Performance Monitoring eines Netzwerks über SNMP mittels WWW-Frontend

Diplomarbeit an der Universität Ulm

Fakultät für Informatik

Rainer Bawidamann

Performance Monitoring eines Netzwerks über SNMP mittels WWW-Frontend: Diplomarbeit an der Universität Ulm; Fakultät für Informatik

von Rainer Bawidamann

Inhaltsverzeichnis

Abbildungsverzeichnis

Beispiele

Zusammenfassung

Netzwerk-Geräte stellen für das Netzwerk-Management ihre Konfiguration und statistische Daten über das *Simple Network Management Protocol* (SNMP) als SNMP-Objekte bereit. Diese Objekte stellen die *Management Information Base* dar und sind in einzelnen MIB-Modulen definiert. Einen Überblick über die Objekte verschaffen MIB-Browser. Mit dieser Arbeit wurde ein MIB-Browser entwickelt, der als Frontend einen WWW-Browser verwendet und damit Client-seitig plattformunabhängig ist und ohne Installation auf jedem Rechner im Netzwerk benutzt werden kann. Als zweites wurde ein Programm entwickelt, das die Statistik-Daten der Netzwerk-Geräte periodisch sammelt und graphisch darstellt. Die Ausgabe der Daten erfolgt wiederum auf einem WWW-Browser. Die Daten werden in unterschiedlichen Auflösungen gespeichert, so daß die Statistiken über mehrere Jahre geführt werden können, ohne zuviel Plattenplatz zu belegen.

Danksagung

Für die Unterstützung bei dieser Arbeit möchte ich mich - in loser Folge - bedanken bei: Meinem Betreuer Karl Gaissmaier, der die Idee zu dieser Arbeit hatte und mir mit Rat und Kritik zu Seite stand; meinem Erst-Prüfer Professor Dr. Hans-Peter Großmann, der mir diese Diplomarbeit ermöglichte und nach einem Großbrand trotz Platznot einen neuen Arbeitsplatz beschaffte; meinem Zweit-Prüfer Professor Dr. Michael Weber; Klaus Espenlaub für den mir unbürokratisch zur Verfügung gestellten Arbeitsplatz und -Rechner, während das Rechenzentrum kurz nach dem Brand kaum Räume zur Verfügung hatte; den anderen Mitarbeitern des Rechenzentrums, die für Fragen jederzeit zu sprechen waren und mich sonst in Ruhe arbeiten ließen; schließlich meiner Freundin für ihre Geduld, da ich in den letzten Wochen recht wenig Zeit für sie hatte.

Kapitel 1. Aufgabenstellung

1.1. Motivation

Die Administration von Computer-Netzwerken kann in folgende Aufgabengebiete eingeteilt werden:

- Fault Management

  Erkennen, Lokalisieren und Beheben von Fehlern

- Accounting Management

  Zugriffsprotokollierung und Abrechnung von Ressourcen

- Configuration and Name Management

  Konfiguration von Netzwerkkomponenten

- Performance Management

  Messung der Auslastung, finden möglicher Engpässe

- Security Management

  Zentrale Verwaltung und Verteilung von Paßwörtern, Zugriffskontrolle und Protokollierung

Diese Arbeit konzentriert sich auf die Punkte Fault Management und Performance Management.

Bei der Auswahl einer Netzwerkkomponente trifft man in der Regel einen Kompromiß zwischen Preis und Leistung. Die Anforderungen an die Leistung ändern sich jedoch mit der Zeit, eine Kontrolle ist nötig: Weder soll die Komponente ungenutzt „herumstehen", noch soll sie einen Engpaß darstellen, der verhindert, daß die Leistung der anderen Komponenten ausgenutzt werden kann. Dies stellt - im wesentlichen - das Performance Management dar.

Ein Engpaß kann auch durch einen Ausfall oder eine Fehlfunktion eines anderen Gerätes oder eines Dienstes entstehen. Während ein (Total-) Ausfall meist recht schnell lokalisiert werden kann, führt eine Fehlfunktion oder ein Teilausfall in einem großen Netzwerk zu oft obskuren Effekten. Antwortet z. B. der Name-Server eines Netzes (zuständig für die Umwandlung von symbolischen Netznamen in numerische Netzadressen) sehr langsam oder gar falsch, scheint das ganze Netz langsam zu sein, oder Teile des Netzwerks, evtl. Teile des externen Netzwerks (Internet), sind nicht mehr erreichbar. Sucht man den Fehler an den Grenze zum nicht-erreichbaren Netz, wird man keine Fehler finden. Ohne Hilfe durch Programme ist die Suche nach dem Fehler sehr mühsam. Hat man ein Protokoll der durchschnittlichen Antwort-Zeiten und Antwort- und Fehler-Raten des Name-Servers, würden in diesem Fall Anomalien in diesen Protokollen auftreten. Diese Protokolle und Statistiken erlauben so ein schnelleres Auffinden von Fehlerquellen, was das Hauptproblem des Fault Management ist.

Alle aktiven Netzwerkkomponenten (mit Ausnahme einfachster Modelle) unterstützen für diesen Zweck das „Simple Network Management Protocol" SNMP, über das Parameter wie z. B. die Menge übertragener Daten als „Variablen" oder „Objekte" abgefragt werden können. Auch für Dienste wie z. B. Name-Service sind SNMP Implementierungen verfügbar, allerdings müssen diese oft separat installiert werden oder sind über zusätzliche Software realisiert.

Diese Variablen oder Objekte sind in sog. „MIB-Modulen“ (Management Information Base) mit Name, Typ, kurzer Beschreibung usw. definiert. Die Definitionen sind in einer Untermenge der Beschreibungssprache „Abstract Syntax Notation One“ (ASN.1, ISO Standard 8824) kodiert.

SNMP-Variablen enthalten in der Regel einfache Zähler: Anzahl gesendeter Pakete, übertragener Bytes, aufgetretener Fehler. Komplexere Werte wie Übertragungsrate, Fehlerrate oder Auslastung eines Interfaces oder Netz-Segmentes sind nicht vorgesehen. Dies liegt daran, daß die Netzwerkkomponenten und Dienste in erster Linie Daten übertragen oder Anfragen beantworten sollen. Einen Zähler hochzuzählen kostet wenig Ressourcen und kann nebenher erledigt werden. Die ständige Berechnung der aktuellen Interface-Auslastung würde das Gerät stärker belasten und könnte die Leistung des Gerätes einschränken. Bei der Erstellung von Statistiken kann es deshalb erforderlich sein, mehrere SNMP-Variablen abzufragen und daraus einen Wert zu berechnen.

1.2. Ausgangsbasis: MRTG

Für Performance und Fault Management einiger Komponenten des Netzwerks der Universität Ulm wird am Rechenzentrum bislang der „Multi Router Traffic Grapher“ [MRTG] eingesetzt. MRTG bietet die Möglichkeit, zwei SNMP-Variablen eines Netzwerk-Gerätes abzufragen, abzuspeichern und graphisch darzustellen.

1.2.1. Grenzen von MRTG

Es gibt jedoch einige Einschränkungen: Es müssen immer zwei SNMP-Variablen pro Gerät sein. Das Meßintervall ist auf fünf Minuten festgelegt. Nach jeder Messung werden die Daten-Log-Dateien, die HTML-Seiten und die Graphiken neu berechnet. Dadurch dauert das Update recht lange, vor allem wenn viele Geräte protokolliert werden. In einer Installation dauerte der komplette Update-Zyklus etwas mehr als 2 Minuten, also fast die Hälfte des Meßintervalls!

MRTG erlaubt es, Statistiken über die Kombination einiger SNMP-Variablen (Summe, Differenz, Produkt, Quotient) zu erzeugen. Die einzelnen Variablen der Kombination werden jedoch nicht separat gespeichert, sondern nur der Ergebnis-Wert. Mehrfach verwendete SNMP-Variablen werden dabei für jede Kombination getrennt geholt, auch bei mehrfacher Verwendung in einer Kombination (z.B. um den relativen Anteil einer Variablen an der Summe beider zu zeigen: `rel = var1 / (var1 + var2)`). Dies verursacht natürlich eine unnötige Netzbelastung.

Die Suche nach der richtigen SNMP-Variablen wird von MRTG überhaupt nicht unterstützt und ist in der Mailing-Liste zu MRTG [MRTG-L] die meist gestellte Frage.

1.3. Teil 1: Der MIB-Browser

Da diese MIB-Module schwer zu lesen sind, nimmt man bei der Auswahl der geeigneten SNMP-Variablen sog. „MIB-Browser“. Diese zeigen die SNMP-Variablen in einer übersichtlichen Form, meist als Baum an. Viele erhältliche MIB-Browser laufen nur auf einer Plattform, sind also nur an wenigen Stellen im Netzwerk verfügbar. Um diese Einschränkung aufzuheben, soll mit dieser Diplomarbeit ein WWW-basierter MIB-Browser entwickelt werden. Der MIB-Browser soll Client-seitig die gängigen WWW-Browser unterstützen, damit er auf jedem Rechner im Netzwerk eingesetzt werden kann. Eine Plattformunabhängigkeit soll auch bei der Server-Komponente angestrebt werden, damit Server-seitig ein Wechsel der Plattform leicht möglich ist.

Der MIB-Browser soll neben der übersichtlichen Darstellung des Objekt-Baums auch die Möglichkeit bieten, die aktuellen Werte von SNMP-Variablen ausgewählter Netzwerkkomponenten abzufragen und evtl. auch neu zu setzen.

1.4. Teil 2: Netzwerk-Statistiken

Hat man die SNMP-Variablen gefunden, die die gewünschten Informationen bereitstellen, reicht es oft nicht aus, den Wert nur einmal abzufragen. Erst eine wiederholte Abfrage, z. B. einmal pro Minute, ergibt die gewünschten Informationen wie den durchschnittlichen Datenverkehr auf einem Netzwerk, also die Auslastung.

Der 2. Teil der Diplomarbeit behandelt zwei Programme, von denen das erste die Sammlung und Speicherung der Daten übernimmt. Das zweite zeigt die Daten aufbereitet in einem WWW-Browser an, also wie beim MIB-Browser Client-seitig plattformunabhängig.

Ein Schwerpunkt ist die Darstellung berechneter („abgeleiteter“) Werte, z. B. die Paket-Fehlerrate (Anzahl fehlerhafter Pakete im Verhältnis zur Anzahl übertragener Pakete) oder die Kollisionsrate (Anzahl Kollisionen zu Anzahl übertragener Pakete). Da SNMP nur einfache Zähler bereit stellt, müssen diese Werte aus mehreren Variablen berechnet werden. Ein Beispiel nach [SimpleTimes]:

Ein Netzwerk-Gerät wie ein Router oder ein Switch bietet pro Interface die Zähler (Variablen) `ifInOctets` und `ifOutOctets` für die Anzahl empfangener bzw. gesendeter Bytes an. Außerdem enthält die Variable `ifSpeed` die Geschwindigkeit des Interfaces in Bits pro Sekunde. Um die Auslastung des Interfaces innerhalb eines Intervalls berechnen zu können, müssen die Variablen-Werte zu Beginn und zu Ende des Intervalls abgefragt werden (nennen wir sie `ifInOctets[start]` und `ifInOctets[end]`). Die Auslastung `utilization` ergibt sich dann als

```
utilization = ( 8 * (   ifInOctets[end] - ifInOctets[start]
                      + ifOutOctets[end] - ifOutOctets[start]
                     )
               ) / ifSpeed
```

(`ifSpeed` enthält die Interface-Geschwindigkeit in Bit/Sekunde während `ifInOctets` und `ifOutOctets` Bytes zählt, deshalb die Multiplikation mit 8).

Dies wird von den erhältlichen kommerziellen Software-Paketen nicht oder nur mangelhaft unterstützt.

Ein weiterer Schwerpunkt ist die effiziente Abfrage und Speicherung der Meßwerte: Bei berechneten Werten verwendete SNMP-Variablen sollen nur einmal abgefragt werden, auch wenn sie in mehreren Berechnungen verwendet werden. In einem Jahr summieren sich diese Meßwerte auf etwa ein Megabyte, bei 100 Routern oder Switches und jeweils 2 bis 24 Interfaces kommen schnell einige Gigabyte an Daten zusammen. Auch dieses Problem wird von kommerziellen Lösungen nicht befriedigend gelöst.

1.5. Mögliche Erweiterungen

1.5.1. Kurzzeitstatistiken

Mit dem MIB-Browser kann der aktuelle Werte einer SNMP-Variablen abgefragt werden. Bei Zählern ist ein einzelner Wert jedoch wenig aussagekräftig. Denkbar ist, die Funktionen des Daten-Sammlers zur Abfrage und Anzeige in den MIB-Browser zu integrieren, so daß eine SNMP-Variable für kurze Zeit protokolliert und graphisch angezeigt werden kann. Das Verhalten des Zählers wird dadurch anschaulich. Dies kann bei der Konfiguration (bei der Installation oder bei Fehlern) nützlich sein, in dem die Auswirkung von Konfigurationsänderungen beobachtet werden können.

1.5.2. Meldung von Ausfällen

Die Erkennung von (Teil-) Ausfällen durch Anomalien in Statistiken erfordert nach wie vor manuelles Eingreifen durch den Netzwerk-Administrator. Da meist die Zeit fehlt, ein- oder gar mehrmals täglich sämtliche Statistiken durchzusehen, muß eine Fehlfunktion erst gemeldet werden, bevor an Suche und Behebung gegangen wird.

Während das Statistik-Programm Daten sammelt, stellt es natürlich fest, wenn ein Gerät ausgefallen ist. Dies könnte z. B. durch eine Mail dem Administrator gemeldet werden.

Kapitel 2. Ausgangssituation

2.1. Existierende MIB-Browser

Die meisten größeren Netzwerk-Management-Programme enthalten auch einen MIB-Browser. Diese sind jedoch nicht als WWW-Lösung implementiert, höchstens als (langsame) Java-Version. An der TU Braunschweig wurde ein WWW-basierter MIB-Browser entwickelt, der auf einer Tcl-Erweiterung basiert [SBROWSER]. Dieser Browser zeigt wahlweise eine Liste der auf dem Server vorliegenden MIB-Module oder die SNMP-Variablen eines Moduls an. Der Tcl-Browser bietet weder eine kombinierte Anzeige der Variablen aller vorliegenden MIB-Module noch eine Baumdarstellung von SNMP-Variablen.

2.2. SNMP

Das „Simple Network Management Protocol" (SNMP) stellt die Grundlage dieser Arbeit dar. Deshalb folgt hier eine Einführung in SNMP. Für die Benutzung von SNMP gibt es bereits fertige Bibliotheken. Auf diese Bibliotheken wird anschließend eingegangen.

Eine umfassende Einführung in SNMP bietet [SNMPWS].

2.2.1. Historie

Bei der Entwicklung von Netzwerkgeräten wurde schnell erkannt, daß eine Möglichkeit nötig ist, die Verwaltungsinformationen der Geräte (Status, Statistiken), über das Netzwerk „remote" warten zu können. Die ersten Geräte verfügten nur über interaktive Zugänge oder proprietäre Protokolle. In einer heterogenen Umgebung mußte ein Administrator so ein Sammelsurium von Programmen und Menüsystemen (bei telnet-Zugängen) bedienen können.

Im Jahr 1988 entwickelte eine Working Group der IETF SNMP. Im August 1988 wurde die erste RFC zu SNMP veröffentlicht (RFC 1067). Inzwischen gibt es mehrere Versionen des Protokolls. Die erste Version, ist in RFC 1157 [RFC], einer Neufassung von RFC 1067 definiert (auch: STD 0015). Die zweite Version führte Security-Funktionalität (Authentifizierung und Verschlüsselung) und einen neuen Abfrage-Typ ein, wurde aber meist nur in einer abgespeckten Form als Version 2c („Community based SNMP") realisiert. Version 2c kombiniert die Authentifizierung von Version 1 mit dem Protokoll von Version 2. Version 2 ist in RFC 1905 beschrieben, Version 2c in RFC 1901.

Mit SNMP Version 3 wurde erneut der Versuch gemacht, Authentifizierung und Verschlüsselung einzubauen: Version 3 führt ein „User-based security model" und ein „View-based access model" ein. Version 3 ist in den RFCs 2570 bis 2576 beschrieben. Viele weitere RFCs enthalten Ergänzungen und Erweiterungen.

Praktische Relevanz haben zur Zeit nur die Version 1 und 2c.

2.2.2. Daten in SNMP

2.2.2.1. Struktur der Verwaltungsinformationen

In SNMP werden die Verwaltungsinformationen als Sammlung von Objekten („managed objects“) betrachtet. Diese Objekte formen eine virtuelle Informationsdatenbank, genannt „Management Information Base“ (MIB). Mengen von verwandten Objekten sind in MIB-Modulen definiert. Diese Module sind in einer Teilmenge der „Abstract Syntax Notation One“ (ASN.1, ISO Standard 8824) geschrieben, genannt „Structure of Management Information“ (SMI, definiert in RFC 1155 für SNMP Version 1 und in RFC 2578 für SNMP Version 2 und 3).

Die Objekte sind über ihren „Object Identifier“ (OID) hierarchisch organisiert. Jedes Objekt ist entweder ein Blattknoten mit Instanzen („object instance“), die die Verwaltungsinformationen beinhalten, oder ein Zwischenknoten, der dazu dient, die Blattknoten zu gruppieren und eine Hierarchie zu bilden.

2.2.2.2. Kodierung der Objekte

Objekt IDs und Daten werden gemäß den „Basic Encoding Rules“ (BER, ISO Standard 8825) kodiert. Diese Kodierung ist unabhängig von der Darstellung von Daten auf den verschiedenen Hardware-Plattformen (Big Endian vs. Little Endian, 32 vs. 64 Bit Architekturen).

2.2.2.3. Datentypen

SNMP kennt einfache, abgeleitete und zusammengesetzte Datentypen.

- SNMP kennt zum einen einige Basis-Datentypen, die den ASN.1 „primitive types“ INTEGER, OCTET STRING, OBJECT IDENTIFIER, NULL entsprechen. Diese werden auch mit „non-aggregate types“ bezeichnet.
- Von diesen „primitive types“ können weitere Datentypen abgeleitet werden. Diese Typen unterscheiden sich evtl. nur durch ihre Bedeutung. Z. B. ist Counter ein INTEGER mit dem Zusatz, daß der Wert monoton steigt, bis ein Maximum erreicht ist. Dann erfolgt ein Überlauf und der Wert beginnt wieder bei 0.
- Ein abgeleiteter Datentyp kann auch den Wertebereich einschränken (z.B. bei TCP-Ports, die nur Werte von 0 bis 65535 annehmen können), oder Aufzählungstypen definieren. Bei einem Aufzählungstyp hat jeder Wert eine spezielle Bedeutung, z.B. beim Status eines TCP-Ports: 1=closed, 2=listen, 3=synSent ...
- Schließlich sind noch zusammengesetzte Datentypen möglich. Ein solcher Typ besteht aus einer Folge („SEQUENCE“) von Werten, die wiederum einfache, abgeleitete oder zusammengesetzte Werte sein können. Zusammengesetzte Datentypen werden in SNMP für Tabellen verwendet.

2.2.3. Protokoll-Operationen

Das SNMP Protokoll besteht aus Operationen, um Objekte abzufragen, zu setzen und Ereignisse zu melden. Objekte werden dabei über ihre Objekt ID und eine Instanz-Nummer ausgewählt. Die

Instanz-Nummer ist bei einfachen Objekten „0“. Bei Tabellen-Objekten ist der Zeilen-Index die Instanz-Nummer. Dieser Index kann eine einfache Zahl sein (z.B. bei ifTable die Interface-Nummer), aber auch jeder andere in SNMP erlaubte Datentyp. So ist bei ipAddrTable der Index eine IP-Adresse (vier Integer-Werte).

2.2.3.1. Abfrage-Operationen

Dies sind Anfragen einer Management Software an eine Netzwerk-Komponente, die Werte bestimmter Objekt Instanzen (SNMP Variablen) zurückzuliefern. Die Objekte können durch ihre Objekt ID direkt angegeben werden. Falls die exakte Objekt ID nicht bekannt ist (Tabellen-Indizes können beliebige Werte annehmen und sind deshalb oft unbekannt), kann auch nach dem Objekt gefragt werden, das einem anderen Objekt direkt folgt (wobei das erstere nicht existieren muß).

GetRequest

Liefert die Werte der angegebenen Objekte zurück; Die Objekte müssen durch ihre ID (mit Instanz) angegeben werden. Kann einer der Werte nicht ermittelt werden, weil z.B. die entsprechende Variable von diesem Gerät nicht bereitgestellt wird, wird ein Fehler gemeldet und für keines der angegebenen Objekte ein Wert zurückgeliefert.

GetNextRequest

Liefert die Werte der den angegebenen Objekten „nächsten“ Objekte. In der RFC wird das nächste Objekt als der „lexikographische Nachfolger“ bezeichnet. Dies entspricht folgendem Algorithmus: Das nächste Objekt ist das Objekt, das

- durch eine Verlängerung der Objekt ID um möglichst wenige Stellen,
- durch eine Erhöhung des Wertes der letzten Stelle der OID oder
- durch Kürzen der OID um möglichst wenige Stellen und Erhöhung des Wertes der dann letzten Stelle

identifiziert wird. Ein Beispiel: Stellt ein Gerät die Objekte

```
iso.org.dod.internet.mgmt.mib-2.interfaces.ifNumber.0
  = 1.3.6.1.2.1.2.1.0        die Anzahl Interfaces des Gerätes
...interfaces.ifTable.ifEntry.ifIndex.65
  = 1.3.6.1.2.1.2.2.1.1.65   die Index-Nummer des 1. Interfaces
...interfaces.ifTable.ifEntry.ifIndex.101
  = 1.3.6.1.2.1.2.2.1.1.101  die Index-Nummer des 2. Interfaces
...interfaces.ifTable.ifEntry.ifIndex.102
  = 1.3.6.1.2.1.2.2.1.1.102  die Index-Nummer des 3. Interfaces
```

bereit, dann liefert eine GetNextRequest-Abfrage nach `...interfaces.ifTable.ifEntry.ifIndex` die Index-Nummer des ersten Interfaces, in diesem Fall 65. Über `...interfaces.ifNumber` kann zwar die Anzahl der Interfaces abgefragt werden, die Nummern der Interfaces selbst bekommt man nur über die entsprechenden GetNextRequest-Abfragen.

Da ifNumber zu keiner Tabelle gehört, hat es eine Instanz-Nummer von 0. Bei den anderen Objekten im Beispiel entspricht die Instanz-Nummer der Interface-Nummer.

GetBulkRequest

Mit GetBulkRequest kann eine größere, evtl. unbekannte Anzahl an Objekten in einer Operation abgefragt werden. Die Abfrage enthält zwei OID Listen, die beide auch leer sein können. Die erste Liste enthält OIDs, die analog zu GetNextRequest bearbeitet werden: Das jeweils „nächste“ Objekt wird zurückgeliefert. Die zweite Liste enthält OIDs, für die mehrere Objekte zurückgegeben werden können. Ein Parameter der Abfrage gibt an, wieviel Objekte maximal zurückgegeben werden sollen. An das Gerät im oberen Beispiel könnte folgende Abfrage gestellt werden:

```
GetBulkRequest
   Liste 1 = (...interfaces.ifNumber)
   max-repetitions = 2
   Liste 2 = (...interfaces.ifTable.ifEntry.ifType,
              ...interfaces.ifTable.ifEntry.ifSpeed)
```

Die Antwort könnte so aussehen:

```
...interfaces.ifNumber = 3
...interfaces.ifTable.ifEntry.ifType.65   = propVirtual(53)
...interfaces.ifTable.ifEntry.ifSpeed.65  = 0
...interfaces.ifTable.ifEntry.ifType.101  = ethernetCsmacd (6)
...interfaces.ifTable.ifEntry.ifSpeed.101 = 10000000
```

Mit einer weiteren Abfrage kann dann die Einstellung des dritten Interfaces abgefragt werden. Mit GetBulkRequest kann so ein großer Teil einer Tabelle mit einer Abfrage übertragen werden. Über die zweite Liste werden die Spalten ausgewählt, mit `max-repetitions` bestimmt man, wieviel Zeilen der Tabelle maximal in einer Abfrage übertragen werden.

GetBulkRequest wurde erst mit SNMP Version 2 eingeführt. Der Grund dafür lag in den immer größer werdenden Routing-Tabellen. Wird eine Tabelle mit GetNextRequest abgefragt, ist pro Tabellen-Zeile eine Abfrage, also ein Netzwerk-Paket nötig. Bei mehreren hundert oder tausend Zeilen dauert die Abfrage so sehr lange.

Mit GetBulkRequest können mehrere Zeilen pro Abfrage übertragen werden. Die Anzahl ist durch die maximale Paket-Größe begrenzt. SNMPv1 verlangt von Protokollimplementierungen, Paketlängen bis zu 484 Bytes zu unterstützen (RFC 1157). Bei SNMPv2 wird empfohlen, Unterstützung für möglichst große Pakete zu implementieren (RFC 1905). Allerdings soll Fragmentierung vermieden werden. Dies bedeutet bei Ethernet eine maximale Paketgröße von knapp 1500 Bytes. Somit können mehrere Dutzend Zeilen in einer Abfrage untergebracht werden.

2.2.3.2. Schreib-Operationen

Mit diesen Operationen können SNMP Variablen verändert und so Netzwerk-Komponenten konfiguriert werden.

SetRequest

Mit dieser Operation wird einem Objekt ein neuer Wert zugewiesen. In der Abfrage müssen die genauen OIDs der Objekte (mit Instanzen) und die Werte, die den Objekten zugewiesen werden sollen, angegeben werden.

2.3. Abfrage-Programme

Für die Kommunikation über SNMP gibt es verschiedene Programme und Bibliotheken. Für einen plattformunabhängigen MIB-Browser interessant ist das komplett in Perl geschriebene Modul **Net::SNMP** [NETSNMP] von David Town. Diese Bibliothek unterstützt die SNMP Versionen 1 und 2c und bietet auch die Möglichkeit, mehrere Objekte in einer Anfrage abzufragen, um die Netzwerk-Belastung niedrig zu halten. Außerdem kann eine SNMP-Tabelle mit einer Bibliotheksfunktion komplett abgefragt werden. Dazu sendet die Bibliothek selbständig so lange **GetNextRequest**- oder **GetBulkRequest**-Anfragen an das Netzwerk-Gerät, bis die ganze Tabelle empfangen wurde.

Andere SNMP-Programme wie UCD-snmp von der Carnegie Mellon University (Pittsburgh) oder Tnm / Scotty von der TU Braunschweig unterstützen teilweise bereits SNMP Version 3 und sind damit den meisten Netzwerk-Geräten einen Schritt voraus.

Auf Grund der Implementierung in reinem Perl und da Netzwerk-Geräte meist nur SNMP Version 1 oder 2c unterstützen, habe ich mich für das Perl-Modul **Net::SNMP** entschieden.

2.4. MIB-Module

Die Definitionen der SNMP Objekte, die MIB-Module, sind in einer Untermenge der Beschreibungssprache ASN.1 geschrieben. Hier als Beispiel ein Auszug aus MIB-II (RFC 1213):

```
RFC1213-MIB DEFINITIONS ::= BEGIN

IMPORTS
        mgmt, NetworkAddress, IpAddress, Counter, Gauge, TimeTicks
            FROM RFC1155-SMI
        IANAifType
            FROM IANAifType-MIB
        OBJECT-TYPE
                FROM RFC-1212;

mgmt          OBJECT IDENTIFIER ::= { iso org(3) dod(6) internet(1) mgmt(2) }
mib-2         OBJECT IDENTIFIER ::= { mgmt 1 }
```

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

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

Dieser Auszug definiert das SNMP Object `iso.org.dod.internet.mgmt.mib-2.system.sysUpTime` oder `1.3.6.1.2.1.1.3` mit dem Typ `TimeTicks`, der aus dem Modul RFC1155-SMI importiert wird. Das Objekt kann nur gelesen werden (`ACCESS read-only`) und sollte von jedem Netzwerkgerät unterstützt werden (`STATUS mandatory`).

`OBJECT-TYPE` ist ein ASN.1-Makro, das in RFC 1155 „Structure of Management Information" (SMI) für SNMPv1 definiert ist. Für SNMPv2 ist eine erweiterte Version in RFC 2578 definiert.

2.4.1. MIB-Compiler

Für diese Syntax gibt es mehrere Parser, darunter auch ein komplett in Perl geschriebenes Perl-Modul **SNMP::MIB::Compiler** [SMC]. Diese Bibliothek stellt die Informationen eines MIB-Moduls als Perl-Daten-Struktur bereit. Die geparsten MIB-Module können von **SNMP::MIB::Compiler** in einer schnell einzulesenden Form gespeichert werden. Dies beschleunigt einen späteren erneuten Zugriff auf das MIB-Modul.

Kapitel 3. Der MIB-Browser

3.1. Aufgabe

Der MIB-Browser soll über ein WWW-Frontend die einzelnen SNMP-Objekte, die in den lokal vorhandenen MIB-Modulen definiert sind, mit ihrer Einordnung in den Baum (über die OID) und ihrer Definition anzeigen.

Dabei soll die Darstellung nicht auf ein MIB-Modul begrenzt sein, sondern die Objekte vieler MIB-Module in einem Baum zusammenfassen. Wünschenswert ist, daß der Browser gleichzeitig mit allen MIB-Modulen umgehen kann, die von den Komponenten eines Netzwerks unterstützt werden. So muß nicht zwischen verschiedenen MIB-Modulen oder Modulsätzen umgeschaltet werden. Der Benutzer muß nicht immer in der Dokumentation nachschauen, welche Module von einer Komponente unterstützt werden. Statt dessen kann er dies durch eine einfache SNMP-Abfrage feststellen (leider wird die Information, welche MIB-Module eine Komponente unterstützt, nicht über SNMP bereitgestellt).

3.2. Einteilung der Anzeige

Das Browser-Fenster wird in drei Bereiche unterteilt: In eine Übersicht der SNMP-Objekte (Objekt-Baum), eine Beschreibung eines ausgewählten Objektes (Objekt-Eigenschaften) und einen Bereich, der das Ergebnis einer SNMP-Abfrage enthält (Abfrageergebnisse).

Der Objektbaum

Die SNMP-Objekte stellen durch ihre OID eine Hierarchie dar. Diese läßt sich anschaulich als Baum darstellen. Da es in HTML keine direkte Unterstützung für einen Baum gibt, muß dies mit einer Tabelle und kleinen Graphiken simuliert werden. Da schon ein MIB-Modul weit über 100 Variablen und Zwischenknoten enthalten kann, muß es möglich sein, beliebige Teile des Baums auszublenden.

Die Objekt-Eigenschaften

Die Beschreibung einer Variablen umfaßt ihren Status, die Zugriffsmöglichkeiten (lesbar, schreibbar), Typ und Syntax (Datentyp) und eine Beschreibung. Dieses Fenster enthält auch die Eingabefelder für Geräteadresse (Hostname oder IP-Adresse) und Instanznummer für eine SNMP-Abfrage. Dabei wird das Objekt abgefragt, das im Fenster beschrieben wird.

Abfrageergebnisse

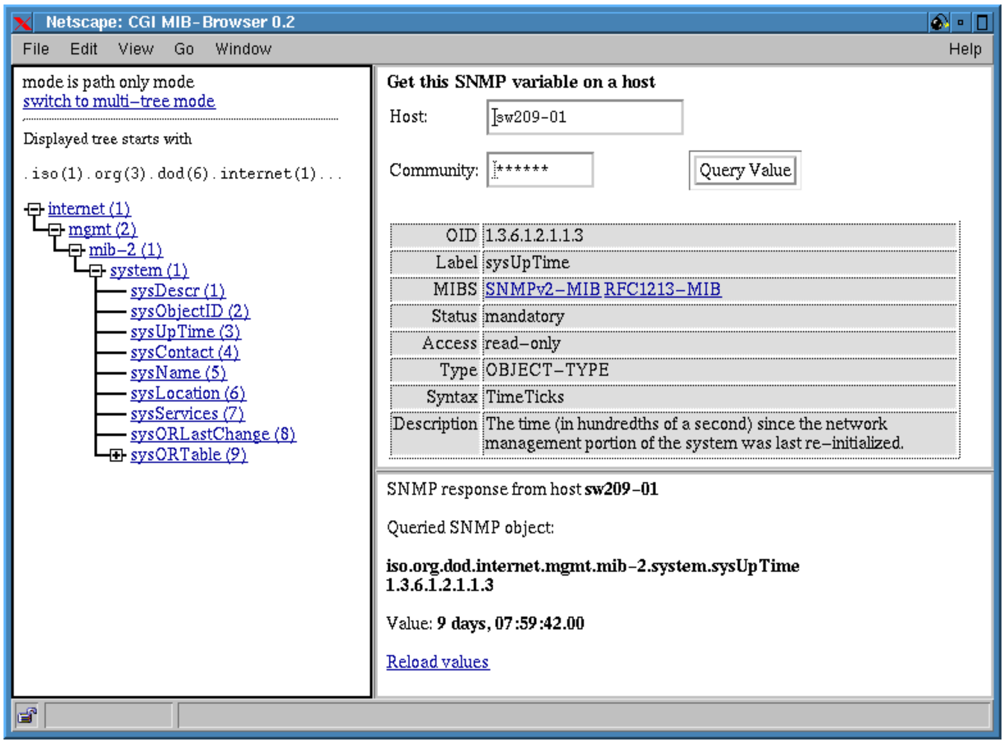
Hier werden die Ergebnisse einer Abfrage angezeigt. Dies kann ein einzelner Wert sein oder eine Liste von Werten. Wird eine Liste angezeigt, sind die Instanznummern der Listenelemente oft genauso wichtig wie die erhaltenen Werte, da sie den Index einer dynamischen Tabelle darstellen.

3.3. Realisierung

Für die Realisierung habe ich die Interpretersprache Perl gewählt. Eine Compilersprache wie C würde zwar eine höhere Performance bieten bzw. geringere Anforderungen an die Serverhardware stellen, Perl bietet dafür auch auf dem Server Plattformunabhängigkeit. Da auch ein MIB-Compiler und ein SNMP-Client in Perl verfügbar sind, konnte ich hier durchgängig Perl verwenden.

3.3.1. Struktur

Abbildung 3-1. Der MIB-Browser im WWW-Browser



Die Einteilung in die drei Teilfenster wird mit einem `Frameset` realisiert. Dies schließt leider Browser, die Frames nicht unterstützen, aus.

Eine kleine HTML-Datei enthält die Frameset-Definition (*mibb.html*). Sie wird vom Browser als erstes geladen. Der Browser teilt das Fenster in die drei angegebenen Bereiche und lädt für jedes Fenster ein neues Dokument vom WWW-Server.

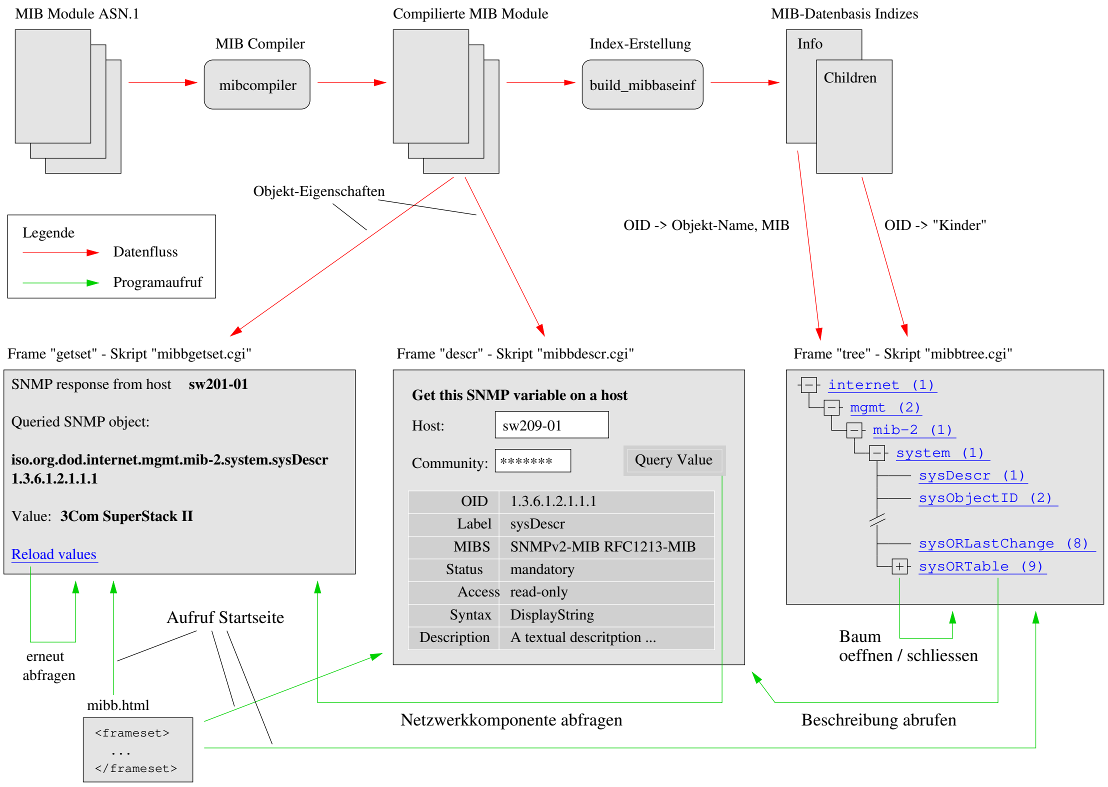
Später bei der Navigation durch den Objektbaum oder der Anzeige der Beschreibung eines Objektes wird jeweils nur ein Bereich des Frameset neu angezeigt und dementsprechend auch nur eine HTTP-Anfrage an den WWW-Server gestellt. Um die Ladezeiten der Perl-Module klein zu halten, habe ich für die drei Bereiche jeweils ein eigenes CGI Skript vorgesehen. Die drei Aufgaben überschneiden sich kaum, so daß

Programmcode nur in geringem Umfang mehrfach vorkommt.

Ein wichtiger Teil des MIB-Browsers sind natürlich die Informationen aus den MIB-Modulen - die „MIB-Datenbasis“. Diese Datenbasis besteht aus den mit SNMP::MIB::Compiler übersetzten MIB-Modulen. Für die häufigsten Anfragen des MIB-Browsers and die Datenbasis gibt es zusätzlich zwei Indizes, um diese Anfragen schnell beantworten zu können.

Die beteiligten Dateien und der Fluß der Daten zwischen Dateien und Programmen:

Abbildung 3-2. Dateien und Datenfluß MIB-Browser



3.3.2. Zusammenspiel der Programme und Datenfluß

Die MIB-Module im ASN.1-Format werden von **mibcompiler** einzeln übersetzt und in compilierter Form gespeichert. Mit **build_mibbaseinf** werden die Indizes über alle Module erstellt.

Beim Öffnen oder Schließen eines Teilbaumes ruft das Skript mibbtree.cgi sich selbst auf, um den veränderten Baum anzuzeigen. Die Ausgabe von mibbtree.cgi erscheint im linken Teilfenster (Frame) „tree“. Die Informationen über den Baum werden aus dem „children“-Index entnommen, die Namen der Objekte und der MIB-Module mit deren Definition aus dem „info“-Index (die Indizes sind im nächsten Kapitel näher beschrieben).

Bei einem Click auf einen Objektnamen im Baum ruft mibbtree.cgi das Skript mibbdescr.cgi auf, das die Eigenschaften des Objektes im rechten oberen Teilfenster „descr“ anzeigt. Die Informationen über das Objekt werden aus dem entsprechenden compilierten MIB-Modul geholt. Der Name der Module wird beim Aufruf des CGI übergeben.

Werden Hostname und Community eingegeben und der Button „query-value“ aktiviert, ruft mibbdescr.cgi das Skript mibbgetset.cgi auf. Dieses Skript schickt eine entsprechende SNMP-Abfrage an die angegebene Netzwerkkomponente und zeigt das Ergebnis im rechten unteren Teilfenster „getset“ an. Der Typ des Objektes (für die formatierte Anzeige) wird wieder aus dem passenden MIB-Modul entnommen, der Name der Module über die URL übergeben.

3.3.3. Die MIB-Datenbasis

Die „MIB-Datenbasis“ besteht aus allen MIB-Modulen, die im MIB-Browser installiert wurden. Da das Parsen der Module im Textformat (ASN.1) sehr lange dauern kann, werden die Module zusätzlich in einer compilierten Form gespeichert. Einzelne MIB-Module können so schnell eingelesen werden, z. B. um die Eigenschaften einer Variablen anzuzeigen.

Mit dem Skript **mibcompiler** wird ein MIB-Modul übersetzt. Das Skript **compile_all_mibs** ruft dieses für jedes installierte Modul auf. Wahlweise können alle alten übersetzten Module gelöscht werden. Ein einzelnes MIB-Modul kann mit dem Skript **insert_mib** in der Datenbasis installiert und übersetzt werden.

Der Objektbaum besteht aus allen Objekten der installierten MIB-Module. Das Einlesen aller MIB-Module für den Baum dauert aber auch bei Verwendung der compilierten Versionen zu lange für ein CGI-Skript. Deshalb werden zwei Indizes über alle MIB-Module erstellt, die auf die Anfragen des MIB-Browsers an die Datenbasis zugeschnitten sind. Diese Indizes sind als DB-Dateien ausgeführt. DB-Dateien enthalten Schlüssel/Wert-Paare; eine Hash-Tabelle erlaubt einen schnellen Zugriff. Module für verschiedene Implementierungen (Berkeley DB, DBM, NDBM, Gnu GDBM, SDBM) werden mit Perl standardmäßig mitgeliefert.

1. Der Index „children“ enthält für jeden „Knoten“ die dazugehörenden Unterknoten.

   Die SNMP Objekte können in zwei Gruppen unterteilt werden:

   - Knoten oder Zwischenobjekte

     Ein SNMP-Objekt, das nicht abgefragt oder gesetzt werden kann und nur dazu dient, die eigentlichen Variablen zu gruppieren und in eine Hierarchie einzuteilen.

   - Blätter oder Variablen

     Die eigentlichen Variablen. Diese Objekte haben keine untergeordneten Objekte, stellen also Blätter im Baum dar.

   Der Index enthält für jeden (Zwischen-) Knoten eine Liste der diesem Knoten untergeordneten Objekte, also wieder Knoten oder Blätter. Zum Beispiel:

   ```
   1.3.6.1.2.1.1 -> 1,2,3,4,5,6,7,8,9
   ```

2. Der Index „info“ enthält für jedes Objekt dessen Namen und die MIB-Module, in denen das Objekt definiert ist.

   Damit können im Baum die Objekte mit Namen aufgeführt werden ohne die MIB-Module einlesen zu müssen. Mit den Informationen aus diesem Index werden die Links erzeugt, bei deren Aktivierung die Eigenschaften eines Objekts im rechten oberen Fenster des MIB-Browsers angezeigt werden. Objektname und Modulname beschreiben ein SNMP-Objekt ebenso eindeutig wie die Objekt-ID (OID). Der Objektname ist nur innerhalb eines Moduls eindeutig. Beispiel:

   ```
   1.3.6.1.2.1.1 -> system; SNMPv2-MIB,RFC1213-MIB
   ```

Diese zwei Indizes werden von dem Skript **build_mibbaseinf** erzeugt, das von den Skripten *compile_all_mibs* und *insert_mib* nach Änderungen an der MIB-Datenbasis aufgerufen wird. Bei einer Testinstallation mit 417 MIB-Modulen (ca. 20 MB im Textformat, 36598 Objekte) dauerte die Erstellung der Indizes 129 Sekunden und benötigte 35 MB Hauptspeicher (auf einem PC K6-2/300). Diese Anzahl an Modulen dürfte in der Praxis selten sinnvoll sein und diente hier nur als Test, ob die Skripte mit einer so großen MIB-Datenbasis zurecht kommen und der Aufbau des Objektbaumes trotzdem schnell vonstatten geht.

Die Indizes müssen neu berechnet werden, wenn zusätzliche Module zur Datenbasis hinzugefügt werden, was selten vorkommen sollte.

3.3.4. Der Objektbaum

Der Objektbaum kann in zwei verschiedenen Modi dargestellt werden:

1. „Full tree mode“: Der Benutzer kann beliebige Teile des Baumes ein- oder ausblenden. Der Baum kann so sehr groß und unübersichtlich werden, erlaubt jedoch eine schnelle Navigation.
2. „Path only mode“: Nur ein Knoten wird mit all seinen Unterknoten gezeigt, ohne evtl. weitere Unterknoten (Enkel). Die Vaterknoten dieses Knotens werden ohne andere Unterknoten gezeigt.

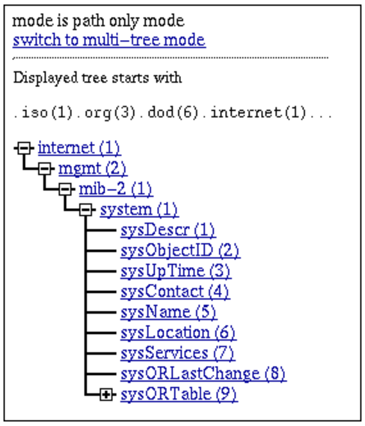
   Wird z.B. der Knoten 1.3.6.1 als Hauptknoten angezeigt, werden dessen Unterknoten 1.3.6.1.* sowie die Knoten 1.3.6, 1.3 und 1 angezeigt, nicht jedoch die Knoten 1.3.6.1.*.* usw.

Die Statusinformation eines Baums (d.h. welche Knoten ein- bzw. ausgeblendet sind sowie der Modus) werden in der URL des CGI-Skriptes übergeben. Im Modus „Path only“ ist das nur der Object Identifier des Objekts, das mit allen Unterknoten angezeigt werden soll. Im Beispiel würde das Skript den Parameter `path=1.3.6.1` bekommen.

Im Modus „Full tree“ sind das die OIDs aller „offenen“ (eingeblendeten) Knoten ohne offene Unterknoten (die Väter eines Knotens „a.b.c“ („a.b“ und „a“) müssen eingeblendet sein, sonst könnte „a.b.c“ nicht angezeigt werden. Deshalb müssen die Väter von „a.b.c“ nicht explizit genannt werden). Der Skriptparameter dafür könnte z. B. lauten: `tree=1.3.6.1.2.1.1:1.3.6.1.2.1.17`

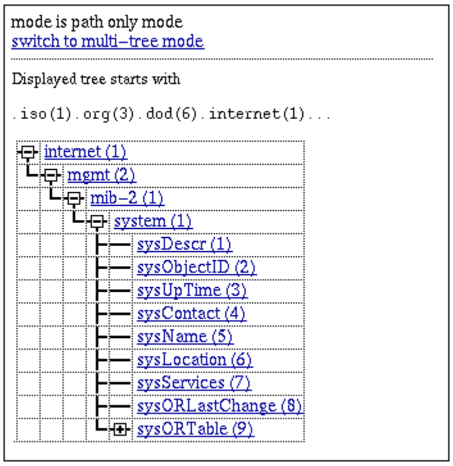
Der Baum wird von einem CGI-Skript erzeugt, das die benötigten Informationen aus den beiden Indizes holt.

Abbildung 3-3. Der Objektbaum



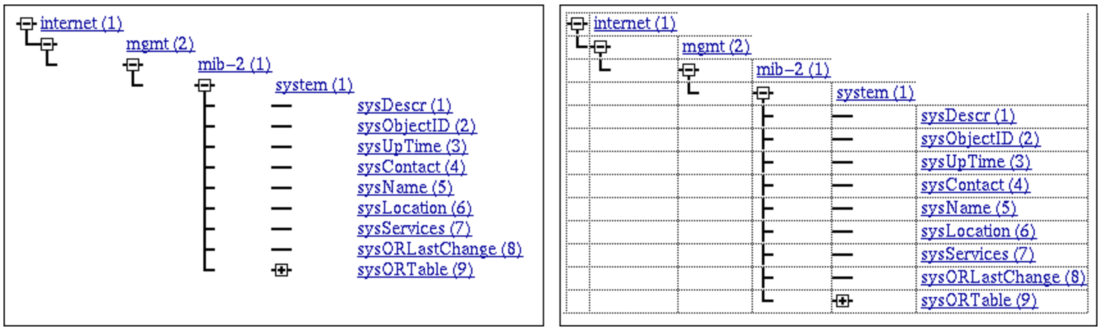
Da HTML keine Bäume unterstützt, wird der Baum mit einer Tabelle simuliert. Die Felder der Tabelle enthalten entweder eine kleine Graphik oder den Namen eines Objekts. Die Objektnamen sind mit einem Link unterlegt, bei dessen Anwahl das Objekt im Eigenschaftenfenster beschrieben wird. Für den Baum kommen die Graphiken  sowie eine leere Graphik zum Einsatz. Ihre Aufteilung auf den Baum zeigt das folgende Bild eines Objektbaumes mit eingeblendeten Tabellenzwischenlinien.

Abbildung 3-4. Der Objektbaum mit Tabellenzwischenlinien



Eine Schwierigkeit besteht darin, daß der Objektname im letzten (rechten) Tabellenfeld breiter ist als die Baumgraphiken. Damit die Graphiken bündig aneinander schließen, muß das rechte Feld über mehrere Tabellenspalten gehen, wie in der letzten Graphik zu sehen ist. Ohne diese Korrektur würde der Baum wie in der folgenden Graphik aussehen:

Abbildung 3-5. Ein HTML-Baum ohne Spaltenkorrektur



Im rechten Bild sieht man, daß alle Spalten so breit wie das breiteste Tabellenfeld in der Spalte gezeichnet werden. Dies führt zu Lücken im Baum, wie im linken Bild deutlich zu sehen ist.

Beim Aufruf bekommt das CGI-Skript eine Liste der OIDs der Knoten, die „offen" angezeigt werden sollen. Eine OID enthält sämtliche Vaterknoten eines Knotens. Die Anzahl der Vaterknoten ist gleich der Anzahl der Spalten für den Baum. Die längste OID bestimmt somit die „Breite" des Baumes und damit der Tabelle. So kann berechnet werden, über wieviel Spalten die letzten Felder bei der Ausgabe gehen sollen.

Leider hängt die Anzeige des Baumes auch vor der Größe der Schrift und damit auch vom verwendeten WWW-Browser ab. Ist die Schrift höher als die Baumgrafiken, entsteht vertikal zwischen den Grafiken ein kleiner Zwischenraum. Größere Grafiken würden hier helfen, würden bei kleinen Schriften jedoch unnötig Platz verbrauchen. Eine genaue Spezifizierung der Schrift über Cascading Stylesheets (CSS) würde auch nicht helfen, da diese Schriften von den verschiedenen WWW-Browsern unterschiedlich dargestellt werden, z. T. abhängig vom Betriebssystem. Man könnte versuchen, je nach WWW-Browser ein anderes Stylesheet zu verwenden. Allerdings gibt es keinen Standard dafür, wie sich WWW-Browser in HTTP-Anfragen melden.

3.3.5. Die Eigenschaftenanzeige

In diesem Bereich werden die Eigenschaften eines SNMP Objektes dargestellt. Dies sind vor allem

- die numerische Objekt-ID (OID, z. B. 1.3.6.1.2.1.1.3),
- der Name des Objekts (z. B. sysUpTime),
- die MIB-Module, in denen das Objekt definiert ist (z. B. SNMPv2-MIB, RFC1213-MIB),
- der Status eines Objekts (vorgeschrieben, optional, veraltet, abgelehnt („deprecated")),
- die Zugriffsmöglichkeiten (schreib- und lesbar, nur lesbar, nur schreibbar, nicht zugreifbar),
- der Typ des Objekts (`OBJECT IDENTIFIER`, `OCTET STRING`, `INTEGER` u.a.)
- die Definition des Index bei Tabellen (welche Objekte mit Typ),
- eine Beschreibung in Klartext für „menschliche" Benutzer.

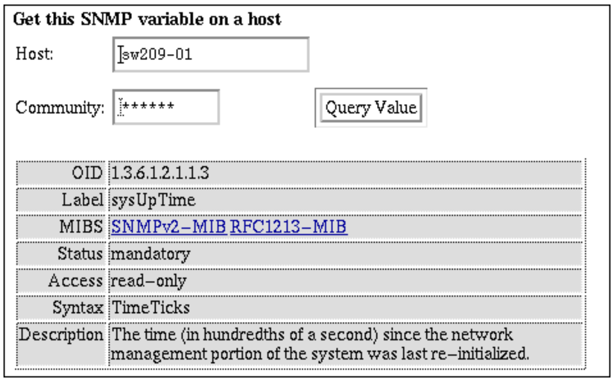
Die Namen der MIB-Module sind mit Links unterlegt. Über diese Links kann ein einzelnes MIB-Modul für die Anzeige bestimmt werden. Dies kann nützlich sein, um das neueste MIB-Modul auszuwählen. Dies kann leider nicht vom Programm festgestellt werden.

Zusätzliche sind hier Eingabefelder für Geräteadresse (Hostname oder IP-Adresse), Community (also das SNMP-Paßwort) und Instanznummer sowie Schaltflächen für eine Abfrage des aktuellen Werts des angezeigten SNMP-Objekts.

Ist das Objekt nicht Teil einer Tabelle, wird kein Eingabefeld für eine Instanznummer angezeigt. Ist ein Objekt nicht abfragbar (z. B. ein Zwischenknoten), werden die Eingabefelder nicht angezeigt.

Die Daten aus den Eingabefeldern (Hostname und Community) werden in einem HTML-„Cookie“ an den WWW-Browser zurückgeschickt. Dieser schickt das Cookie bei späteren Anfragen, um z. B. die Eigenschaften eines anderen Objekts anzuzeigen, wieder an den Web-Server. Das CGI-Skript für die Eigenschaftanzeige (mibbdescr.cgi) verwendet die Informationen aus dem Cookie, um Vorgaben für die Eingabefelder zu machen. Diese enthalten also die Daten der letzten SNMP-Abfrage. So können leicht mehrere Objekte von einem Gerät hintereinander abgefragt werden.

Abbildung 3-6. Die Beschreibung des sysUpTime Objekts



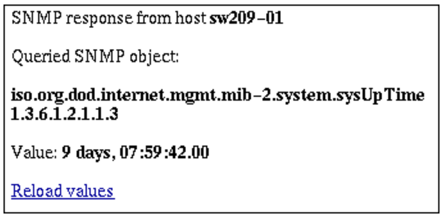
3.3.6. Die Abfrageergebnisse

In diesem Bereich werden die Ergebnisse von SNMP-Abfragen angezeigt. Dies ist entweder eine einzelne Objekt-ID (numerisch und textuell) oder eine Liste von Objekt-IDs (wenn eine Tabelle abgefragt wird).

Das Abfrageskript benutzt für die SNMP-Abfragen das Perl-Modul `Net::SNMP`.

Außerdem setzt das Abfrageskript das oben erwähnte Cookie mit den Werten von Geräteadresse, Community-String und Instanznummer für das Eigenschaften-Skript.

Abbildung 3-7. Das Ergebnis einer Abfrage des sysUpTime Objekts



3.3.7. Sicherheitsaspekte

Die Informationen über die MIB-Module und -Objekte stellen an sich noch keine schützenswerte Information dar. Die Definitionen der Module sind in der Regel öffentlich zugänglich. Die Möglichkeit, SNMP-Geräte mit dem MIB-Browser abzufragen, wirft jedoch zwei Sicherheitsfragen auf:

- SNMP-Abfragen (SNMP Version 1 und 2c) werden durch ein Paßwort, die „Community" autorisiert. Diese Community muß im MIB-Browser eingegeben werden und wird dann über HTTP zum WWW-Server übertragen. Bei dieser Übertragung ist die Community nicht verschlüsselt und könnte abgefangen werden.
- Da SNMP in den meist benutzten Versionen 1 und 2c keine Verschlüsselung bietet, werden Netzwerkgeräte in der Regel so konfiguriert, daß SNMP-Abfragen nur von einigen wenigen Rechnern aus möglich sind. Der MIB-Browser muß auf einer dieser Maschinen installiert sein, um die SNMP-Abfragen ausführen zu können. Dadurch ist aber der SNMP-Zugriff von jedem Rechner über den MIB-Browser möglich.

3.3.7.1. Privatheit der „Community"

Das beschriebene Cookie enthält den Community-String, also das SNMP-Paßwort im Klartext. Die Übertragung in der HTTP-Anfrage stellt ein Sicherheitsproblem dar (außer das Netzwerk zwischen dem WWW-Browser, von dem aus der MIB-Browser bedient wird, und dem Computer, auf dem die Server-Komponente des MIB-Browsers läuft, ist abhörsicher). Dies kann durch den Einsatz von SSL bei der Kommunikation zwischen WWW-Server und -Browser umgangen werden (dabei werden die Daten verschlüsselt übertragen).

Ist das Netzwerk zwischen WWW-Browser und -Server das gleiche wie das, über das die SNMP-Abfragen gemacht werden, ist eine Verschlüsselung des Cookie nicht nötig: SNMP kennt in den meist benutzten Versionen 1 und 2c keine Verschlüsselung. In der SNMP-Abfrage geht der Community-String ebenso im Klartext übers Netzwerk wie im Cookie.

Um Benutzer des MIB-Browsers davon abzuhalten, den MIB-Browser trotz ungesicherter Verbindung für SNMP-Abfragen einzusetzen, könnte der MIB-Browser so verändert werden, daß SNMP-Abfragen nur von gesicherten Verbindungen aus gestartet werden können. Da der MIB-Browser vermutlich oft von einem der späteren Benutzer selbst installiert worden ist, würde das bedeuten, daß sich der Benutzer selbst (oder den Mitbenutzern) mißtraut. Das muß nicht unbedingt schlecht sein.

Ein zweites Problem ist, daß WWW-Browser Cookies auf Platte abspeichern. Benutzt der Netzwerkadministrator den MIB-Browser auf einem fremden Rechner, z. B. bei einem Benutzer, der ein Problem gemeldet hat, so könnte der Benutzer des fremden Rechners den Community-String später aus der Cookie-Datei auslesen. Vom MIB-Browser wird bei der Übertragung des Cookie an den WWW-Browser deshalb keine (!) „expire“-Zeit mitgegeben (ein Parameter, der angibt, wie lange das Cookie gespeichert werden soll). Laut Definition der Cookies sollte das Cookie dann nicht auf Festplatte gespeichert sondern nur im Hauptspeicher festgehalten und beim Beenden des WWW-Browsers gelöscht werden. Beendet der Netzwerkadministrator nach der Benutzung des MIB-Browsers den WWW-Browser, ist der Community-String nicht mehr auf dem Rechner gespeichert.

Falls einem WWW-Browser in dieser Hinsicht mißtraut wird, kann durch eine SNMP-Anfrage über den MIB-Browser mit einem beliebigen, ungültigen Community-String das Cookie auf diesen ungültigen Wert gesetzt werden und so das Problem umgangen werden.

3.3.7.2. Zugriffskontrolle für SNMP-Anfragen

Der eigentliche Zugriffskontrolle find bei SNMP an den Geräten selbst statt. Zum einen wird dort die Community überprüft. Zweitens sollten die Geräte so konfugiriert werden, daß nur Anfragen von einigen wenigen Rechnern beantwortet werden. Der MIB-Browser stellt nun eine Art „Proxy-Funktion“ für SNMP bereit: SNMP-Anfragen werden von einem beliebigen Rechner initiiert und vom MIB-Browser ausgeführt. Die Zugriffsbeschränkung auf wenige Rechner wird umgangen. Hat ein Netzwerkgerät die Standard-Community *public* oder gar eine bekannte „Hintertür-Community“, würde der Zugriff jedem offenstehen. Deshalb ist es nötig, den Zugriff auf den MIB-Browser, zumindest auf die SNMP-Funktion, zu beschränken.

Eine Möglichkeit ist es, den WWW-Zugriff auf den MIB-Browser generell oder nur auf das CGI-Skript *mibbgetset.cgi* zu beschränken. WWW-Server bieten dazu die Möglichkeit, den Zugriff auf WWW-Seiten oder CGI-Skripte auf Anfragen von bestimmten Rechnern (IP-Adressen oder Hostname) oder von autorisierten Benutzern zu begrenzen.

Alternativ könnte im MIB-Browser eine Überprüfung erfolgen. Hier könnte eine feinere Unterscheidung gemacht werden, z. B. daß ein Benutzer nur bestimmte Geräte abfragen darf.

Kapitel 4. Netzwerkstatistiken

4.1. Aufgabe

Ein periodisch (oder ständig im Hintergrund) laufendes Programm soll ausgewählte SNMP-Variablen abfragen und die Werte in Log-Dateien abspeichern. Es soll möglich sein, viele Werte über einen langen Zeitraum (Jahre) zu protokollieren.

Der Zugriff auf die gespeicherten Daten in den Log-Dateien soll so schnell sein, daß es möglich ist, neue Werte mindestens einmal pro Minute einzutragen und gleichzeitig die Daten auszulesen, um die (graphischen) Statistiken zu erzeugen. Der „Datensammler" wird dabei regelmäßig auf alle vorhandenen Log-Dateien zugreifen. Das WWW-Frontend muß - evtl. mehrere - Log-Dateien ganz einlesen. Der Datensammler darf deshalb nur einen kleinen Teil der verfügbaren Ressourcen belegen.

Auf den sparsamen Umgang mit Speicherplatz soll ebenfalls geachtet werden. Bei sehr großen Netzen könnte der verfügbare Plattenplatz sonst den Einsatz beschränken.

Die gesammelten Daten sollen als Graphen über einen WWW-Browser angezeigt werden. Es soll möglich sein, die einzelnen SNMP-Variablen mit einfachen arithmetischen Operationen zusammenzufassen, d.h. daß z.B. die Summe mehrerer Variablen angezeigt wird oder das Verhältnis zweier Variablen zueinander.

4.2. Problem der Datenmenge

Ein Problem ist die anfallende Datenmenge: In einem Netz mit mehreren hundert angeschlossenen Systemen fallen pro Jahr leicht mehrere GigaByte an Meßwerten an. Kritisch ist die Zugriffsgeschwindigkeit: Neue Daten werden in schneller Folge eingefügt (mehrere hundert oder tausend Werte pro Minute), gleichzeitig müssen Statistiken erstellt werden. Für die Statistiken müssen evtl. große Teile der Dateien eingelesen werden, die Zeit für die Erstellung der Statistik soll aber möglichst unter einer Sekunde bleiben. Dies stellt hohe Anforderungen an das Festplattensubsystem. Die Überarbeitung der Daten für die Statistiken stellt bei den erwähnten Datenmengen hohe Anforderungen an den Systemprozessor.

4.2.1. Verringerung durch Kompression

Eine Möglichkeit, die Datenmenge zu verringern, wäre die verlustfreie Kompression („Packen"). Werden nur ältere Dateien gepackt und die Dateien mit den gesammelten Werte für den laufenden Tag nicht, müßte das Kompressionsprogramm nicht bei jedem Einfügen von neuen Daten die Dateien erst entpacken und danach wieder packen; das Einfügen von Daten wäre so nicht verlangsamt.

Für die Erzeugung der Statistiken müßten die Dateien dann erst entpackt werden. Beim Entpacken entsteht temporär wieder die ursprüngliche Datenmenge. Diese Daten würden zwar nicht auf Festplatte gespeichert werden, müßten aber bearbeitet werden, um daraus die Daten für die Statistiken zu erhalten. Für die Erzeugung einer Jahresstatistik mit mehrere SNMP-Variablen müßten mehrere Megabyte eingelesen und bearbeitet werden. Dies würde ein sehr schnelles System erfordern, vor allem da die Dateien ja zuvor entpackt werden müssen.

4.2.2. Verringerung durch Reduktion - reduzierte Auflösung

Eine andere Möglichkeit, die Datenmenge zu verringern ist, die gesammelten Daten bereits beim Einfügen zu bearbeiten und Durchschnittswerte oder Maxima/Minima zu berechnen, analog zu der Berechnung, die z. B. bei der Generierung einer Jahresstatistik gemacht wird:

Eine Jahresstatistik wird bei einer Datenquelle, die einmal pro Minute abgefragt wird, aus ca. 500.000 Meßwerten berechnet. Wird eine graphische Statistik mit 500 Punkten Auflösung in horizontaler Richtung generiert, repräsentiert jeder Bildpunkt 1000 Meßwerte oder 1000 Minuten gleich 16 Stunden und 40 Minuten. Dieser eine Anzeigenwert entspricht damit dem Mittel/Maximum/Minimum von 1000 Meßwerten.

Geht man davon aus, daß weit zurückliegende Meßzeiträume nur als Monats- oder Jahres-Überblick betrachtet werden, könnte man diese Daten in der entsprechenden „Auflösung“ speichern. Für ein Jahr alte Daten könnte dies z. B. bedeuten, nur das Mittel/Maximum etc. eines ganzen Tages zu speichern. Die gespeicherte Datenmenge wird so drastisch reduziert ohne eine Einschränkung für den typischen Anwendungsfall.

Die Reduktion kann abgestuft erfolgen: Für die letzten Tage werden alle Meßwerte gespeichert. Für die letzten Wochen wird auf einen Wert pro 30 Minuten reduziert, für die letzten Monate auf einen Wert pro zwei Stunden, für ältere Daten auf einen Wert pro Tag. Mit den konkreten Zahlen

- Intervall: eine Abfrage pro Minute
- ein Wert pro Minute speichern für eine Woche
- ein Wert pro 30 Minuten für einen Monat
- ein Wert pro zwei Stunden für vier Monate
- ein Wert pro Tag für vier Jahre

werden für einen Meßzeitraum von vier Jahren insgesamt 14.324 Werte gespeichert. Ohne Reduktion wären es 2.102.400 Werte. Werden die Werte im C-Format „double“ (acht Byte pro Wert) gespeichert, hätte die Datei ohne Reduktion eine Größe von ca. 16 MegaByte, mit Reduktion ungefähr 111 kiloByte.

Die Reduktion kann beim Abspeichern der Daten erfolgen. Im Beispiel würde das bedeuten, alle 30 Minuten ein Mittel/Maximum etc. von den Daten zu berechnen, die in den letzten 30 Minuten gesammelt wurden. Dieser berechnete Wert wird - zusätzlich zu dem aktuellen Wert - abgespeichert. Analog erfolgt alle zwei Stunden und einmal täglich eine Zusammenfassung der Daten der vergangenen zwei Stunden bzw. des vergangenen Tages.

Diese Reduktion benötigt Ressourcen während der Eintragung neuer Daten. Die nötigen Festplattenzugriffe für die Berechnungen dürften durch die drastisch verkleinerten Dateigrößen kompensiert werden (die Meßwerte liegen in den Dateien näher beieinander). Die Zeit für die Berechnung der Mittelwerte/Maxima, also z. B. die Mittelwertbildung von 30 Werten, liegt deutlich unter der Zeit für die Netzwerkkommunikation beim Sammeln der Daten über SNMP.

4.3. Realisierung

Wie auch beim MIB-Browser habe ich mich für die Skriptsprache Perl entschieden. Einzig für die Abspeicherung der Daten und die Erzeugung von Graphiken kommt ein in C geschriebenes Programm zum Einsatz.

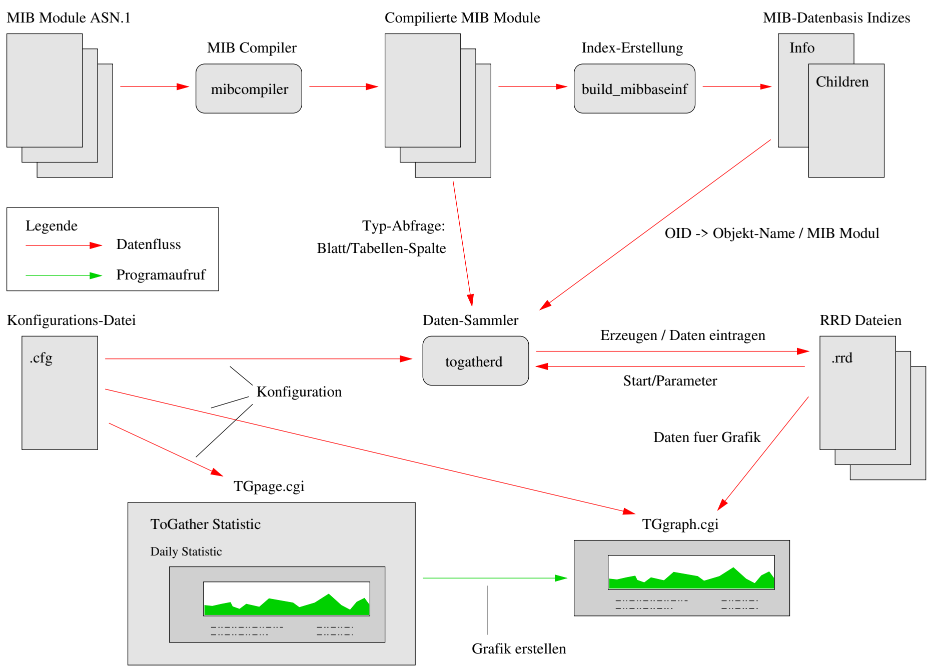
Die Aufgabe unterteilt sich in folgende Bereiche:

- Die Datenspeicherung,
- Ein Programm, das periodisch die Daten sammelt und abspeichert,
- Ein CGI-Skript, das die Daten aufbereitet graphisch darstellt,
- Die Konfiguration der Komponenten.

4.3.1. Struktur: Dateien und Programme

Neben dem Datensammelprogramm und den CGI-Skripten sind noch zwei verschiedene Dateien beteiligt: Konfigurationsdateien und Archivdateien für die gesammelten Daten. Das folgende Schaubild zeigt das Zusammenspiel der Programme und Dateien.

Abbildung 4-1. Dateien und Datenfluß ToGather



Die MIB-Datenbasis ist nötig, um Namen von SNMP-Objekten aufzulösen. Werden nur die numerischen OIDs benutzt, kommt ToGather ohne MIB-Datenbasis aus. Der Typ von SNMP-Variablen wird generell nicht überprüft: Nur Variablen, die den Typ INTEGER oder einen davon abgeleiteten Typ haben, sind für Statistiken geeignet.

Der Daemon **togatherd** liest aus der Konfigurationsdatei die Information, welche Daten von welchen Geräten abgefragt und wo gespeichert werden sollen. Die Daten werden in den RRD-Dateien gespeichert (s. nächstes Kapitel). Die RRD-Dateien werden bei Bedarf vom Daemon erzeugt. Beim Start überprüft der Daemon die Einstellungen der RRD-Dateien mit der Konfiguration und korrigiert die Dateien, falls möglich.

Das CGI-Skript **TGpage.cgi** stellt gemäß der Konfiguration HTML-Seiten bereit. Über eingebettete HTML-IMG-Tags (Links auf Graphiken) wird das CGI-Skript **TGgraph.cgi** aufgerufen. Die Daten für den Graphen entnimmt das Skript den RRD-Dateien.

4.3.2. Die Datenspeicherung: RRDTool

Für die Datenspeicherung verwende ich das Programm **RRDTool** von Tobias Oetiker [RRD]. Dieses Programm löst das Problem der Datenspeicherung hinreichend: Das Dateiformat von RRDTool erlaubt die Speicherung mehrerer Zahlenfolgen in einer Datei. Die Zahlenfolgen werden in den Dateien in Archiven gespeichert. Für jede Zahlenfolge können mehrere Archive definiert werden, in denen die Zahlen in verschiedenen Auflösungen (Anzahl Werte pro Zeiteinheit) gespeichert werden. Die Auflösungen der Archive und der Zeitraum, für den ein Archiv Meßwerte enthält, können getrennt festgelegt werden. Allerdings müssen diese Einstellungen bei der Erzeugung der RRD-Datei feststehen. Allenfalls die Länge eines Archivs (Zeitraum) kann später geändert werden.

RRDTool unterstützt die automatische Reduktion der Daten: Wenn ein neuer Meßwert in eine RRD-Datei gespeichert wird, wird er in alle Archive eingetragen. In Archive mit der „originalen“ Auflösung (die Auflösung entspricht der Meßrate) werden die Daten direkt eingetragen. Bei Archiven mit niedrigerer Auflösung werden die Daten beim Einfügen zusammengefaßt. Dazu stehen mehrere „Consolidation Functions“ zur Verfügung (Durchschnitt, Maximum, Minimum, zuletzt gemessener Wert).

(Anmerkung: Im folgenden werde ich in Beispielen die Durchschnittsfunktion angeben. Grundsätzlich können dann auch immer die anderen „Consolidation Functions“ benutzt werden)

Die Daten werden in den Archiven in zirkulären Listen gespeichert. Diese Listen werden bereits beim Erzeugen einer RRD-Datei so angelegt, daß für alle Daten eines Archivs Platz ist. Wird ein neuer Wert eingetragen, werden nicht alle bisherigen Werte verschoben, um für den neuen Wert am Anfang oder Ende Platz zu schaffen. Statt dessen weist ein Zeiger in die Liste das Element aus, das aktuell das Ende der Liste darstellt. Das neue Element ersetzt an dieser Stelle den ältesten Meßwert, der Zeiger wird um eins erhöht. Dieses Vorgehen ermöglicht es, mit wenigen Schreiboperationen auf Dateien einen neuen Wert in ein Archiv einzufügen. So können viele Werte in kurzer Zeit archiviert werden.

Für die Darstellung als Graph können die Daten dann dem Archiv entnommen werden, das eine Auflösung hat, die am besten zur Auflösung des Graphen paßt. So müssen nicht alle Meßwerte eingelesen werden, was die Erzeugung von Graphen beschleunigt. Die Werte für die Graphen können aus mehreren Dateien stammen und mittels einfacher arithmetische Operationen kombiniert werden.

Bereits bei der Speicherung können auf den Daten einfache Operationen ausgeführt werden. Dazu kennt RRDTool vier verschiedene Typen von „Datenquellen“:

- GAUGE

  Die Werte werden unverändert gespeichert, z. B. für eine Temperaturstatistik
- COUNTER

  Zuerst wird die Differenz zwischen dem neuen und vorhergehenden Wert berechnet. Dann wird durch die Zeit dividiert, die seit dem letzten Update verstrichen ist, also meist die Länge des Meßintervalls. Ist der berechnete Wert negativ, wird von einem Überlauf des Zählers ausgegangen und der Wert entsprechend korrigiert. Dies ist vor allem für Zähler von Netzwerkgeräten geeignet, die meist dieses Verhalten zeigen. Ohne diese Korrektur würde der Graph bei jedem Zähler-Überlauf einen einzelnen, sehr großen Wert enthalten (einen „Spike").
- DERIVE

  Wie bei COUNTER wird die Differenz der zwei letzten Werte gebildet und durch die verstrichene Zeit dividiert. Ein Wert kann im Gegensatz zu COUNTER negativ sein, es wird aber kein Test auf Überlauf gemacht.
- ABSOLUTE

  Die Werte werden, wie bei COUNTER, beim Eintragen durch die verstrichene Zeit dividiert. Allerdings wird nicht die Differenz der zwei letzten Werte berechnet, sondern der Wert direkt verwendet. Dies ist z. B. für Zähler geeignet, die beim Lesen zurückgesetzt werden.

Da RRDTool speziell für langfristige Statistiken entwickelt wurde, bietet es noch weitere Funktionen, die in meinem Programm benutzt werden:

4.3.2.1. Verspätete Updates

Die Daten für RRD-Dateien werden in regelmäßigen Intervallen gesammelt und gespeichert. Jeder Meßwert sollte genau auf einer Intervallgrenze liegen. Dies kann nicht immer sichergestellt werden. RRDTool paßt deshalb Daten, die nicht zum exakten Update-Zeitpunkt gespeichert werden an. In der Regel wird bei Netzwerkstatistiken eine Rate wie z.B. Übertragungsrate oder Fehlerrate gemessen, also mit der Einheit Wert pro Zeiteinheit. RRDTool berechnet den Durchschnittswert (die Rate) zwischen dem zuletzt eingetragenen Wert und dem neuen und errechnet damit einen Schätzwert für die Zeit, zu der das Update hätte stattfinden sollen.

Um verspätete Updates von fehlenden Werten unterscheiden zu können, kann mit einem Parameter angegeben werden, wieviel Zeit maximal zwischen zwei Update-Zeitpunkten liegen darf. Bei RRDTool wird dies „Heartbeat" genannt. Wird diese Heartbeat-Zeit überschritten, wird als Wert „unbekannt" eingetragen.

4.3.2.2. Fehlende Werte

Beim Eintragen von Werten in eine RRD-Datei kann es sein, daß für ein Intervall kein Wert oder der Wert „unbekannt" eingetragen wurde. Dies geschieht entweder automatisch dadurch, daß der „Heartbeat" (s. o.) zwischen Updates überschritten wurde, oder explizit: Anstatt einer Zahl kann „U" als Wert für ein Intervall eingetragen werden.

Bei der Berechnung der Werte für die niedrig auflösenden Archive stellt sich die Frage, wie mit solchen fehlenden Werten umgegangen wird: Ein Mittelwert läßt sich nicht von unbekannten Werten errechnen. Würde man den berechneten Wert auch auf unbekannt setzen, würde ein einziger fehlender Wert pro Tag, daß kein Tageswert mehr berechnet werden kann und die Tagesstatistik nur „unbekannte“ Werte enthält.

Läßt man den unbekannten Wert weg, weicht das Ergebnis evtl. vom wahren Wert ab. Fehlt ein Wert von 30 sollte der Fehler akzeptabel sein. Fehlen 25 von 30 ist der davon berechnete Wert nur noch eine sehr grobe Schätzung der Realität.

RRDTool bietet einen Kompromiß: Über einen Parameter kann bei der Erzeugung einer RRD-Datei angegeben werden, wieviel Daten eines Archivintervalls bekannt sein müssen, um daraus gültige Daten berechnen zu können. Bei RRDTool heißt dieser Wert „x-Files Faktor“.

4.3.3. Der Datensammler „ToGather“

Die Daten werden von einem Programm gesammelt, das ständig im Hintergrund läuft (ein „Daemon“). Das Programm wird von einem Timer periodisch geweckt und sammelt dann die Meßwerte von den verschiedenen Geräten ein. Anstatt eines eigenständigen Daemons könnte auch ein „cronjob“ verwendet werden (ein Programm, das periodisch von **cron** aufgerufen wird), allerdings wäre dann eine Minute die kleinste Zeiteinheit. Intervalle mit Minutenbruchteilen wie z.B. 30 Sekunden oder 2,5 Minuten wären nicht möglich. Ein Daemon erleichtert es außerdem, die verschiedenen Geräte in unterschiedlichen Intervallen abzufragen.

4.3.3.1. Programmablauf

Beim Start liest der Daemon „togatherd“ zunächst die Konfigurationsdatei ein und baut im Speicher eine Liste der Datenquellen auf. Dann wird für jede Datenquelle überprüft, ob die entsprechende RRD-Datei bereits existiert und ob ihre Parameter zu der Konfiguration passen (Intervall, Anzahl und Namen der gespeicherten Variablen). Bei Parametern, die bei RRD-Dateien nachträglich verändert werden können (Variablen-Typ, Maximal- und Minimal-Werte, Heartbeat), wird die RRD-Datei an die Konfiguration angepaßt, da der Benutzer diese geändert haben könnte. Die RRD-Datei enthält auch die Information, wann der letzte Update stattfand. Aus diesem Wert und dem Intervall wird errechnet, wann der nächste Update stattfinden sollte. Falls keine RRD-Datei vorhanden ist, wird gemäß der Konfiguration eine neue erzeugt.

Falls beim Start über eine Kommandozeilenoption angegeben wird eine Übersicht über die eingelesene Konfiguration ausgegeben und das Programm beendet. Andernfalls geht das Programm in den Hintergrund und läuft als Daemon weiter.

Wenn der Daemon das Signal „SIGHUP“ erhält, liest er die Konfiguration neu ein und führt die Überprüfung der RRD-Dateien erneut durch. Dadurch kann die Konfiguration geändert oder erweitert werden, ohne den Daemon anhalten und neu starten zu müssen, wobei ein Update-Zyklus verpaßt werden könnte.

In einer Schleife berechnet der Daemon jetzt für jede Datenquelle die Zeit für den nächsten Update, also wann die Daten das nächste Mal geholt und gespeichert werden sollen. Bis dahin schläft der Daemon, um nicht unnötig Rechenzeit zu verbrauchen.

4.3.3.2. Das Sammeln der Daten

Beim Update versucht der Daemon zunächst, alle Objekte (Variablen) eines Gerätes in einer SNMP-Abfrage zu holen. Gibt es dabei einen Fehler, ist z.B. ein Objekt nicht verfügbar, liefert das Gerät leider keine Teilantwort. Auch wenn also ein Fehler nur bei einem Objekt auftritt, wird für kein Objekt eine Antwort geliefert. Der Daemon fragt darauf die einzelnen Objekte mit einzelnen SNMP-Abfragen ab, um so viele Daten wie möglich zu bekommen.

4.3.3.3. Verspätete Updates

Ist ein Gerät nicht erreichbar, steht dies evtl. erst nach einer gewissen Zeit fest: Ein Gerät schickt keine Nachricht, daß es nicht erreichbar ist. Außerdem kann ein Netzwerkpaket verloren gehen (SNMP basiert auf dem ungesicherten Protokoll UDP), worauf evtl. die Abfrage wiederholt geschickt werden muß. Es kann also je nach eingestellter Wartezeit (timeout) einige Sekunden dauern, bis feststeht, daß für eine Datenquelle kein Wert verfügbar ist. Dies führt bei Datenquellen, die zur gleichen Zeit abgefragt werden sollen, aber zufällig nach der fehlerhaften Datenquelle an der Reihe sind, dazu, daß die Abfrage einige Sekunden später als geplant stattfindet. Eine Verzögerung kann es auch geben, wenn sehr viele Geräte zur gleichen Zeit abgefragt werden müssen.

In diesen Fällen hilft RRDTool, in dem es, wie oben beschrieben, die Daten für den eigentlichen Meßzeitpunkt umrechnet. Durch diese Schätzung entstehen natürlich Fehler. Der Fehler sollte in der Regel klein sein und die Statistik nur unwesentlich beeinflussen: Die Messungen finden in gewissen Intervallen statt, es werden also nicht alle Änderungen der Daten mitgeschrieben, sondern einzelne Zwischenstände zu ausgewählten Zeitpunkten. Ein Meßwert gilt für ein ganzes Intervall. Falls sich die Daten innerhalb eines Intervalls mehrmals ändern können (was bei der Protokollierung von Netzwerkgeräten, also in der Regel Übertragungsraten, der Fall ist), stellen die gemessenen Werte einen Durchschnitt der Werte des Intervalls vor dem Meßzeitpunkt dar. Eine Verschiebung einer Intervallgrenze um einen kleinen Teil der Intervallänge bedeutet eine Änderung des Meßwertes entsprechend der durchschnittlichen Datenrate.

Ändern sich die Werte innerhalb eines Intervalls sehr stark, haben die einzelnen Meßwerte eine geringe Aussagekraft, da sie nur eine Momentaufnahme darstellen und eher zufälligen Charakter haben. Nur aus einer längerfristigen Betrachtung der Werte erhält man aussagekräftige Informationen. Eine kleine Verschiebung des Meßzeitpunktes verringert die Aussagekraft eines einzelnen Meßwertes nicht weiter.

4.3.3.4. Verpaßte Updates

Durch die Verzögerung bei der Netzwerkabfrage kann es bei sehr kurzen Intervallen im Sekundenbereich oder bei einer sehr großen Anzahl von Datenquellen dazu kommen, daß ein Update für eine Datenquelle so spät kommt, daß ein oder mehrere Intervalle über den Soll-Meßzeitpunkt verstrichen sind, ohne daß ein Update stattgefunden hat. In diesem Fall muß für diese Intervalle in der RRD-Datei der Wert „unknown“ eingetragen werden.

4.3.3.5. Kompensation fehlender Werte

Fehlen Daten, z. B. durch nicht-antwortende Geräte oder Verzögerungen beim Datensammeln, kann eigentlich kein genauer Wert für die Langzeitstatistiken für das entsprechende Intervall berechnet werden.

Fehlt z. B. an einem Tag ein Meßwert, kann der genaue Wert für diesen Tag im Jahresgraph nicht bestimmt werden. Damit in Langzeitgraphen keine großen Lücken entstehen, werden einzelne fehlende Werte in den Meßreihen toleriert. Fehlt z. B. nur ein Wert von zehn, wird von den restlichen neun der Durchschnitt (oder das Maximum o. ä.) gebildet. Da die niedriger auflösenden Archive nur Tendenzen oder einen groben Verlauf darstellen, ist der entstehende Fehler oft akzeptabler als ein lückenhafter Graph.

Für diese Kompensation wird eine Funktion von RRDTool verwendet. Als Voreinstellung wird das Fehlen von bis zu 50% der Meßwerte toleriert.

4.3.4. Darstellung der Daten im WWW

Die Darstellung besteht aus zwei Aufgaben: Der Erzeugung von Graphen aus den gespeicherten Daten und der Erzeugung von HTML-Seiten, in denen die Graphen beschrieben und angezeigt werden.

Für die Erzeugung der Graphen ist ein eigenes CGI-Skript zuständig. Dieses Skript bekommt beim Aufruf als Parameter, welche Daten angezeigt werden sollen und für welchen Zeitraum.

Die HTML-Seiten werden von einem weiteren Skript erzeugt. Dieses Skript unterstützt die Erzeugung von Index-Seiten, die Links auf die anderen Seiten enthalten und so eine Gruppierung der Seiten ermöglichen. Die Index-Seiten können auch Index-Graphen enthalten. Dies sind im Prinzip normale Graphen von Meßwerten, die für diesen Zweck ohne Legende und kleiner dargestellt werden können.

4.4. Konfiguration

Die Konfiguration aller beteiligten Programme (Datensammler, CGI-Skripte) erfolgt über eine gemeinsame Konfigurationsdatei.

Generell teilt der Benutzer dem Programm über eine Konfigurationsdatei mit, was es machen soll. Einerseits muß es die Konfigurationssprache erlauben, alle Möglichkeiten des Programms auszunutzen; andererseits sollte sie so einfach wie möglich sein, damit zumindest die grundlegenden Funktionen ohne lange Einarbeitung genutzt werden können.

Die Konfiguration der Statistikprogramme dieser Arbeit teilt sich im wesentlichen in drei Bereiche:

- grundsätzliche Einstellungen wie das Arbeitsverzeichnis,
- die Definition der Datenquellen, also der SNMP-Variablen,
- die Konfiguration der Ausgabe: welche SNMP-Variablen werden mit welcher Beschreibung angezeigt.

Die Konfiguration der Ausgabe kann wieder in drei Teile untergliedert werden:

- Festlegung der angezeigten SNMP-Variablen mit Beschreibung,
- Beschreibung der Graphen: Größe, angezeigter Meßzeitraum, Farben ...
- Definition von Index-Seiten, mit denen die eigentlichen Statistikseiten gruppiert werden.

Diese Bereiche finden sich in der Konfiguration wieder. Hier ein Beispiel:

Beispiel 4-1. Beispielkonfiguration für ToGather

```
### globale Einstellungen
basedir /var/spool/netstat;
# das Unterverzeichnis fuer die Graphen, evtl. relativ zu basedir
graphdir tmp/graph;
# das Unterverzeichnis fuer die rrd Dateien
datadir data;

### eine Datenquelle
datasource "router-1-4" {
        host gw-134;
        community public;
        interval 60; # eine Minute
        variables in out;
        oid in ifInOctets;
        oid out IfOutOctets;
        port 4;
        mibs IF-MIB; # fuer die Aufloesung von ifInOctets + ifOutOctets
}

### eine Statistikseite
target "router-1" {
        title "Router gw-134 Interface 4";
        descr "Datenaufkommen des Subnets 65 an Router gw-134";
        datasource router-1-4;
        calcvars sum=in,out,+;
        graphvars in out sum;
        legend in "Datenverkehr aus dem Netzwerk heraus";
        legend out "Datenverkehr in das Netzwerk hinein";
        legend sum "Summe aus ein- und ausgehendem Datenverkehr";
        # welche Graphen sollen generiert werden:
        graphs daily monthly;
        # Index-Seiten enthalten einen kleineren Graphen
        indexgraph daily.s;
}

### eine Index-Seite
page "routers" {
title "Router Statistiken";
        targets router-1;
}

# die Graphen
group {
        # die Groesse als default fuer beide Graphen
        size 500 150;
        color back "#ffffff"; # Hintergrund weiss
# Farben fuer die Linien: gruen/blau/rot
colors "#00cc00" "#0000ff" "#cc0000";
```

```
        # ein Graph zeigt die Daten eines Tages an
        graph "daily" {
                length 1d; "1 day"
        }
        # ein Graph fuer die letzten (ca.) 30 Tage
        graph "monthly" {
                length 1mon; "1 month"
        }
        graph "daily.s" {
                size 200 100; # kleiner fuer Index
                length 1d;
                withlegends no; # keine Legende wg. Groesse
        }
}
```

4.4.1. Syntax der Konfigurationssprache

Die Konfigurationssyntax lehnt sich an das Format von **bind** und **dhcpd** des ISC (Internet Software Consortium) an.

Die Konfiguration besteht aus einer Liste von `statements`. Ein `statement` ist entweder eine `declaration`, eine `option` oder eine `group`. Eine `declaration` besteht aus einem Schlüsselwort, einem Deklarationsnamen und in geschweiften Klammern eingeschlossenen `options`. Eine `option` besteht aus einem Optionsname, evtl. mehreren Werten, durch Leerzeichen getrennt sowie einem abschließenden Semikolon „`;`“. Eine `group` besteht aus dem Schlüsselwort „`group`“ und in geschweiften Klammern eingeschlossenen `statements`.

Außerdem kann die Konfiguration Kommentare enthalten.

Zum besseren Verständnis hier ein Beispiel:

Beispiel 4-2. Syntax der Konfiguration

```
# Das Doppel-Kreuz "#" kennzeichnet einen Kommentar
option1 "wert1" "wert2";
declaration "name" {
        option2 "wert3";
        size 500 150;
}
group {
        graph "daily" { length 1d; }
        graph "monthly" { length 1mon; }
}
```

Hier die Definition in einer EBNF-ähnlichen Syntax:

```
configSyntax ::= statements
statements ::= statement | statements SP? statement
statement ::= option | declaration | group | comment
options :== option | options SP? option
```

```
option ::= optionname SP values SP? ";"
optioname ::= string
values ::= value | values SP value
value ::= string
declaration :== keyname SP declarationname SP? "{" SP? options SP? "}"
keyname ::= string
declarationname ::= string
group ::= "group" SP? "{" SP? statements SP? "}"
comment ::= "#" string "\n"
string ::= [beliebige Zeichen in "" oder eine Folge von Zeichen ohne SP]
SP ::= whitespace | SP whitespace
whitespace ::= " " | "\t" | "\n"
```

4.4.2. Globale Einstellungen

Hier können allgemeine Einstellungen vorgenommen werden, wie ein Basisverzeichnis für die anderen Verzeichniseinstellungen oder ob ausführliche Meldungen generiert werden sollen (um Fehler in der Konfiguration zu finden). Außerdem können alle Optionen der Deklarationen hier als Voreinstellungen (defaults) angegeben werden.

4.4.3. Definition der Datenquellen

Die Datenquellen werden durch `datasource` Deklarationen konfiguriert. Eine Deklaration steht für eine RRD-Datei. Damit haben die SNMP-Variablen einer Deklaration das gleiche Intervall (dies ist eine Einschränkung von rrdtool). Mit einer Deklaration können beliebig viele SNMP-Variablen von einem Netzwerkgerät abgefragt werden.

Beispiel 4-3. Konfiguration einer Datenquelle mit mehreren SNMP-Variablen

```
datasource "gw-134-fehler" {
        host gw-134;
        community public;
        interval 60; # eine Minute
        variables if1 if2 if3 if4 if5;
        oid if1 ifInErrors.1; # Fehler-Zaehler von Interface 1
        oid if2 ifInErrors.2;
        oid if3 ifInErrors.3;
        oid if4 ifInErrors.4;
        oid if5 ifInErrors.5;
        mibs IF-MIB; # enthaelt die Definition von ifInErrors
}
```

SNMP-Objekte können als numerische OID angegeben werden (z. B. 1.3.6.1.2.1.1.0) oder als Bezeichner. Über die Option `mibs` werden die MIB-Module angegeben, in denen nach den Definitionen der Objekte gesucht werden soll. Ist ein Objektname wie ifInErrors bzgl. der angegebenen MIB-Module eindeutig, kann der Objektname ohne vollständigen Pfad benutzt werden. Im Beispiel genügt so ifInErrors anstatt der kompletten Bezeichnung iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifInErrors.

Die Objekte können mit oder ohne Instanznummer angegeben werden. Bei Tabellenobjekten muß entweder bei `oid` oder über die Option `port` die Instanz angegeben werden, bei anderen Objekten wird automatisch die Instanz „.0“ benutzt. Dazu ist es nötig, daß der Typ des Objekts über das entsprechende MIB-Modul bestimmt werden kann.

Die Angabe der MIB-Module kann weggelassen werden, wenn die Objekte mit vollständiger numerischer OID mit Instanz angegeben sind.

4.4.3.1. Externe Programme als Datenquellen

Zur Abfrage von Datenquellen können auch externe Programme eingesetzt werden. Dadurch ist es möglich, von Diensten Statistiken zu erstellen, die kein SNMP unterstützen. Das externe Programm ist dann dafür zuständig, die gewünschten Daten zu ermitteln und an den Datensammler ToGather zu übergeben. Im folgenden Beispiel wird das Programm **uptime** benutzt, um die „load“, also die Systembelastung zu ermitteln und zu protokollieren. **uptime** hat die Ausgabe

```
  3:04pm  up 5 days,  4:45,  3 users,  load average: 1.21, 0.75, 0.32
```

Wendet man den regulären Ausdruck `(\d+\.\d\d)` (also eine Folge von ein oder mehr Ziffern, einem Punkt und zwei Ziffern) mehrfach auf diese Zeile an, erhält man die drei Zahlen „1.21“, „0.75“ und „0.32“. Die Konfiguration hierfür:

Beispiel 4-4. Konfiguration einer externen Datenquelle

```
datasource "localhost-uptime" {
        type exec;
        interval 60; # eine Minute
        variables load1 load5 load15;
        dstype load1 gauge;
        dstype load5 gauge;
        dstype load15 gauge;
        command "uptime";
        # match: wende regulaeren Ausdruck auf Programm-Ausgabe an
        match regex;
        # die erste Zeile der Ausgabe enthaelt die Information
        matchlines 1;
        # der regulaere Ausdruck im Perl-Format
        regex "(\d+\.\d\d)";
}
```

Es gibt noch zwei weitere Möglichkeiten anzugeben, wie die Daten aus der Ausgabe des Programms extrahiert werden sollen:

1. „fields“ - Die Ausgabe wird mit einem definierbaren `separator` in Felder geteilt. Welche Felder die Daten enthalten, gibt man über deren Nummer an (`field`).

2. „lines“ - Bestimmte Zeilen der Programmausgabe enthalten nur jeweils einen Datenwert. Über die Option `line` wird bestimmt, welche Zeile den Wert für eine datasource-Variable enthält.

Dieser Modus „lines“ ist vor allem für MRTG-Hilfsprogramme gedacht. MRTG verlangt von externen Programmen, daß die ersten zwei Ausgabezeilen nur jeweils eine Zahl mit dem Meßwert enthalten. Mit der Konfiguration oben können diese Programme weiterverwendet werden.

Diese und alle anderen Optionen sind in der Man-Page **TGconfig(5)** ausführlich beschrieben.

4.4.4. Konfiguration der Ausgabe

Die Ausgabe besteht aus Sicht des Benutzers aus Index-Seiten mit Links auf andere Indizes und Statistikseiten, die eine Beschreibung und mehrere Graphen enthalten. Die Graphen möchte der Benutzer in der Regel für bestimmte gewohnte Zeiträume wie Tag, Woche, Monat, Jahr sehen.

Aus Sicht der Programme, die die Ausgabe erzeugen, besteht die Ausgabe aus WWW-Seiten, die die Graphen und Links auf andere Seiten enthalten und den Graphen selbst.

Die Konfiguration ist mehr an die Bedürfnisse des Benutzers angepaßt, also an den, der die Konfigurationsdateien erstellen muß. Dies ergibt eine Aufteilung der Konfiguration der Ausgabe in drei Teile:

- Die Definition der Statistikseiten mit Beschreibung und Auswahl der Datenquellen,
- die Definition der Index-Seiten mit ihren Links und
- die Beschreibung von Graphentypen für die gewünschten Intervalle

Dazu kommt die Definition der Datenquellen für den Datensammler.

Die Programme müssen evtl. Definitionen aus mehreren Bereichen zusammenfassen, um die Definition einer WWW-Seite oder eines Graphen zu erhalten.

4.4.4.1. Konfiguration der Graphen

Ein Graph ist definiert durch:

- die Variablen, die angezeigt werden, evtl. mit Berechnungsvorschrift,
- den Zeitraum, der angezeigt wird,
- im Graph enthaltene Beschreibungen (Legende),
- optische Dinge wie Größe, Farben.

Die Konfiguration der Graphen ist auf die Deklarationen `target` und `graph` verteilt. Mit `graph` wird ein „Graphentyp“ definiert: der Zeitraum, der im Graph dargestellt wird, sowie Größe und Farben. In `target` werden die Parameter definiert, die die Datenquelle beschreiben: Aus welchen `datasource` Definitionen die Daten kommen, die textuelle Beschreibung, die Legenden und welche Graphentypen angezeigt werden sollen.

Dies kann so aussehen:

Beispiel 4-5. Eine komplette Graphen-Konfiguration

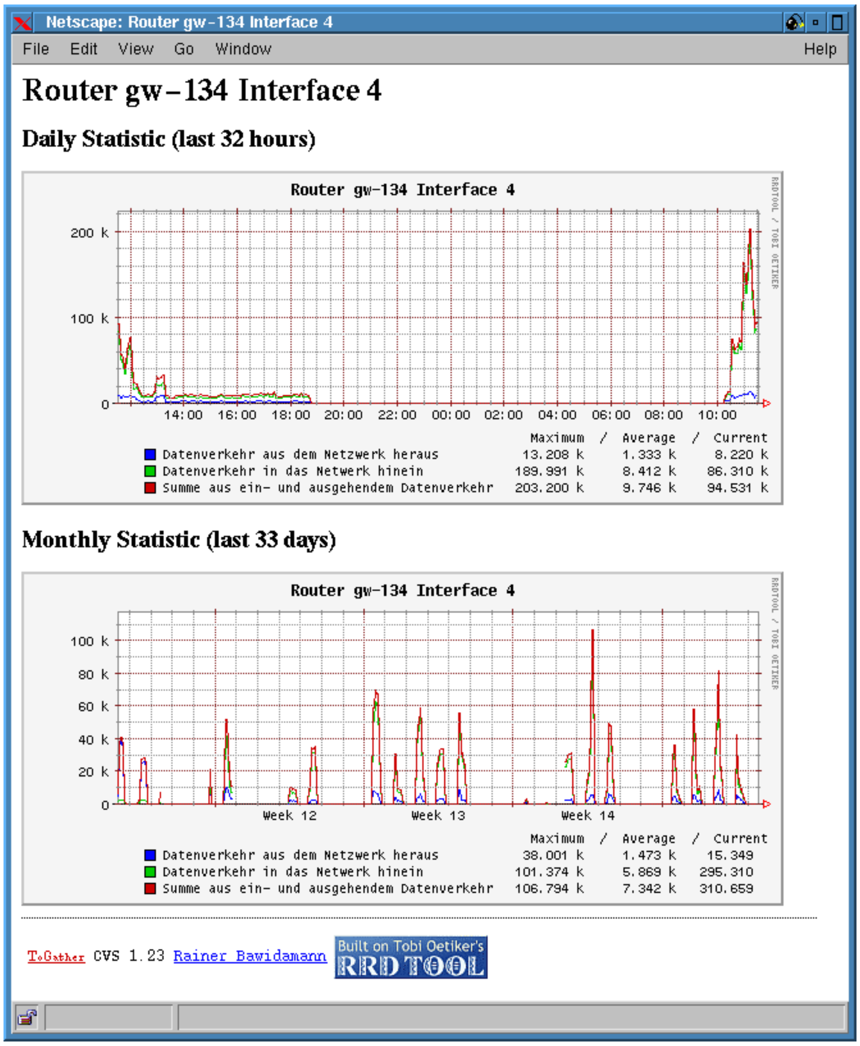
```
group {
        size 500 150;
        color back "#ffffff"; # Hintergrund-Farbe
        colors "#0000ff" "#00cc00" "#cc0000"; # Farben fuer die Graphen

        graph "daily" { length 1d; }
        graph "monthly" { length 1mon; }
}
target "router-1" {
        title "Router gw-134 Interface 4";
        vars in=router-1-4:in:average
             out=router-1-4:out:average;
        calcvars sum=in,out,+;
        graphvars in out sum;
        graphs daily monthly;
        legend in "Datenverkehr aus dem Netzwerk heraus";
        legend out "Datenverkehr in das Netwerk hinein";
        legend sum "Summe aus ein- und ausgehendem Datenverkehr";
}
datasource "router-1-4" { ... }
```

Mit dieser Konfiguration werden zwei Graphen („daily“ und „monthly“) für die Statistikseite „router-1“ definiert. Beide verwenden die Daten aus `datasource` „router-1-4“, wobei die zwei gemessenen SNMP-Variablen unverändert (ohne Umrechnung) angezeigt werden, sowie ein berechneter Wert als Summe aus den beiden SNMP-Variablen. Die drei Datenquellen werden als Linien in den Farben blau, grün respektive rot angezeigt (Option `colors`).

Ein Beispiel für zwei Graphen in einer Statistikseite:

Abbildung 4-2. Eine Statistikseite im WWW-Browser



Die Daten für einen Graphen können auch aus mehreren Datenquellen stammen. Auch können anstatt der gemessenen Werte nur abgeleitete Werte dargestellt werden:

Beispiel 4-6. Ein Graph aus mehreren Datenquellen

```
target "full router-1" {
        title "gesammtes Datenaufkommen des Routers gw-134";
        vars in1=router-1-1:in:average   in2=router-1-2:in:average
             in3=router-1-3:in:average   in4=router-1-4:in:average;
        calcvars sum=in1,in2,in3,in4,+,+,+;
        graphvars sum;
        graphs daily monthly;
        legend sum "Datenaufkommen am Router gw-134";
```

```
}
group {
        host gw-134;
        community public;
        interval 60; # eine Minute
        variables in out;
        oid in ifInOctets;
        oid out IfOutOctets;
        mibs IF-MIB;

        datasource "router-1-1" { port 1; }
        datasource "router-1-2" { port 2; }
        datasource "router-1-3" { port 3; }
        datasource "router-1-4" { port 4; }
}
```

Die Graphen enthalten hier nur eine Linie, die die Summe der "in"-Variablen der Datenquellen router-1-1 ... router-1-4 darstellt.

Durch die Trennung der Definitionen von Datenquellen und Ausgabe können die Datenquellen in mehreren Graphen bzw. Statistiken verwendet werden, z.B. um die Interfaces eines Routers einzeln und als Summe darzustellen. Hiermit wird vermieden, daß eine Datenquelle mehrfach abgefragt wird.

4.4.4.2. Berechnete Daten

Berechnete oder abgeleitete Daten werden mit der Option `calcvars` definiert. Jeder Parameter definiert eine neue Variable als Ergebnis einer Kombination anderer Variablen. Die Variablen für eine solche Rechenformel können protokollierte Daten sein (in `datasource`-Deklarationen definiert) oder vorher über `calcvars` definierte berechnete Variablen. Die Berechnungsvorschrift ist eine Formel in Umgekehrt Polnischer Notation UPN oder Reverse Polish Notation RPN. [RPNorADL] enthält eine grundlegende Einführung in UPN.

Das Prinzip von UPN basiert auf der Idee eines Stapels (Stack). Daten werden auf diesen Stapel geschoben. Operationen (Addition usw.) holen die benötigte Anzahl von Werten vom Stapel und schieben das Ergebnis wieder auf den Stapel. Daten und Operationen werden als Liste angegeben und der Reihe nach abgearbeitet. Der UPN-Ausdruck `4,5,+` berechnet den (Infix-) Ausdruck `4 + 5`: Zuerst werden die Zahlen `4` und `5` auf den Stapel geschoben. Dann wird die Operation `+` ausgeführt. Diese Operation holt zwei Operanden vom Stapel, addiert sie und schiebt das Ergebnis wieder zurück.

Bei der Definition von berechneten Datenquellen gilt zusätzlich: Als Teil des UPN-Ausdruckes können bereits definierte Variablen angegeben werden. Diese werden dann entsprechend auf den Stapel geschoben und können so weiterverarbeitet werden. Nach Berechnung des Ausdrucks muß genau ein Wert auf dem Stapel verbleiben. Dieser Wert wird für den Graphen verwendet.

Der UPN-Ausdruck `sum=in,out,+` aus dem ersten Beispiel berechnet also die Summe aus den Variablen `in` und `out`. Der Ausdruck `sum=in1,in2,in3,in4,in5,+,+,+,+` oben berechnet die Summe der fünf angegebenen Variablen.

Die UPN-Ausdrücke werden unverändert an RRDTOOL weitergegeben. Eine vollständige Beschreibung der Syntax mit der Auflistung aller möglichen Operatoren ist in der Dokumentation zu RRDTOOL, speziell zu RRDGRAPH enthalten.

Weitere Beispiele für Ausdrücke:

- `permin=in,60,*`

  rechnet z.B. eine Übertragungsrate pro Sekunde in eine pro Minute um.

- `percin=in,100,*,in,out,+,/`

  berechnet den relativen Anteil von `in` an `in+out`, also z.B. den Anteil des eingehenden Datenaufkommens am Gesamtdatenaufkommen. Dies könnte, falls `sum` wir oben definiert ist, auch als `percin=in,100,*,sum,/` geschrieben werden.

- `back=percin,95,GT,INF,NEGINF,IF`

  Dieser Ausdruck setzt die Variable `back` auf `INF` bzw. `NEGINF` (dies wird bei der Generierung des Graphen zu einem Punkt am oberen bzw. unteren Rand). Der Operator `GT` ist ein Vergleichsoperator (greater than). Er nimmt zwei Werte vom Stapel und schiebt, je nach Ergebnis des Vergleichs, den Wert 0 oder 1 auf den Stapel. Der Operator `IF` wählt einen von zwei Werten aus: Falls der Wert an der drittobersten Stelle im Stapel (hier: das Ergebnis von `GT`) ungleich null ist, ist anschließend `INF` auf dem Stapel, sonst `NEGINF`.

[RPNTUT] enthält ein Tutorial zu RPN Ausdrücken.

4.4.4.3. Konfiguration der WWW-Seiten

Das Beispiel im letzten Abschnitt definiert neben zwei Graphen auch die Statistikseite „router-1“. Diese Seite zeigt das Datenaufkommen eines Router-Interfaces. Eine solche Statistikseite stellt eine bestimmte Kombination von Daten (gemessener und berechneter) in mehreren Graphen für verschiedene Zeiträume dar. Auch dies wird mit den Deklarationen `target` und `graph` konfiguriert.

Ein Beispiel für eine Statistikseite zeigt Abbildung 4-2.

Die zweite Sorte von WWW-Seiten, die generiert werden können, sind Index-Seiten. Diese werden mit der Deklaration `page` definiert. Im Beispiel wird eine Index-Seite mit Links auf sechs Statistikseiten definiert:

Beispiel 4-7. Konfiguration einer Index-Seite

```
page "router-1" {
        title "verfuegbare Statistiken des Routers gw-134";
        columns 2;
        targets router-1-1 router-1-2 router-1-3 router-1-4 "full router-1";
}
datasource ...
target ...
```

Eine Index-Seite kann wiederum Links auf andere Index-Seiten enthalten. So kann eine Hierarchie (genauer: ein Netz) von Index- und Statistikseiten erzeugt werden.

4.5. Default-Seiten

Um die Konfiguration zu vereinfachen und dem Einsteiger einen schnellen Start zu ermöglichen werden in bestimmten Fällen Standard-WWW-Seiten oder Graphiken erstellt.

Beim ersten Aufruf des CGI **TGpage.cgi** bekommt dieses keinen Parameter wie sonst beim Aufruf über einen Link aus einer von **TGpage.cgi** generierten Seite. Zuerst versucht das Skript, über die globale Option `startpage` herauszufinden, was es anzeigen soll. Ist diese Option nicht gesetzt, wird eine Liste aller vorhandenen *page*-Deklarationen angezeigt. Wird nur eine gefunden, wird dies direkt angezeigt.

Sind keine *page*-Deklarationen vorhanden, wird eine Liste der *target*-Deklarationen gezeigt. Ist nur eine davon vorhanden, wird die Seite direkt angezeigt.

Werden keine *target*-Deklarationen gefunden, wird die Liste der vorhandenen *datasource*-Deklarationen gezeigt. Analog bei nur einer Deklaration die Seite direkt.

Eine *datasource*-Deklaration kann direkt als *target* angesprochen werden, z. B. in der `targets`-Option einer *page*-Deklaration. Die Seiten enthalten die vier Graphen „daily“, „weekly“, „monthly“ und „yearly“. Jeder Graph enthält alle Variablen der *datasource*.

Auch für *datasource*-Deklarationen gelten Defaults:

```
community = public;
interval = 300;
variables ds0 ds1;
oid ds0 1.3.6.1.2.1.2.2.1.16; # ifInOctets
oid ds1 1.3.6.1.2.1.2.2.1.10; # ifOutOctets
dstype * counter;
archive average,max a:2d 30min:2w 2h:2mon 1d:2y;
```

So genügt die Konfiguration

```
datasource "my-router" { host gw-134; port 5; }
```

um eine Statistik der Übertragungsrate des Interfaces 5 des Routers „my-router“ zu erstellen und eine WWW-Seite mit Tages-, Wochen-, Monats- und Jahresgraphen zu bekommen.

Kapitel 5. Ergebnisse

5.1. Der MIB-Browser

5.1.1. Funktionsumfang

Der MIB-Browser erfüllt im wesentlichen alle gestellten Erwartungen. Die Objekte praktisch beliebig vieler MIB-Module können mit dem Objekt-Baum durchstöbert werden. Die Abfragemöglichkeit von Netzwerkgeräten erlaubt eine schnelle Überprüfung des Status eines Gerätes. Auch hilft es dabei, die Funktion eines Objektes oder eines Netzwerkgerätes zu verstehen. So gibt die Liste der Interface-Typen eines Switches Aufschluß darüber, ob das Gerät virtuelle Netzwerke („VLAN“) unterstützt. Dies entspricht allerdings nicht ganz der Intention, die zur Entwicklung des MIB-Browsers führte.

Bei Tests des MIB-Browsers kam der Wunsch auf, neben einzelnen SNMP-Objekten und Spalten einer Tabelle auch eine ganze Tabelle abfragen zu können. In manchen Fällen hilft die Abfrage eines Objekts oder einer Spalte nicht weiter, die Informationen erschließen sich nur Zeilenweise oder als ganze Tabelle. Außerdem kann ein Tabellen-Index (eine *instance*) aus mehreren Tabellen-Feldern bestehen, also beliebig komplex werden und ist dann schwer von Hand einzugeben. Diese Komplexität der Indizes macht die Implementierung im MIB-Browser aber komplex und umfangreich und konnte in der Zeit, die für die Diplomarbeit zur Verfügung steht, nicht realisiert werden. Dies soll nach Ende der Diplomarbeit nachgeholt werden. Damit würde der MIB-Browser eine Funktion bieten, die bei vergleichbaren Produkten unbekannt ist.

Ein Problem konnte mit dem MIB-Browser nicht befriedigend gelöst werden: Die Suche nach dem richtigen MIB-Modul und dem richtigen Objekt. Zwar kann der ganze Objekt-Baum durchsucht werden; durch die Einteilung der Objekte in eine Hierarchie, die vor allem durch die Module bestimmt ist, fehlt eine Anordnung der Objekte nach ihrer Funktion. Vor allem die Verknüpfung von Standard- mit herstellerspezifischen Modulen wäre hilfreich: Viele Hersteller stellen über Objekte in eigenen MIB-Modulen Ergänzungen zu Standard-Modulen bereit. Diese finden sich dann aber im Unterbaum für die herstellereigenen Module, also an einer ganz anderen Stelle im Baum. Durch die große Menge von Objekten ist es nicht praktikabel, alle Objekte der entsprechenden MIB-Module anzeigen zu lassen. Praktisch wäre eine systematische Sortierung nach Funktion anstatt nach Modul. Alternativ könnte eine Voll-Text-Suche in den Objekt-Beschreibungen helfen.

5.1.2. Performance

Durch die verschiedenen Indizes ist die Geschwindigkeit akzeptabel. Sie könnte durch den Einsatz von mod_perl [MODP] gesteigert werden (die Perl-Skripte würden dann nur einmal geladen und übersetzt anstatt bei jedem Aufruf). Bei großen Objekt-Bäumen kann die Navigation etwas „zäh“ werden, da manche WWW-Browser relativ viel Zeit für die Darstellung großer Tabellen (wie der Objekt-Baum) benötigen.

Die Erzeugung der verschiedenen Indizes kann bis zu wenigen Minuten dauern. Dies ist aber nur bei der Einspielung neuer MIB-Module nötig.

5.2. Statistiken

5.2.1. Funktionsumfang

Die Statistik-Software erfüllt die Anforderungen hinsichtlich Flexibilität und Effizienz beim Sammeln und Darstellen der Daten: Daten werden nicht mehrfach geholt. Sie können fast beliebig für die Darstellung kombiniert werden, die Erfassung und Speicherung der Daten bleibt davon unberührt. Die Statistiken können über einen langen Zeitraum geführt werden, ohne übermäßig Plattenplatz zu beanspruchen. Die Aussagekraft der Statistiken wird dabei im typischen Anwendungsfall nicht verringert.

Graphen und WWW-Seiten werden „on-demand" generiert, also nur wenn sie abgerufen werden und nicht nach jeder Messung neu wie beim MRTG. Durch die Zwischenspeicherung der Graphen wird eine unnötige Neuberechnung vermieden, falls der Graph kurz hintereinander mehrfach abgefragt wird.

Die Konfiguration erlaubt es, die WWW-Seiten in weiten Grenzen zu gestalten, bietet aber auch Voreinstellungen, so daß eine Konfiguration sehr schnell erstellt werden kann.

5.2.2. Zuverlässigkeit

Der Datensammler **ToGather** war während meiner Arbeit an der Diplomarbeit, vornehmlich der Ausarbeitung, mehrere Wochen im Einsatz und erfüllte seine Aufgabe in dieser Zeit fehler- und absturzfrei.

5.2.3. Ressourcen-Verbrauch

Ein togatherd-Daemon benötigt etwa 6-8 MB Hauptspeicher, wovon etwa 3 MB ausgelagert werden können. Die Prozessorlast ist je nach Anzahl Datenquellen und der Länge der Intervalle unterschiedlich, eine nennenswerte Belastung der verwendeten Test-Rechner konnte nicht festgestellt werden.

Die WWW-Seiten werden zügig erzeugt. Enthält eine Seite allerdings viele Graphen, die neu berechnet werden müssen, dauert die Anzeige der Seite teilweise mehrere Sekunden. Dies ist natürlich stark von der Performance des Servers abhängig. Da ein WWW-Browser in der Regel mehrere Anfragen gleichzeitig absetzt (meist bis zu fünf), wird der Server durch die gleichzeitige Berechnung mehrerer Graphen stark belastet. Würden die Graphen nacheinander berechnet werden können, würde die Ladezeit aufgeteilt. Regelmäßig in kurzen Intervallen würde der Benutzer einen weiteren der Graphen sehen, die Seite würde sich allmählich aufbauen. Dies wird trotz gleich langer Gesamt-Ladezeit normalerweise angenehmer empfunden als eine lange Wartezeit ohne Ergebnis, nach der die ganze Seite auf einmal dargestellt wird.

Kapitel 6. Ausblick

In der Einführung wurden als mögliche Erweiterungen Kurzzeitstatistiken und Meldung von Ausfällen genannt. Dieses Kapitel denkt über diese und weitere Möglichkeiten und Entwicklungen nach.

6.1. Kurzzeitstatistiken

Die Idee ist, eine im MIB-Browser ausgewählte SNMP-Variable für eine kurze Zeit zu protokollieren und sukzessive graphisch darzustellen. Der MIB-Browser könnte dabei eine einfache Konfigurationsdatei für ToGather erstellen und den Daemon starten. Um schnell Ergebnisse sehen zu können, müßte das Meßintervall sehr kurz sein, vielleicht nur eine Sekunde. Soll jeder Meßwert sofort in der Graphik erscheinen, müßte RRDTool die Graphik jedesmal komplett neu berechnen. Effizienter wäre es, die Meßwerte in der Graphik nachzutragen. Dies würde eine Erweiterung oder einen Ersatz für RRDTool erfordern. Da die Statistik nur für kurze Zeit erstellt wird, vielleicht einige Minuten bis wenige Stunden, könnte auf eine Reduktion der Daten verzichtet werden.

Die sukzessive Darstellung der Daten im WWW-Browser ist über entsprechende Expire- und Refresh-Werte für die HTML-Seiten oder über Server-Push möglich. Dabei wird jedoch die Graphik jedesmal komplett übertragen. Evtl. könnten eine sukzessive Übertragung nur der Änderungen der Graphik durch die Verwendung eines „Multi-GIF“ realisiert werden. Dies müßte aber noch untersucht werden.

6.2. Meldung von Ausfällen

Beim Sammeln der Daten fällt die Information ab, welche Geräte aktuell erreichbar sind und welche nicht. Dies könnte dem Netzwerkadministrator z. B. per Mail gemeldet werden.

Aber: Bei der hierarchischen Struktur eines Netzwerkes bedeutet ein (Total-) Ausfall eines Gerätes, daß alle aus Sicht des Statistikprogramms „danach“ angeordneten Geräte nicht mehr erreichbar sind. Zwischen *nicht-erreichbar* und *ausgefallen* besteht jedoch insofern kein Unterschied, als daß die Reaktion auf SNMP-Abfragen die gleiche ist: ein Ausbleiben jeglicher Antwort oder (Fehler-) Meldung. Die Mail an den Administrator würde also nicht ein einzelnes sondern viele Dutzend Geräte nennen und wäre nicht mehr als der Anruf eines Benutzers des betroffenen Teilnetzes: „Das Netz geht nicht mehr!“. Der Administrator müßte einen Netzplan konsultieren oder alle betroffenen Statistiken anschauen.

Die Hierarchie eines Netzwerks, die *Topologie*, ändert sich nicht so häufig. Wäre die Topologie dem Statistikdatensammler bekannt, könnte es dadurch zwischen ausgefallenen und nicht-erreichbaren Geräten unterscheiden. Die Meldung an den Administrator würde nur ein Gerät benennen, das dann konkret vom Administrator untersucht werden kann.

Durch Auslesen von Routing- und Switch-Tabellen wäre es möglich, die Topologie eines Netzes automatisch zu erkennen. Dies allein würde aber wohl eine eigene Diplomarbeit ausfüllen.

Einen Teilausfall oder eine Fehlfunktion automatisch zu erkennen wäre wesentlich komplexer: Die Bedingung für den Zustand „Totalausfall“ ist schnell definiert und überprüft. Bei einer Fehlfunktion „kann alles passieren“, im Prinzip muß die Bedingung für „korrekte Funktion“ definiert und getestet werden. Eine SNMP-Abfrage würde diesem Anspruch nicht genügen: Ein funktionierender SNMP-Agent muß nicht bedeuten, daß der Router noch korrekt routet, oder daß ein Name-Server noch auf Anfragen antwortet. Ein automatischer Test würde also eine Überprüfung aller wichtigen Funktionen erfordern. Dies wäre aufwendig zu implementieren und zu konfigurieren und würde weit größere Ressourcen benötigen, als die Erstellung von Statistiken.

Ein Kompromiß wäre der Versuch, Anomalien in den Statistiken automatisch zu erkennen. Dazu könnte man Schwellwerte für die gesammelten Werte setzen, bei deren Überschreitung Alarm ausgelöst wird. Aufwendiger wäre eine Approximation des Verlaufs der Daten mit einer Funktion, um durch die Ableitung der Funktion starke Schwankungen festzustellen.

6.3. Benutzerprofile, History

Wenn man Netzwerkgeräte abfragt, werden oft die gleichen Objekte ausgewählt. Eine automatische oder von Hand zu erstellende Liste von häufig benutzten Objekten würde es ersparen, immer wieder dieselben Objekte im Baum anzusteuern. Denkbar ist auch die Möglichkeit, eine ausgewählte Gruppe von Objekten gleichzeitig abzufragen.

6.4. „Device-Browser“

Nicht jedes Netzwerkgerät unterstützt jedes MIB Modul. Ein „Device-Browser“ könnte eine Liste aller Objekte eines Gerätes vom Gerät selbst abfragen und als Objektbaum anzeigen. Der Benutzer kann dann die Objekte durchsuchen und so auch das Gerät konfigurieren.

6.5. Veröffentlichung der Software

Als ich meine „Versuchssoftware“ *14all* geschrieben hatte, entschloß ich mich dazu, sie zu veröffentlichen, also den Leuten zugänglich zu machen, die mir mit der Entwicklung und Veröffentlichung von Programmen und Bibliotheken (Perl-Module, RRDTool) diese Diplomarbeit erst ermöglicht haben [14all]. Ich bekam viel Feedback und dadurch einen Eindruck davon, wie SNMP-Statistiksoftware eingesetzt wird und welche Funktionen hauptsächlich genutzt werden. Neben den zu erwartenden Fehlermeldungen („Bug-Reports“) wurden mir auch viele Erweiterungsvorschläge und fertige Erweiterungen („Patches“) geschickt. So hat die eröffentlichung der Programmqualität und dem Funktionsumfang gedient

Deshalb werde ich auch den MIB-Browser und die ToGather-Programme nach Ende der Diplomarbeit veröffentlichen. Ich möchte damit zum einen der Internetgemeinschaft („Internet-Community“) etwas zurückgeben. So wie am Rechenzentrum der Uni Ulm *MRTG* bei der Verwaltung des Intranet hilft, so könnten meine Programme anderen Administratoren die Arbeit erleichtern. Dann erhoffe ich mir durch den Feedback, die Programme weiter verbessern zu können.

Die WWW-Adresse wird voraussichtlich `http://www.uni-ulm.de/urz/mibbtogather` sein.

Kapitel 7. Rückblick

Dieses Kapitel bietet einen Übersicht über den Verlauf der Arbeit und die Entwicklung der Programme.

7.1. Einarbeitung in das Thema

Die erste Aufgabe war die Einarbeitung in SNMP. Eine gute Einführung bot ein Artikel aus *SimpleWeb* [SNMPMIB], einer Website mit einer Sammlung von Informationen und Links zu SNMP. Einen Überblick über weitere Netzwerkmanagementprotokolle bot die Diplomarbeit von Walter Zimmer [WZ99].

Danach habe ich mich mit MRTG beschäftigt. Dabei lernte ich RRDTool kennen, das vom gleichen Autor stammt. Zum besseren Verständnis von MRTG und RRDTool habe ich eine Erweiterung von MRTG entwickelt, so daß anstatt der MRTG-eigenen Routine zur Speicherung der Daten und Generierung der Graphiken RRDTool verwendet wird. Dabei wurde mir klar, daß es nur durch eine Neu-Strukturierung und damit Neu-Implementierung von MRTG möglich wäre, die Grenzen von MRTG aufzuheben.

Für den MIB-Browser war es nötig, MIB Module zu parsen und die Information schnell zugreifbar zu speichern. Neben in C geschriebenen Programmen fand ich bald das Perl-Modul SNMP::MIB::Compiler. Da es ganz in Perl geschrieben ist, war es genau richtig, um einen plattformunabhängigen MIB-Browser zu entwickeln. Ich erweiterte das Modul noch ein wenig für meine Bedürfnisse (der Name des MIB Moduls, in dem ein Objekt definiert ist, wurde nicht mit dem Objekt gespeichert).

Für das Statistikprogramm brauchte ich zusätzlich die Unterstützung von SNMP. Hier gibt es mehrere Perl-Module. Eines davon, Net::SNMP, kommt ohne C-Bibliotheken aus, enthält also nur Perl-Code. Dieses Modul konnte ich unverändert verwenden.

7.2. Implementierung

7.2.1. Der HTML-Baum

Das erste Zwischenziel bei der Implementierung war, einen Baum in HTML nachzubilden. Erste Versuche mit einer HTML-Tabelle brachten schnell ein optisch ansprechendes Ergebnis. Die Berechnung der Breite der letzten Tabellenfelder kostete einige Versuche. Anfangs kodierte ich den Status des Baumes (welche Knoten sichtbar waren) über einen Bitstring (ein Bit je Knoten: 1 = sichtbar). Der Bitstring wurde jedoch schnell sehr lang und so wandelte ich ihn erst in hexadezimale Darstellung um und komprimierte ihn mit einer Lauflängen-Kodierung („run length encoding“, RLE). Ein Nachteil dabei ist, daß die Gesamtzahl an Objekten in der MIB-Datenbasis bekannt sein muß, da die Zuordnung der Objekte zu den Bits davon abhängt.

Schließlich erwies es sich als besser, beim Aufruf des CGI-Skriptes direkt die Liste der „offenen“ Knoten zu übergeben. Dadurch ist es nicht mehr nötig, die Gesamtanzahl an Objekten in der MIB-Datenbasis zu kennen. Durch die Entfernung der redundanten Knoten ist diese Liste in der Regel auch sehr kurz.

7.2.2. Performance Tests mit vielen MIB-Modulen

Als das Skript für den Objektbaum des MIB-Browsers in einer ersten Version fertig war, besorgte ich mir eine größere Anzahl von MIB-Modulen, um so die Performance zu testen. Ich konnte eine MIB-Basis von 20 Megabyte im Quellformat ansammeln. Diese MIB-Basis umfaßt insgesamt 36598 Objekte, von denen 7519 Zwischenknoten sind. Der Test damit zeigte, daß eine Optimierung der Datenbasis für die Performance nötig war.

Ich probierte mehrere Formen von Indizes über die Datenbasis, einmal sogar mit Einsatz einer SQL-Datenbank. Bei diesen Tests mußte ich feststellen, daß SNMP-Objektnamen nur innerhalb eines Moduls eindeutig sind. Dies war mir aus der verwendeten Literatur so nicht klar geworden. Da ich bisher Objektnamen als Schlüssel verwendet hatte, mußte ich auf die Objekt-Identifier als Schlüssel umstellen. Schließlich optimierte ich die Indizes auf den MIB-Browser Fall und kam zu der jetzt verwendeten Form.

7.2.3. Der Datensammler

Der erste Schritt war die Entwicklung einer Konfigurationssprache und eines passenden Parsers. Über Syntax und Semantik der Konfiguration ist der Funktionsumfang festgelegt. Hier konnte ich die Erfahrungen, die ich mit MRTG, 14all und anderer Software gemacht hatte, einbringen. Die Entwicklung des Datensammlers selbst verlief dann zügig und brachte schnell Ergebnisse.

Bei der Arbeit an den CGI-Skripten konnte ich Teile von 14all, meinem Versuch mit MRTG und RRDTool, wiederverwenden. Datensammler und CGI-Skripte testete ich zuerst auf meinem Arbeitsrechner. Die Ergebnisse waren bald zufriedenstellend und ich habe dann das Programm auch auf dem Netzwerkprotokoll-Rechner des Rechenzentrums installiert, was ja ein Ziel der Diplomarbeit war.

7.3. Dokumentation

Als der MIB-Browser grundsätzlich funktionierte, begann ich mit der Erstellung der Dokumentation. Gleichzeitig fing ich mit der Arbeit am Datensammler und den CGI-Skripten an.

Ich entschloß mich, für die Dokumentation den SGML-Standard „DocBook“ zu verwenden. Da die entsprechende Software auch auf den Mehrbenutzerrechnern des Rechenzentrums vorhanden ist, hielt ich mir so die Möglichkeit offen, die Arbeit in einem Studenten-Rechnerpool fertigstellen zu können, für den Fall, daß ich durch einen Zwischenfall mit meinem Rechner nicht weiterarbeiten könnte.

7.4. „Widrige Umstände“

Das Schicksal vergaß mich nicht: Etwa zur Halbzeit der sechs Monate Bearbeitungszeit wurde die Universität von einem Brand heimgesucht, ich verlor Arbeitsplatz und Rechner. Zum Glück konnte ich meine Daten retten. Da das ganze Rechenzentrum vom Brand beeinträchtigt war (alle Räume waren unbenutzbar), mußte ich mir selbst einen Ersatzarbeitsplatz und -Rechner suchen und wurde in der Informatik fündig. Ich mußte jedoch die verwendete Software neu installieren. Nach und nach bekam ich meinen Rechner zurück, einen neuen Arbeitsplatz im Rechenzentrum (zu meinem Erstaunen, immer noch herrscht hier Platznot) und schließlich (nach zwei Monaten) auch meine Notizen zurück. Die zwei Umzüge von Raum zu Raum und der Software von Rechner zu Rechner kosteten einige Zeit, weshalb eine Verlängerung der Zeit für die Diplomarbeit nötig war.

Durch den Brand war auch der Netzwerkoperator-Rechner *noc* des Rechenzentrums lange Zeit außer Betrieb, der mir als Testrechner dienen sollte. Aus demselben Grund war auch kein Ersatzrechner verfügbar. Erst in den letzten vier Wochen konnte ich meine Software auf *noc* testen.

Literaturverzeichnis

Bücher

[SNMPWS] *Snmp, Snmpv2, Snmpv3, and Rmon 1 and 2*, William Stallings, Januar 1999, Addison-Wesley Pub Co., ISBN: 0201485346.

Diplomarbeiten

[WZ99] *Netzwerkmanagement*, Walter Zimmer, Februar 1999, Universität Ulm, Fakultät für Informatik.

Online-Dokumente

[SNMPMIB] *The SimpleWeb*, http://wwwsnmp.cs.utwente.nl/, Dave Perkins, „Understanding SNMP MIBs“, September 1993.

[RFC] *Request For Comment (RFC)*, http://www.rfc-editor.org/rfc.html.

[SimpleTimes] *The Simple Times*, ISSN: 1060-6080, http://www.simple-times.org, Allan Leinwand, „Accomplishing Performance Management with SNMP“.

[NETSNMP] David Town, *Net::SNMP Perl Modul*, http://www.cpan.org/modules/by-authors/id/D/DT/DTOWN/.

[SMC] Fabien Tassin, *SNMP::MIB::Compiler Perl Modul*, http://www.cpan.org/modules/by-authors/id/FTASSIN/.

[SBROWSER] Jürgen Schönwälder, *WWW SNMP MIB Browser*, http://www.ibr.cs.tu-bs.de/cgi-bin/sbrowser.cgi.

[TNM] Jürgen Schönwälder, *Tnm - Tcl Extensions for Network Management Applications*, http://wwwhome.cs.utwente.nl/~schoenw/scotty/.

[RPNorADL] James Redin, 1997, *RPN or DAL*, http://www.dotpoint.com/xnumber/rpn_or_adl.htm.

[MRTG] *Multi Router Traffic Grapher MRTG*, Tobias Oetiker, http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/mrtg.html.

[MRTG-L] *MRTG Mailing-Liste*, http://www.ee.ethz.ch/~slist/mrtg/.

[RRD] *Round Robin Data Tool RRDTOOL*, Tobias Oetiker, http://ee-staff.ethz.ch/~oetiker/webtools/rrdtool/.

[RPNTUT] *Reading RRDTool RPN Expressions*, Steve Rader, enthalten in RRDTool [RRD].

[14all] *14all - Combine MRTG and RRDTOOL*, Rainer Bawidamann, http://www.uni-ulm.de/~rbawidam/mrtg-rrg/.