, dass bei jeder Nachricht ein bestimmter Teil an Userinformationen mit übertragen werden muss, so dass nur der Empfänger die Nachricht lesen kann. Dazu muss der User auch auf dem Empfangssystem bekannt sein. Anhand der Userinformation kann der Empfänger dann mit Hilfe der verschiedene Sicherheitsmodule das Paket prüfen und bestimmte Operationen zulassen.

7.2.2 Authentication Submodul

Hier wird auch eine Authentifizierung anhand des Users vorgenommen. Allerdings wird aus der Userinformation und dem Paketinhalt ein *Key* generiert, der an das Paket angehängt wird. Am Ziel wird mit Hilfe des Paketes und des auch dort bekannten Users der *Key* noch mal generiert und die Übereinstimmung wird geprüft.
Als Key erzeugende Verfahren sind folgende Verfahren vorgesehen, einmal MD5 [Riv92], das unterstützt werden muss und SHA, was unterstützt werden sollte.

7.2.3 Timeless Submodul

Hier wird bei jeder Nachricht ein Zeitstempel, wie lange entsprechende Ressource seit dem letzten neu starten betriebsbereit ist, mitgeschickt. Auf der Empfängerseite wird der Zeitstempel überprüft, sollte der Wert sich extrem von den anderen bisher erhaltenen Werten unterscheiden, wird das Paket verworfen, es sei denn, die Werte beginnen wieder bei Null, dann hat nur ein Neustart des Gerätes stattgefunden.

7.2.4 Privacy Submodul

In diesem Modul wird sichergestellt, das kein anderer als der Empfänger die Nachricht lesen kann, da sie hier mit DES verschlüsselt wird.

7.2.5 View-based Access Control Model

[WPM98] Das View-based Access Control Model stellt Funktionen bereit, mit denen sich überprüfen lässt, ob eine Funktion auf eine *MIB*-Variablen zulässig ist oder nicht. Vor jedem Zugriff wird diese Überprüfung durchgeführt und die angeforderte Funktion ausgeführt oder nicht. Die Information, wer was darf, wird über Gruppen realisiert. Jeder User kann nur in einer Gruppe sein und die Gruppe hat genau spezifizierte Rechte auf einem *MIB*-Objekt.

8 Zusammenfassung

Die heutigen Entwickler von Netzwerkressourcen haben es schwer. Sie haben die Auswahl zwischen einem ziemlich alten Protokoll, dem SNMPv1, welches bekannte Mängel aufweist und nicht mehr auf dem aktuellesten Stand ist. Dann gibt es noch verschiedene Varianten von SNMPv2. Doch gerade die Auswahl an Protokollen ist keine Hilfe, denn der Hersteller will ja Ressourcen produzieren, die mit möglichst vielen anderen Komponenten zusammenarbeiten. Und SNMPv3 ist noch relativ jung, so dass man noch nicht weiss, ob sich das durchsetzen wird. Bleibt nur abzuwarten, inwiefern SNMPv3 eine breite Unterstützung findet. Allerdings ist der Hersteller auf der sicheren Seite, wenn er SNMPv1 implementiert, da SNMPv3 durch den Standard das schon unterstützt und SNMPv2 Komponenten mit Hilfe eine Proxy-Agenten diese auch unterstützen können, wobei da die bekannten Probleme von SNMPv1 in Kauf genommen werden.
Alles in allem bleibt die Marktdurchdringung von SNMPv3 abzuwarten, die Entwicklung scheint weitesgehend abgeschlossen, denn im Dezember 2002 sind die SNMPv3 Dokumente noch mal überarbeitet eingereicht worden und zwar RFC 3411 - RFC 3418 ([HPW02], [CHPW02], [LMS02], [BW02], [WPM02], [PE02c], [PE02b], [PE02a]).

Literatur

[Bla97] Darryl P. Black. Managing Switched Local Area Networks - A Practical Guide. November 1997.

[Bos93] S. Bostock. SNMP over IPX, RFC1420. March 1993.

[BW98] U. Blumenthal and B. Wijnen. User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3), RFC2274. January 1998.

[BW02] U. Blumenthal and B. Wijnen. User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3), RFC3414. December 2002.

[CFSD88] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol, RFC1067. August 1988.

[CFSD90] J.D. Case, M. Fedor, M.L. Schoffstall, and C. Davin. Simple Network Management Protocol (SNMP), RFC1157. May 1990.

[CHPW98] J. Case, D. Harrington, R. Presuhn, and B. Wijnen. Message Processing and Dispatching for the Simple Network Management Protocol (SNMP), RFC2272. January 1998.

[CHPW02] J. Case, D. Harrington, R. Presuhn, and B. Wijnen. Message Processing and Dispatching for the Simple Network Management Protocol (SNMP), RFC3412. December 2002.

[CMRW93a] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2), RFC1450. April 1993.

[CMRW93b] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Transport Mappings for version 2 of the Simple Network Management Protocol (SNMPv2), RFC1449. April 1993.

[HPW02] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks, RFC3411. December 2002.

[LMS02] D. Levi, P. Meyer, and B. Stewart. Simple Network Management Protocol (SNMP) Applications, RFC3413. December 2002.

[MR88a] K. McCloghrie and M.T. Rose. Management Information Base for network management of TCP/IP-based internets, RFC1066. August 1988.

[MR88b] K. McCloghrie and M.T. Rose. Structure and identification of management information for TCP/IP-based internets, RFC1065. August 1988.

[MR90] K. McCloghrie and M.T. Rose. Management Information Base for network management of TCP/IP-based internets, RFC1156. May 1990.

[MR91] K. McCloghrie and M.T. Rose. Management Information Base for Network Management of TCP/IP-based internets:MIB-II, RFC1213. March 1991.

[MR93] G. Minshall and M. Ritter. SNMP over AppleTalk, RFC1419. March 1993.

[PE02a] R. Presuhn and Ed. Management Information Base (MIB) for the Simple Network Management Protocol (SNMP), RFC3418. December 2002.

[PE02b] R. Presuhn and Ed. Transport Mappings for the Simple Network Management Protocol (SNMP), RFC3417. December 2002.

[PE02c] R. Presuhn and Ed. Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP), RFC3416. December 2002.

[Riv92] R. Rivest. The MD5 Message-Digest Algorithm, RFC1321. April 1992.

[RM90] M.T. Rose and K. McCloghrie. Structure and identification of management information for TCP/IP-based internets, RFC1155. May 1990.

[Ros91] M.T. Rose. SNMP MUX protocol and MIB, RFC1227. May 1991.

[Ros93] M. Rose. SNMP over OSI, RFC1418. March 1993.

[uWS93] Rainer Jansen und Wolgang Schott. SNMP: Konzepte - Verfahren - Plattformen. April 1993.

[Wal91] S. Waldbusser. Remote Network Monitoring Management Information Base, RFC1271. November 1991.

[WB89] U.S. Warrier and L. Besaw. Common Management Information Services and Protocol over TCP/IP (CMOT), RFC1095. April 1989.

[WPM98] B. Wijnen, R. Presuhn, and K. McCloghrie. View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP), RFC2275. January 1998.

[WPM02] B. Wijnen, R. Presuhn, and K. McCloghrie. View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP), RFC3415. December 2002.