
Netzwerkmonitoring mit Cacti

Alexander Kern, Oliver Skawronek

Praktikumsarbeit für Netzwerk- und Systemmanagement

vorgelegt im: März 2012
Matrikelnummer: 56948 (Alexander Kern)
57601 (Oliver Skawronek)
Studiengang: Master Informatik
Kurs: INM11

Hochschule für Technik, Wirtschaft und Kultur Leipzig
HTWK-Leipzig

Inhaltsverzeichnis

1	Einleitung	1
2	Überblick	1
2.1	Einsatzmöglichkeiten	2
2.2	Messdatenerfassung	3
2.3	Messdatenspeicherung	3
2.4	Messdatendarstellung	4
2.5	Templates	4
2.6	Benutzerverwaltung	5
2.7	Plug-Ins	5
2.8	System-Architektur	6
3	Messdatenerfassung	6
3.1	Abfrage Script-basierter Messwerte	7
3.2	Abfrage SNMP-basierter Messwerte	10
3.2.1	SNMP-Grundlagen	10
3.2.2	Abfrage eines Managed-Objects	14
3.2.3	Abfrage von Tabellen	15
4	Messdatenspeicherung	16
4.1	RRDTool-Grundlagen	17
4.1.1	Data-Source	18
4.1.2	Round-Robin-Archive	19
4.2	Zusammenhang zwischen RRDTool und Cacti	21
5	Messdatendarstellung	22
5.1	Graph-Items	23
5.2	Berechnungen mittels CDEF-Anweisungen	24
6	Anwendungsbeispiel: Privater Homeserver	26
6.1	Einleitung	26
6.2	Konfiguration des Homeservers	27
6.3	Anwendungsbeispiele von Cacti	27
6.3.1	Ventrilo: Aktuelle Teilnehmerzahl	28
6.3.2	Verbrauch eines Verzeichnisses samt Unterverzeichnissen	28
6.3.3	Apache2: Vollständige Downloads	28
6.3.4	OpenSSH: Zählen missglückter Anmeldeversuche	28
6.3.5	Firewall: Blockierte Eingangsversuche	28
6.3.6	Downloads-Messung unter Berücksichtigung alter Log-dateien	28

6.3.7 Jabber: Onlinestatus eines Accounts	28
A Beispiel Anhang	28
Literatur	32

1 Einleitung

Der Betrieb eines Netzwerkes erfordert u. a. die Überwachung von Netzwerkkomponenten. Cacti ist eine open-source Softwarelösung für diese Aufgabe. In Cacti werden in regelmäßigen Abständen Werte von Netzwerkkomponenten aufgezeichnet und dem Benutzer als Graphen präsentiert. Cacti kann u. a. Kapazitätsengpässe grafisch hervorheben und erlaubt damit ein schnelles Reagieren in kritischen Situationen.

Unter einer Weboberfläche kombiniert Cacti verschiedene Technologien zum Erfassen, Speichern und Darstellen von Messwerten. Detailkenntnisse in den Technologien werden vom Benutzer nicht vorausgesetzt. Cacti bietet hierzu Voreinstellungen für Standardaufgaben, bspw. für das Überwachen der Netzlatenz mittels Ping. Erfahrene Benutzer können jedoch individuelle Einstellungen treffen.

Ziel dieser Arbeit ist es dem Leser einen Überblick über Cacti zu geben, die Funktionalität zu erläutern und anhand eines privaten Homeservers Einsatzmöglichkeiten aufzuzeigen.

2 Überblick

Cacti ist eine Webanwendung zum Erfassen, Speichern und Darstellen von Messwerten. Hauptsächlich wird Cacti im Bereich des Netzwerkmonitorings eingesetzt. Über eine Weboberfläche erfolgt die Administration. Hier können die zu überwachenden lokalen oder entfernten Netzwerk-Geräte, wie Server, Router und Switches, verwaltet werden. In regelmäßigen Abständen ruft Cacti die Messdaten dieser Geräte ab und speichert sie langfristig in Archiven. Neben der Abfrage über Scripts können die Daten auch aus Anfragen über das *Simple Network Management Protocol* (SNMP) stammen. Aus ausgewählten Messdaten (gesendete/empfangene Bytes, erfolgreiche/fehlgeschlagene Anmeldungen etc.) werden Graphen erstellt und auf der Weboberfläche präsentiert.

Für den Einsatz in größeren Netzumgebungen erleichtert Cacti die Verwaltung gleichartiger Netzkomponenten durch sogenannte „Templates“. Die Idee dabei ist, für eine Geräteklasse (ggf. eines bestimmten Herstellers) in einem Template festzulegen, welche Messdaten erfasst und dargestellt werden sollen. Wird ein neues Gerät dieser Klasse in die Netzumgebung installiert, werden zu diesem Gerät alle benötigten Graphen automatisch erstellt.

Cacti unterstützt den Mehrbenutzerbetrieb. Durch Vergabe von Nutzerrech-

ten lässt sich festlegen, welche Einstellungen ein Benutzer treffen kann und welche Graphen ihm angezeigt werden. Die Anmeldung erfolgt über die Weboberfläche und kann sowohl lokal als auch entfernt über einen Browser geschehen.

2.1 Einsatzmöglichkeiten

Cacti bietet eine Vielzahl von Einsatzmöglichkeiten, von denen im Folgendem einige herausgegriffen werden:

Technische Einsatzmöglichkeiten

- Auslesen von Log-Dateien. Im Kapitel 6 werden dazu Beispiele vorgestellt.
- Benachrichtigung via E-Mail, wenn kritische Werte über- bzw. unterschritten werden. Hierfür ist ein zusätzliches Plug-In einzubinden.
- Einige Datenbank-Managementsysteme bieten die Abfrage von Metriken, wie Verbindungen, Sperren und Cache Misses, via SNMP an. Diese Metriken lassen sich dann mit Cacti überwachen.

Einsatzmöglichkeiten im Netzwerk- und Systemmanagement

- Überwachen der Einbruchsversuche durch Anbindung an ein Firewall-system
- Darstellung des Netzwerkverkehrs sowie der Speicher- und Prozessorauslastung
- Überwachen der Signalstärke von WLAN-Stationen
- Abfrage der Metriken von Proxyservern, Routern, VoIP-Telefonanlagen etc.
- Statistische Auswertung des E-Mailverkehrs (Speicherverbrauch, Spam, usw.)
- Überwachen der Auslastung einzelner Prozessorkerne. Auf Grundlage dieser Messungen können Kaufentscheidungen getroffen werden, bspw. ob sich die Anschaffung weiterer Mehrkern-Prozessoren unter der eingesetzten Software lohnt.
- Gegenüberstellung des Verbrauchs unterschiedlicher Ressourcenkategorien, wie Anwendungsdaten und Media.

2.2 Messdatenerfassung

Die Messdatenerfassung erfolgt regelmäßig in festen Abständen (standardmäßig im Fünf-Minuten-Takt) über den sogenannten „Poller“. Dazu werden Scripts und Anfragen über SNMP ausgeführt und die ermittelten Messwerte gespeichert. Unter Linux wird bspw. der Poller als Cronjob¹ aufgerufen. Cacti bietet zum Erfassen von Messdaten mehrere Möglichkeiten an:

- Data-Input-Methods: Abfrage skalarer Werte, bspw. der Umgebungstemperatur eines Switches. Die Werte können von der Ausgabe eines Scripts oder einer SNMP-Abfrage stammen.
- Data-Querys: Abfrage Index-basierter Werte. Die Daten stammen aus einer Tabelle. Jeder Tabellenzeile ist ein eindeutiger Indexwert zugewiesen. Beispielsweise könnte ein Host in einer Tabelle die gesendeten und empfangenen Daten aller installierten Netzwerkadapter anbieten, wobei jedem Netzadapter ein Laufindex zugeordnet ist. Abgefragt werden die Daten entweder per Script oder SNMP.

2.3 Messdatenspeicherung

Die langfristige Speicherung von Messdaten erfordert einen ständig wachsenden Bedarf an Speicherplatz. Misst man bspw. alle fünf Minuten die Ping-Zeit, so müssen 12 Messpunkte pro Stunde, $12 * 24 = 288$ Messpunkte pro Tag, $288 * 365 = 105120$ Messpunkte pro Jahr usw. gespeichert werden. Während man aktuelle Messdaten über kurze Zeitabstände betrachten möchte, genügt typischerweise nur ein Überblick über ältere Werte. Beispiel:

- Alle fünf Minuten wird ein Messwert erfasst.
- 12 Messwerte werden zu einem konsolidierten „Stunden-Messwert“,
- 24 „Stunden-Messwerte“ zu einem konsolidierten „Tages-Messwert“, usw. zusammengefasst.

Aktuelle Werte werden somit in hoher Auflösung und ältere Werte in geringer Auflösung gespeichert. Mögliche Konsolidierungsfunktionen sind: *Mittelwert*, *Maximum*, *Minimum* usw.

¹Ein Cronjob ist unter Linux eine sich wiederholende Aufgabe. Er besteht aus der Angabe des aufzurufenden Befehls und dem zeitlichen Abstand zwischen zwei Aufrufen.

In Cacti werden die erfassten Messwerte nach diesem Prinzip abgespeichert. Dazu basiert Cacti auf dem RRDTool (RRD für Round Robin Database). Es garantiert über längere Zeiträume einen konstanten Platzbedarf zur Speicherung der Messwerte.

Data-Input-Methods bzw. Data-Querys dienen der Abfrage von Messwerten. Die sogenannten „Data-Sources“ legen hingegen fest, wie die Messdaten in den Round-Robin-Datenbanken (RRDs) gespeichert werden. Diese RRDs speichern mehrere Messgrößen in Archiven unterschiedlicher Auflösung ab. Eine zu speichernde Messgröße, bspw. die empfangenen Byte eines Netzwerkadapters, entspricht in Cacti einem „Data-Source-Item“. Mehrere Data-Source-Items sind einer Data-Source zugeordnet. Die Data-Source legt hauptsächlich fest, welche Archive (Stunden-Archiv, Tages-Archiv, Wochen-Archiv usw.) für die Messgrößen angelegt werden und bestimmt damit das Schema der zugeordneten RRD.

2.4 Messdatendarstellung

Das RRDTool dient Cacti neben der Speicherung der Messwerte auch zum Erstellen der Messdatendarstellung in Form von Graphen. Mehrere Messgrößen lassen sich in einem Graph darstellen. Neben der Möglichkeit, Titel, Legende, Achsenbeschriftung, Farbe etc. festzulegen, ist hier die Fähigkeit von RRDTool hervorzuheben, mathematische Funktionen auf die Eingabegrößen anzuwenden. Damit können bspw. einfache Größenumrechnungen, wie von Bit/s nach Byte/s, umgesetzt werden. Die Bestandteile eines Graphen, wie eine darzustellende Messgröße oder die Einträge in der Legende, werden als „Graph-Items“ bezeichnet.

2.5 Templates

Insbesondere der angebotene Template-Mechanismus erleichtert den Einstieg Netzressourcen mit Cacti zu überwachen, da Detailkenntnisse über Data-Sources u. Ä. nicht vorausgesetzt werden.

Es werden drei Arten von Templates unterschieden:

1. Data-Templates: Sie kommen zum Einsatz, wenn mehrere Data-Sources gleiche Charakteristika teilen. Auf diese Weise werden automatisch mit jeder Data-Source (die auf einem Data-Template basiert) die zugehörigen Data-Source Items angelegt.

2. Graph-Templates: Mittels Graph-Templates lassen sich einheitlich aussehende Graphen erstellen. Titel, Legende, Achsenbeschriftung etc. werden in einem Graph-Template einmal festgelegt, und alle daraus abgeleiteten Graphen erben diese Eigenschaften.
3. Host-Templates: Einem Host-Template sind eine Menge von Data-Templates und Graph-Templates zugeordnet. Angelegt werden Host-Templates für gleichartige Geräteklassen. Wird ein Gerät einer solchen Geräteklasse der Netzumgebung hinzugefügt, können automatisch alle Data-Sources zur Messdatenerfassung und Graphs zur Messdatendarstellung erstellt werden.

Durch den Template-Import und -Export ist es Möglich, Templates zu verteilen. Beispielsweise werden unter [5] Templates für „Cisco-Router“ zum Download angeboten.

2.6 Benutzerverwaltung

Mit jeder Standardinstallation von Cacti werden die Nutzer „admin“ und „guest“ angelegt. Weitere Benutzer lassen sich hinzufügen. Jedem Benutzer ist ein Benutzername und Kennwort zugeordnet sowie eine Menge von Benutzerrechten. Die Benutzerrechte legen fest, welche Einstellungen in Cacti von einem Benutzer getroffen werden können und welche Graphen für ihn sichtbar sind.

2.7 Plug-Ins

Es existiert eine Vielzahl an Plug-Ins für Cacti. Über sie können zusätzliche Informationen angezeigt, Funktionen ergänzt und Daten und Graphen manipuliert werden. Beispielsweise dient das Plug-In „Thold“ [2] der Überwachung von Messgrößen. Überschreitet ein Messwert einen bestimmten Schwellwert, etwa eine zu hohe Festplattenauslastung, kann Thold automatisch den Administrator per E-Mail darüber benachrichtigen.

Cacti kommuniziert über sogenannte „Hooks“ mit den Plug-Ins. Hooks sind Call-Back-Funktionen, für die sich jedes Plug-In registrieren muss. Cacti ruft diese Funktionen auf und übergibt Informationen in Form von Argumenten. Soll ein Plug-In bspw. auf Statusänderungen eines Geräts reagieren, muss es sich für den Hook „Update-Host-Status“ registrieren und eine gleichnamige Funktion implementieren.

2.8 System-Architektur

Zur Abfrage nutzt Cacti einen Cronjob-basierten Poller um Daten aus verschiedenen Quellen abzufragen. Die abgefragten Daten werden in RRD-Dateien gespeichert. Alle Einstellungen zur Systemkonfiguration werden in MySQL-Tabellen gehalten. Sowohl die Konfiguration als auch die Darstellung der Graphen auf der Weboberfläche wurde hauptsächlich in der Scriptsprache PHP implementiert. Cacti ist u. a. für Windows, Linux und Solaris verfügbar [6].

3 Messdatenerfassung

Zu den grundsätzlichen Aufgaben in Cacti zählen die Messdatenerfassung, -Speicherung und -Darstellung. In diesem Kapitel werden die unterschiedlichen Möglichkeiten der Messdatenerfassung mit Cacti vorgestellt. Die Messdaten können von verschiedenen, ggf. entfernten Quellen stammen. Im Bereich des Netzwerkmonitorings ist vor allem die Möglichkeit zu nennen, Messwerte mittels SNMP abzufragen. Eine Vielzahl an Netzwerk-Geräten, wie Switches, Router und Drucker, unterstützen dieses Protokoll. Neben SNMP bietet Cacti an, Daten über Scripte abzufragen. Das bedeutet, der Ausgabe-Stream von Linux-Befehlen bzw. Linux-Scripts kann ausgelesen werden. Daneben besteht die Möglichkeit, direkt PHP-Scripts auszuführen.

Die Messdatenerfassung ist konzeptmäßig von der -Speicherung und -Darstellung getrennt. Für die weitere Verarbeitung werden die Messwerte in ein einheitliches Format kodiert, d. h. es muss nicht beachtet werden, aus welcher Quelle die Daten stammen. Die einzige Voraussetzung ist, dass die Werte ganzzahligen oder reellwertigen Datentypen zugeordnet sind.

Abgerufen werden die Messwerte in einem festen Intervall durch den Poller. Wie der Name bereits erahnen lässt, wird die aktive Abfragemethode „Polling“ umgesetzt, d. h. die Geräte senden auf Anfrage ihre Werte an Cacti. Standardmäßig führt der Poller im Fünf-Minuten-Takt die SNMP-Anfragen bzw. Scripts aus, und speichert anschließend die Werte in Archiven ab. Sind die Messgrößen einem bestimmten Gerät zugeordnet, prüft der Poller zusätzlich, ob das Gerät noch aktiv ist, bspw. durch eine Ping-Anfrage.

Es existieren zwei Implementierungen des Pollers:

1. cmd.php: In PHP implementierter Standard-Poller.
2. Spine: In C implementierter Poller.

URBAN empfiehlt in [6] `cmd.php` für kleine bis mittelgroße Netzumgebungen. Für große Netzumgebung sollte hingegen Spine eingesetzt werden. Das ist u. a. damit zu begründen, dass Spine multitaskingfähig ist.

Je nach Art der abzufragenden Daten wird zwischen Data-Input-Methods und Data-Querys unterschieden:

- **Data-Input-Methods:** Abfrage eines oder mehrere skalarer, nicht indexbasierter Werte.
Beispiele: Temperatur, Speicherverbrauch, Anzahl eingeloggter Benutzer, Anzahl empfangener/gesendeter Pakete, Anzahl erfolgreicher/abgebrochener Datenbank-Transaktionen
- **Data-Querys:** Abfrage indexbasierter Werte. Die Werte werden aus einer Tabelle ausgelesen, wobei jeder Tabellenzeile ein eindeutiger Indexwert zugewiesen ist.
Beispiele: Liste mit Netzwerkkarten (Verbindungsgeschwindigkeit, empfangene/gesendete Pakete usw.), Liste mit Festplatten (Gesamtkapazität, Verfügbarer Speicher usw.)

Für beide Arten werden im Folgenden Möglichkeiten aufgezeigt, Werte über Scripts oder SNMP abzufragen.

3.1 Abfrage Script-basierter Messwerte

Das Auslesen Script-basierter Daten erlaubt in Verbindung mit den Kommandozeilen-Befehlen „awk“, „grep“, „wc“ u. a. eine Reihe einfacher Abfragen auf Textdateien, wie Server-Logs. Kombiniert werden diese Befehle häufig über Pipes, d. h. der Ausgabestrom des vorhergehenden Befehls dient als Eingabe für den nächsten Befehl. In diesem Abschnitt soll anhand von Beispielen erklärt werden, in welchem Format letztlich die Ausgabe vorliegen muss, damit Cacti sie einlesen kann, und wie die Scripts gesteuert werden.

Beispiel Einlesen eines einzelnen Wertes:

Der Einfachheit halber soll mittels Echo-Befehl der konstante Wert 10,45 ausgegeben werden:

```
echo '10.45'
```

Zum Einlesen des Wertes muss eine neue Data-Input-Method mit dem Input-Type „Script/Command“ erstellt werden (siehe Abbildung 1). Im Feld Input-String ist der auszuführende Befehl bzw. der Pfad zum auszuführenden Script anzugeben.

Abbildung 1: Auslesen eines Wertes mittels Data-Input-Method

Abbildung 2: Der Data-Input-Method wird ein Output-Field hinzugefügt

Einer Data-Input-Method sind eine Menge von Input- und Output-Fields zugeordnet. Dabei sind Input-Fields im Falle von Scripts als Übergabeparameter aufzufassen. Output-Fields sind die auszulesenden Messgrößen, d.h. die Werte, die Cacti mit jedem Polleraufruf aus dem Ausgabestrom des Scripts ermittelt. Es muss mindestens ein Wert ermittelt werden. In Folge muss auch mindestens ein Output-Field einer Data-Input-Method zugeordnet sein. Die Anzahl an zugeordneten Input-Fields ist jedoch beliebig.

Das Script aus dem obigen Beispiel hat keine Eingabeparameter und einen Ausgabewert, d.h. es muss ein Output-Field angelegt werden, wie in Abbildung 2 gezeigt. Nach diesem Schritt ist Cacti in der Lage, den einzelnen Wert mit jedem Polleraufruf abzufragen und ihn anschließend in einem Archiv zu speichern sowie in einem Graph darzustellen.

Beispiel Auslesen von Ping-Statistiken:

Zur Demonstration, wie mehrere Input- und Output-Fields genutzt werden können, sollen mit einer Data-Input-Method Ping-Statistiken ausgelesen werden. Einem Host werden mehrere Echo-Request-Pakete gesendet. In den Sta-

tistiken wird ausgewertet, wie viele davon nicht durch Echo-Reply-Pakete von ihm beantwortet wurden (Paketverlust) und welche Zeit insgesamt dazu benötigt wurde.

Eine Anfrage sieht wie folgt aus:

```
ping -c 5 google.de
```

Hier werden dem Host *google.de* fünf Echo-Request-Pakete gesendet. Der Ping-Befehl gibt daraufhin folgendes aus:

```
...
—— google.de ping statistics ——
5 packets transmitted, 5 received, 0% packet loss, time 4013
    ms
rtt min/avg/max/mdev = 14.578/20.870/25.298/4.008 ms
```

Für das Beispiel ist die Zeile interessant, in der der Paketverlust und die benötigte Zeit steht.

Im vorherigen Beispiel wurde nur ein Wert ausgelesen. Sollen mehrere Werte ausgelesen werden, d. h. der Data-Input-Method sind auch mehrere Output-Fields zugeordnet, erwartet Cacti den einzulesenden Ausgabestrom in folgendem Format:

```
<fld_1>:<val_1> <fld_2>:<val_2> ... <fld_n>:<val_n>
```

Also eine leerzeichen-separierte Liste von Feldname-Wert-Paaren.

Für das Umformatieren obiger Ausgabe bietet sich das Programm „awk“ an. Es kann Textstellen mit regulären Ausdrücken aufsuchen und in ein anderes Format ausgeben:

```
ping -c <packageCount> <host> |
awk '/transmitted/ {match($0,
    /([[:digit:]]+)% packet loss, time ([[:digit:]]+)ms/,
    groups);
    printf("loss:%d time:%d", groups[1], groups[2]) }'
```

Mittels Pipe „|“ wird die Ausgabe von ping an awk weitergeleitet. Daraufhin sucht awk alle Zeilen, die das Wort „transmitted“ enthalten. Bei jeder gefundenen Zeile werden die in geschweiften Klammern „{...}“ angegebenen Aktionen ausgeführt. Mittels „match“-Befehl werden die interessanten Werte in das Array „groups“ geschrieben, und mittels „printf“-Befehl neu formatiert. Letztlich sieht die umformatierte Ausgabe wie folgt aus:

```
loss:0 time:4013
```

Damit die Abfrage flexibel bleibt, wurden die Übergabeparameter in spitzen Klammern „<...>“ angegeben: `-c <packageCount> <host>`. Diese müssen

durch Input-Fields modelliert werden. Beim Anlegen einer Data-Source für das Speichern der Werte müssen den Input-Fields dann konkrete Argumente zugeordnet werden. Zudem ist es möglich, einen regulären Ausdruck den Input-Fields zuzuordnen. Damit prüft Cacti, ob die Argumente einem bestimmten Format entsprechen. Im Beispiel ist der reguläre Ausdruck `[0-9]+` für „packageCount“ sinnvoll, damit nur positive, ganzzahlige Werte übergeben werden dürfen.

Zusammengefasst besteht die Data-Input-Method aus:

- Input-Field „packageCount“: Anzahl an Echo-Request-Paketen
- Input-Field „host“: Host, an den die Pakete gesendet werden sollen
- Output-Field „loss“: Paketverlust in Prozent
- Output-Field „time“: Benötigte Zeit in Millisekunden

Damit ist es möglich, von einem beliebigen Host Ping-Statistiken abzufragen und in einem Graphen darzustellen.

3.2 Abfrage SNMP-basierter Messwerte

3.2.1 SNMP-Grundlagen

SNMP ist die Abkürzung für *Simple Network Management Protocol*. In den Aufgabenbereich dieses Protokolls fällt die Verwaltung entfernter, netzwerkfähiger Geräte. Es können einerseits Statusinformationen abgefragt, andererseits Konfigurationen getroffen werden. Beispielsweise ist es möglich, entfernt die Temperatur eines Switches abzufragen oder eine Netzwerkkarte eines Routers zu deaktivieren. Das Adjektiv „simple“ bezieht sich auf die Menge an SNMP-Operationen. Das Protokoll liegt aktuell (Januar 2012) in der Version 3 vor und wird im Request for Comments (RFC) 3410 [1] beschrieben. Entwickelt wurde SNMP von der Internet Engineering Task Force (IETF), die auch für das Internet Protocol (IP) und Transmission Control Protocol (TCP) zuständig ist.

Aufgrund der weiten Verbreitung von SNMP bei netzwerkfähigen Geräten, wie Router, Switches, Drucker aber auch Datenbanksystemen sollen in diesem Abschnitt die Grundlagen zu SNMP gelegt werden. Aufbauend darauf wird im nächsten Abschnitt der Einsatz in Cacti gezeigt.

Kommunikationspartner

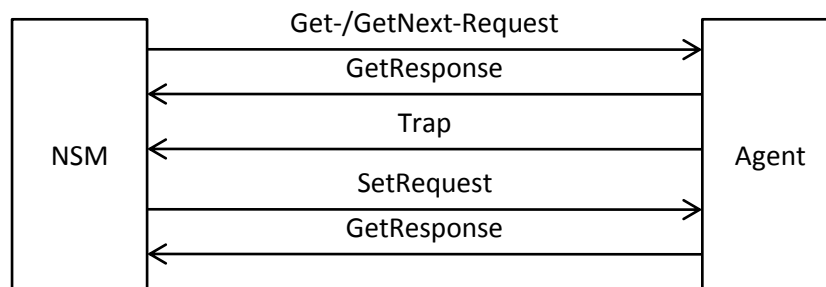


Abbildung 3: Die zwei SNMP-Kommunikationspartner

Grundsätzlich werden in SNMP zwei Kommunikationspartner definiert: *Managers* und *Agents* (Abbildung 3). Ersterer wird auch als „Network Management Station“ (NMS) bezeichnet. Er ist für das Polling (mittels Get-Anfragen) und das Empfangen von sogenannten „Traps“ zuständig. Ein Agent ist ein Programm, das auf dem zu verwaltenden Netzwerkgerät läuft. Er stellt den NMS Verwaltungsinformationen über das Gerät zur Verfügung. Beispielsweise überwacht er permanent den Status einer Netzwerkkarte in einem Router und kann die aktuelle Übertragungsgeschwindigkeit bereitstellen. Weitere Aufgaben sind das Versenden von Traps und das Treffen von Einstellungen, wobei diese Aufgaben nicht in den Bereich von Cacti fallen. Traps sind asynchron versendete Benachrichtigungen an die NMS, bspw. dass ein Switch gerade hoch/herunter gefahren ist. Sie werden eigenständig von den Agents versendet. Da sie nicht von der NMS bestätigt werden, gibt es jedoch keine Garantie für den Empfang.

Managed-Objects

In der Terminologie von SNMP werden die Werte, die abgefragt bzw. gesetzt werden können, als „Managed-Objects“ bezeichnet. Beispielsweise wird die aktuelle Anzahl an gestarteten Prozessen als ein Managed-Object bezeichnet. Nach [3] lassen sich Managed-Objects durch folgende drei Eigenschaften beschreiben:

1. Name: Der Name, auch „Object-Identifizier“ (OID) genannt, identifiziert ein Managed-Object eindeutig. Dabei sind zwei Notationen für OIDs gebräuchlich: Eine 1) numerische und eine 2) textuelle Notation. Beispiel: `.1.3.6.1.2.1.25.1.6.0` (numerisch) und `iso.org.dod.internet.mgmt.mib-2.host.hrSystem.hrSystemProcesses` (textuell). Beide OIDs identifizieren dasselbe Managed-Object.
2. Typ und Syntax: Jedem Managed-Object ist ein Datentyp zugeordnet. Datentypen werden durch die Beschreibungssprache „Structure of Ma-

nagement Information“ (SMI), einer Teilsprache der „Abstract Syntax Notation One“ (ASN.1), definiert. Neben den vordefinierten Datentypen, etwa „BOOLEAN“, „INTEGER“ und „UTF8String“, lassen sich mit ASN.1 komplexe Datentypen eindeutig beschreiben.

3. **Enkodierung:** Die mit ASN.1 beschriebenen Datentypen sind plattformunabhängig. Deshalb ist eine Zuordnung der in ASN.1 definierten Strukturen (abstrakte Syntax) zu Bit-Mustern für die Übertragung (Transfer-Syntax) nötig. Bei SNMP erfolgt die Kodierung der abstrakten Syntax zur Transfer-Syntax durch sogenannte „Encoding Rules“, die in den „Basic Encoding Rules“ (BER) definiert sind. Mit BER wird also festgelegt, wie Managed-Objects enkodiert/dekodiert werden, damit sie über ein Medium verschickt werden können.

OIDs werden in einer Baum-Struktur verwaltet. Es liegt daher nahe, jedem Knoten nach seiner Position im Baum zu benennen. Von der Wurzel an beginnend, setzt sich die OID eines Knotens aus seinen Namen und den Namen seiner Vorgänger zusammen, wobei sie jeweils durch einen Punkt „.“ getrennt sind. Im Beispiel `iso.org.dod.internet.mgmt.mib-2.host.hrSystem.hrSystemProcesses` ist `iso` die Wurzel, `org` der sechste Vorgänger, `dod` der fünfte Vorgänger usw. im Baum.

SNMP-Operationen

SNMPv1 definiert folgende Operationen:

- **GetRequest:** Die **GetRequest**-Operation wird von der NMS an den Agent gesendet. Die Anfrage enthält eine Liste von OIDs. Im Erfolgsfall antwortet der Agent mit einem **GetResponse** und der zugehörigen Liste von angeforderten Managed-Objects.
- **GetNextRequest:** Diese Operation dient vor allem dem zeilenweisen Traversieren von Tabellenstrukturen. Sie arbeitet wie **GetRequest**, jedoch wird jeweils der *Nachfolger* der angeforderten Managed-Objects zurückgegeben. Dazu sind Spalteneinträge zeilenweise, aufsteigend durchnummeriert. Im Erfolgsfall antwortet der Agent ebenfalls mit einem **GetResponse**.
- **SetRequest:** Mit dem **SetRequest**-Befehl kann eine NMS den Wert eines oder mehrere Managed-Objects ändern. Dazu sendet sie eine Liste von OID-Wert-Paaren an den Agent. Der Agent antwortet im Erfolgsfall ebenfalls mit einem („leeren“) **GetResponse**, wobei der Fehlerstatus auf „noError“ gesetzt wird.

- **Trap:** Ein Trap wird von einem Agent an eine Liste ihm bekannter NMS gesendet. Damit signalisiert er ein Ereignis, bspw. dass ein „Kaltstart“ geschehen ist (und damit einige Einstellungen ihre Gültigkeit verloren haben könnten). Traps werden nicht von NMS bestätigt.

Cacti nutzt die **GetRequest**- und **GetNextRequest**-Operationen für die Abfrage einzelner, skalarer Werte bzw. indexbasierte Werte in einer Tabellenstruktur.

Einordnung im TCP/IP-Stack

Im TCP/IP-Stack ist SNMP auf der Anwendungsebene einzuordnen, wie in Abbildung 4 dargestellt. Für die Übermittlung von Daten zwischen NMS und Agents setzt SNMP auf das verbindungslose Transportprotokoll UDP. Im Vergleich zum verbindungsorientierten TCP entfällt der „Overhead“ u. a. für den Verbindungsauf- und Abbau. Durch diesen Aspekt kann nicht garantiert werden, dass Datagramme erfolgreich empfangen werden. Stattdessen sendet die NMS eine Anfrage an den Agent und wartet auf seine Antwort. Nach Überschreiten eines „Timeouts“ wird ein Fehler bei der Übertragung angenommen, und ggf. die Anfrage erneut versendet. Bezugnehmend auf Traps gibt es jedoch keine Garantie für den Agent, ob ein Trap von einer NMS empfangen wurde.

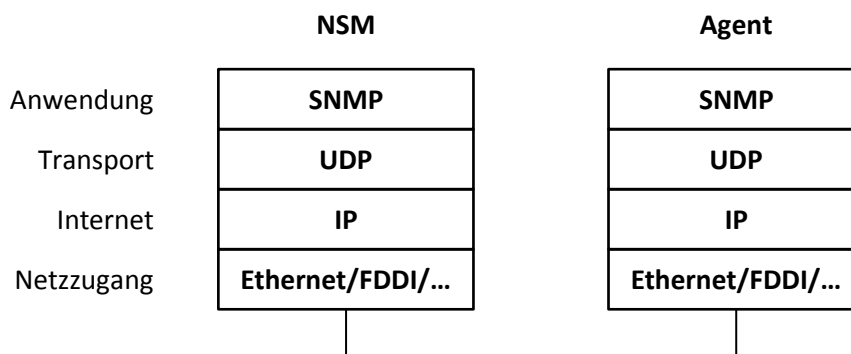


Abbildung 4: Einordnung von SNMP im TCP/IP-Stack

Standardmäßig empfängt ein Agent Anfragen auf UDP-Port 161 und versendet Antworten zurück zum Quell-Port; Traps werden auf UDP-Port 162 versendet. Sowohl Timeout als auch der UDP-Port (Standard 161) können in Cacti festgelegt werden.

Communitys

In den Versionen SNMPv1 und SNMPv2 erfüllen Communitys den gleichen Zweck wie Passwörter. Empfängt ein Agent eine Anfrage mit einer ihm un-

bekannten Community, wird er auf die Anfrage nicht reagieren; empfängt eine NSM einen Trap mit einer ihr unbekannten Community, wird sie den Trap verwerfen. Ein Agent definiert dazu drei Communitys: 1) „Read-Only“, 2) „Read-Write“ und 3) „Trap“. Wie die Namen bereits vermuten lassen, ist 1) für den Lesezugriff (Get-Anfragen), 2) für den Lese- und Schreibzugriff (Get- und Set-Anfragen) sowie 3) für das Versenden von Traps zuständig. Typischerweise werden 1) und 2) auf „public“ bzw. „private“ gesetzt.

An dieser Stelle sei darauf hingewiesen, dass eine Community im Klartext übertragen wird. Es besteht also die Gefahr, dass ein Dritter Pakete mitliest, und aus dem Paket-Header die Community ermittelt.

3.2.2 Abfrage eines Managed-Objects

Für die Abfrage eines Managed-Objects bietet Cacti bereits eine generische Data-Input-Method namens „Get SNMP Data“ mit Input-Type „SNMP“ an. Sie fragt mittels `GetRequest` ein Managed-Object zu einer gegebenen OID ab. Durch Anlegen einer Data-Source mit „Get SNMP Data“ werden die zugehörigen Input-Fields der Data-Input-Method sichtbar, wie in Abbildung 5 dargestellt.

Custom Data [data input: Get SNMP Data]	
OID	<input type="text" value=".1.3.6.1.2.1.1.7.0"/>
SNMP Authenticaion Protocol (v3)	<input type="text"/>
SNMP Community	<input type="text" value="public"/>
SNMP IP Address	<input type="text" value="127.0.0.1"/>
SNMP Password	<input type="text"/>
SNMP Port	<input type="text"/>
SNMP Privacy Passphrase (v3)	<input type="text"/>
SNMP Privacy Protocol (v3)	<input type="text"/>
SNMP Username	<input type="text"/>
SNMP Version (1, 2, or 3)	<input type="text" value="1"/>

Abbildung 5: Input-Fields von „Get SNMP Data“

Wichtig sind die Input-Fields 1) „OID“, 2) „SNMP Community“, 3) „SNMP IP Address“ und 4) „SNMP Port“. Ihnen werden 1) die OID des abzufragenden Managed-Objects, 2) die Community für den Lesezugriff (Standard „public“),

3) die IP-Adresse des SNMP-Agents und 4) der UDP-Port des Agents für den Empfang von Anfragen (Standard 161) übergeben.

3.2.3 Abfrage von Tabellen

Im Gegensatz zu Data-Input-Methods werden Data-Queryys für die Abfrage von indexbasierten Listen/Tabellen genutzt. Eingesetzt werden Data-Queryys bspw. bei der Abfrage von Netzwerkkarten eines Switchs, oder von Partitionen eines File-Servers, wie in folgender Tabelle:

Index	Descr	AllocationUnits	Size	Used
1	C:\	4096	20479999	12520394
2	D:\	4096	57636351	9515705
3	E:\	0	0	0
4	F:\	0	0	0
5	Virtual Memory	65536	129919	57153
6	Physical Memory	65536	64974	50965

Dabei ist „AllocationUnits“ die Blockgröße in Byte, „Size“ der verfügbare und „Used“ der verwendete Speicherplatz in Blöcken. Über SNMP lässt sich diese Tabelle abfragen. Die Werte befinden sich dazu im Teilbaum von .1.3.6.1.2.1.25.2.3.1.

Konfiguriert werden Data-Queryys über eine XML-Datei, die das Schema der Tabelle festlegt. Sie hat folgenden Aufbau:

```
<query>
  <name>Get SNMP Storage Areas</name>
  <description>Liste von Partitionen</description>
  <oid_index>.1.3.6.1.2.1.25.2.3.1.1</oid_index>
  <index_order_type>numeric</index_order_type>
  <fields>...</fields>
</query>
```

Im Element `oid_index` ist die OID anzugeben, mit der die Indizes der Tabellenzeilen abgefragt werden. Die Sortierung der Zeilen kann bezüglich des Index entweder numerisch (`index_order_type = numeric`) oder alphabetisch (`index_order_type = alphabetic`) erfolgen.

Die Spalten werden im Element `fields` festgelegt:

```
<fields>
  <hrStorageDescr>
    <name>Description</name>
    <method>walk</method>
    <source>value</source>
    <direction>input</direction>
```

```

        <oid>.1.3.6.1.2.1.25.2.3.1.3</oid>
    </hrStorageDescr>
    ...
</fields>

```

Interessant sind hier die zwei Elemente **direction** und **oid**. Ersteres kann entweder auf „input“ oder „output“ gesetzt werden. Input-Spalten werden dem Nutzer für die Auswahl der Tabellenzeilen angezeigt, bspw. wenn er eine bestimmte Partition überwachen möchte; Output-Spalten hingegen sind die Werte, die in einem Graphen dargestellt werden können, bspw. „Size“ und „Used“. Im zweiten Element **oid** ist die zur Spalte zugehörige OID anzugeben.

Per **GetNext** fragt Cacti die Werte zeilenweise für jede Spalte ab. Zur Veranschaulichung soll hier die Spalte „Descr“ mit dem Werkzeug „net-snmp“ abgefragt werden:

```

snmpgetnext -On -c public -v 2c localhost
.1.3.6.1.2.1.25.2.3.1.3
> .1.3.6.1.2.1.25.2.3.1.3.1 = STRING: C:\
snmpgetnext -On -c public -v 2c localhost
.1.3.6.1.2.1.25.2.3.1.3.1
> .1.3.6.1.2.1.25.2.3.1.3.2 = STRING: D:\
...
snmpgetnext -On -c public -v 2c localhost
.1.3.6.1.2.1.25.2.3.1.3.5
> .1.3.6.1.2.1.25.2.3.1.3.6 = STRING: Physical Memory

```

Wobei hier die Community auf „public“ und die Version auf v2 (SNMPV2) gesetzt wurde.

Nach Anlegen einer Data-Query kann der Benutzer für jede Zeile (hier Partition) eine Data-Source anlegen, zu jeder gewünschten Output-Spalte ein Data-Source-Item hinzufügen und diese zur Visualisierung Graph-Items zuordnen. Letztlich ist es damit möglich, jede Partition einzeln in einem Graph zu überwachen.

4 Messdatenspeicherung

Die mittels Data-Input-Methods und Data-Querys erfassten Messwerte können mit Cacti über lange Zeiträume gespeichert und visualisiert werden. Für beide Aufgaben nutzt Cacti das RRDTool. Hinsichtlich der Messdatenspeicherung bietet RRDTool den Vorteil, dass der Speicherplatz stets Konstant bleibt. Der Kompromiss dabei ist, dass viele ältere Messwerte zu wenigen konsolidierten Messwerten zusammengefasst werden. Auf welche Weise das

geschieht, wird im nächsten Abschnitt erläutert.

4.1 RRDTool-Grundlagen

Ein Ringpuffer ist eine Speicherstruktur mit fester Größe. Es steht eine feste Anzahl an Speicherplätzen bzw. Elementen zur Verfügung. Diese Elemente sind wie in einem Array benachbart. Im Unterschied zum Array sind auch das erste und das letzte Element benachbart, sodass eine Ringstruktur entsteht. Ein Pointer zeigt auf das aktuelle Element und wird zum Schreiben um je eine Position weiter verrückt. Zeigt der Pointer auf das letzte Element, wird er wieder auf das erste Element gesetzt. In Konsequenz wird (nach einem Pufferüberlauf) der jeweils älteste Wert durch den neusten Wert überschrieben.

Dieses Prinzip wendet RRDTool bei der Speicherung der Messwerte in sogenannten „Round-Robin-Archiven“ (RRAs) an. In Abbildung 6a ist ein RRA mit sechs Datenpunkten veranschaulicht. Um 01:01 Uhr wurde der erste Messwert 6,1 eingetragen, um 01:02 Uhr der zweite Messwert 4,2 usw.; insgesamt wurden sechs Messwerte eingetragen. Der Pointer zeigt auf den letzten Eintrag, d. h. der Puffer ist voll. In 6b ist der Zustand dargestellt, nach dem der siebte Messwert 3,9 um 01:07 Uhr hinzugefügt wurde. Es ist zu erkennen, dass der älteste Messwert (um 01:01 Uhr) durch den neusten (um 01:07 Uhr) überschrieben wurde.

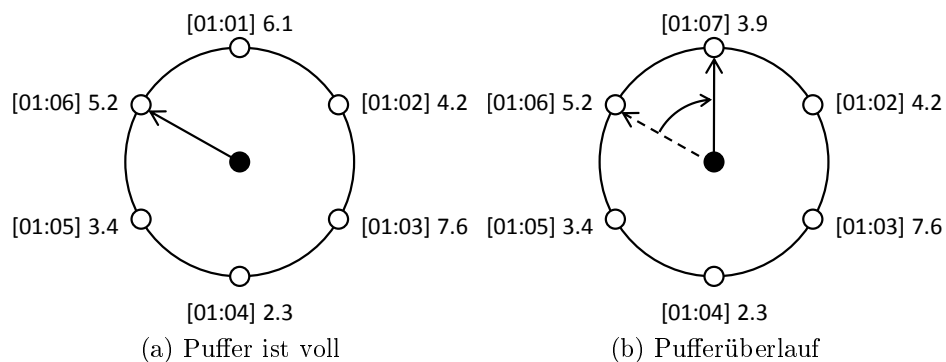


Abbildung 6: Ringpufferstruktur eines RRAs. Angelehnt an [4]

Das RRA in Abbildung 6 speichert die Messwerte der letzten sechs Minuten in einer Zeitauflösung von einer Minute ab. Ist man an einem größeren Zeitraum, bspw. einer Stunde interessiert, muss nicht die Größe des RRAs erhöht werden. Stattdessen sollte ein weiteres RRA hinzugefügt werden, dass bspw.

zehn Datenpunkte mit einer Zeitauflösung von sechs Minuten speichert. Je sechs Datenpunkte aus dem ersten RRA würden so zu einem Datenpunkt des zweiten RRAs zusammengefasst, bspw. in dem der Durchschnitt oder das Maximum gebildet wird. Durch Hinzufügen weiterer RRAs ließen sich so die Messwerte über Tage, Monate usw. ablegen. Da jedes RRA eine feste Anzahl an Datenpunkten besitzt, ist der benötigte Speicherplatz stets konstant. Der Kompromiss dabei ist, dass ältere Messwerte nicht in der Zeitauflösung verfügbar sind, wie aktuelle.

Das obige Beispiel zeigt nur das Speicherverhalten für *eine* Messgröße, tatsächlich können in einem RRA auch Werte für *mehrere* Messgrößen gespeichert werden, bspw. für die Anzahl empfangener und gesendeter Byte einer Netzwerkkarte. Diese Messgrößen werden bei RRDTool als „Data-Sources“ (DS) bezeichnet. Für jede DS verwaltet RRDTool je ein Ringpuffer pro RRA (Anzahl der Datenpunkte sowie Zeitauflösung bleibt pro RRA identisch).

An dieser Stelle sei auf die unterschiedliche Begriffsbelegung von „Data-Source“ in Cacti und RRDTool hingewiesen. Im Bezug auf Cacti wird daher im weiteren Verlauf „Data-Source“ ausgeschrieben; im Bezug auf RRDTool die dort gebräuchliche Abkürzung „DS“ verwendet.

Zusammengefasst werden DS und zugehörige RRAs zu „Round-Robin-Datenbanken“ (RRDs). RRDTool verwaltet sie in Form von Dateien. Im Allgemeinen sind sie an der Dateiendung „.rrd“ zu erkennen. Zur Definition einer RRD gehören neben den Angaben zu DS und RRAs auch die zwei Parameter „Start-Time“ und „Step“. Messwerte, die älter sind als die Start-Time (Angabe in Sekunden), werden ignoriert. Der Parameter Step gibt den erwarteten zeitlichen Abstand (in Sekunden) zwischen zwei Messwerten an. Wird er unterschritten, werden die Messwerte zu einem Durchschnittswert zusammengefasst; wird er überschritten, werden Zwischenwerte aus dem letzten und aktuellen Messwert interpoliert.

4.1.1 Data-Source

Eine RRD kann Messwerte aus mehreren DS aufnehmen, bspw. die Anzahl empfangener und gesendeter Byte einer Netzwerkkarte. Innerhalb der RRD werden DS über ihren Namen identifiziert. Weitere Parameter sind „Min-“ sowie „Max-Value“, „Data-Source-Type“ (DST) und „Heartbeat“.

Min- und Max-Value legen den kleinsten bzw. größten gültigen Wert fest. Werden diese von einem Messwert unterschritten bzw. überschritten, wird er als „UNKNOWN“, d. h. als „ungültig“, in das RRA eingetragen.

Die Wahl des DSTs hängt von der Art der Messgröße ab, bspw. ob es sich um

eine kontinuierlich wachsende Größe handelt. In Cacti stehen vier DSTs zur Auswahl: „GAUGE“, „COUNTER“, „DERIVE“ und „ABSOLUTE“. Sie unterscheiden sich dahingehend, wie der übergebene Messwert in ein RRA eingetragen wird. Seien (t_l, v_l) , (t_n, v_n) die Zeitpunkt-Messwert-Paare der letzten bzw. aktuellen Messung, wobei t_l und t_n in Sekunden angegeben sind. Der eingetragene Messwert v_u berechnet sich dann wie folgt:

- GAUGE: $v_u = v_n$, d. h. der aktuelle Messwert wird direkt übernommen.
- COUNTER:

$$v_u = \begin{cases} \frac{v_n - v_l + 2^{32}}{t_n - t_l}, & \text{wenn } -2^{32} \leq (v_n - v_l) < 0 \\ \frac{v_n - v_l + 2^{64}}{t_n - t_l}, & \text{wenn } -2^{64} \leq (v_n - v_l) < -2^{32} \\ \frac{v_n - v_l}{t_n - t_l}, & \text{sonst} \end{cases}$$

Dieser DST ist für kontinuierlich wachsende Größen (Zähler) geeignet, bspw. die Anzahl empfangener Byte. Im Normalfall gilt $v_n \geq v_l$, d. h. die Messgröße ist gewachsen oder gleichgeblieben. Das Besondere an COUNTER ist die Behandlung von Überläufen. Tritt ein Überlauf des Zählers ein, wird wieder ab Null begonnen zu zählen, also gilt $v_n - v_l < 0$. Die ersten beiden Fälle der Fallunterscheidung behandeln den Überlauf eines 32-Bit bzw. 64-Bit Zählers, der letzte Fall entspricht dem Normalfall (der Wert ist gestiegen oder gleich geblieben).

- DERIVE: $v_u = \frac{v_n - v_l}{t_n - t_l}$, wie COUNTER aber ohne Überlaufbehandlung.
- ABSOLUTE: $v_u = \frac{v_n}{t_n - t_l}$, ist geeignet für Zählgrößen, die während der Messung auf Null zurück gesetzt werden.

Wie oben angegeben, erwartet RRDTool die Messwerte in einem gewissen Zeitabstand, der durch den Parameter Step angegeben wird. Mit dem Parameter Heartbeat wird festgelegt, in welchen Zeitabstand höchstens zwei Messungen auseinander liegen dürfen, bis die aktuelle als ungültig gespeichert wird. Bei einer späteren Datenabfrage wird angenommen, dass die Messwerte im Zeitabstand Step eingetragen wurden.

4.1.2 Round-Robin-Archive

Ein RRA besteht aus mehreren Ringpuffern (für jede DS einer). Darin werden die Messdaten mit gleicher Zeitaufösung gehalten. Die größte wählbare Zeitaufösung, d. h. der kleinste Zeitabstand, gibt die übergeordnete RRD

mit dem Parameter Step an. Messdaten, die ggf. zusammengefasst (Zeitabstand war kleiner als Step) oder interpoliert (Zeitabstand war größer als Step) wurden, werden als „Primary-Data-Points“ (PDPs) bezeichnet. Pro RRA muss durch den Parameter „Steps“ angegeben werden, wie viele PDPs zu einem „Consolidated-Data-Point“ (CDP) zusammengefasst werden. Diese CDPs sind die Einträge in den Ringpuffern.

Insgesamt gehören folgende Parameter zur Definition eines RRAs: „Rows“, „Steps“, „Consolidation-Function“ (CF) und „X-Files-Factor“ (XFF). Ein Ringpuffer speichert eine feste Anzahl an Elementen (hier CDPs) ab. Die Anzahl an CDPs legt der Parameter Rows fest. Werden mehr CDPs in einem Ringpuffer abgelegt, als in Rows angegeben, werden ältere Einträge durch neuere überschrieben.

Steps legt fest, wie viele PDPs zu einem CDP zusammengefasst werden. Demzufolge kann ein RRA Messdaten in einer Zeitspanne von $\text{Step} * \text{Steps} * \text{Rows}$ Sekunden archivieren.

Wie die PDPs zu einem CDP zusammengefasst (konsolidiert) werden, legt die CF fest. Folgende vier CFs werden angeboten:

1. MIN: $\text{CDP} = \min(\text{PDP}_1, \text{PDP}_2, \dots, \text{PDP}_{\text{Steps}})$, d. h. der *kleinste* PDP wird ausgewählt.
2. MAX: $\text{CDP} = \max(\text{PDP}_1, \text{PDP}_2, \dots, \text{PDP}_{\text{Steps}})$, d. h. der *größte* PDP wird ausgewählt.
3. LAST: $\text{CDP} = \text{PDP}_{\text{Steps}}$, d. h. der *letzte* PDP wird ausgewählt.
4. AVERAGE: $\text{CDP} = \frac{1}{\text{Steps}} \sum_{i=1}^{\text{Steps}} \text{PDP}_i$, d. h. es wird der Durchschnittswert aus allen PDPs berechnet.

Während der Messung können mehrere PDPs als ungültig gespeichert wurden sein, bspw. weil der Min- oder Max-Value unter- bzw. überschritten wurde. Der Parameter $\text{XFF} \in [0, 1)$ gibt das maximale Verhältnis von ungültigen zu gültigen PDPs an. Wird es unterschritten, wird der CDP selber als ungültig gespeichert.

Beispiel Eine RRD mit zwei RRAs:

Zur Veranschaulichung ist in Abbildung 7 eine Beispiel-RRD mit $\text{Step} = 60$ s, sechs Messwerten und zwei RRAs angegeben. Im oberen Bereich wird gezeigt, wie die Messwerte auf PDPs abgebildet werden. Der Messwert zum Zeitpunkt 90 s hat zum vorhergehenden Messwert nur 30 s Abstand und unterschreitet damit den Step-Wert von 60 s. Daher wird er mit dem nächsten

Messwert (Zeitpunkt 120 s) zum PDP 3 = $\frac{1+5}{2}$ zusammengefasst. Alle weiteren Messwerte haben einen Abstand von 60 s und können damit direkt auf PDPs abgebildet werden.

Das erste RRA speichert Rows = 5 CDPs. Da Steps = 1 gesetzt wurde, wurden die PDPs direkt auf die CDPs abgebildet. Mit 5 CDPs ist der Ringpuffer voll. Jeder weitere Eintrag eines CDPs würde ältere CDPs überschreiben.

Beim zweiten RRA wird die CF = AVERAGE benötigt. Es werden jeweils aus Steps = 5 PDPs ein CDP gebildet, in dem der Durchschnitt berechnet wird. Hier ergibt sich der erste CDP 5,8 aus $\frac{4+3+6+9+7}{5}$. Alle vier weiteren CDPs im RRA sind noch nicht belegt, und daher auf „Not a Number“ (dt. „keine Zahl“, NaN) gesetzt.

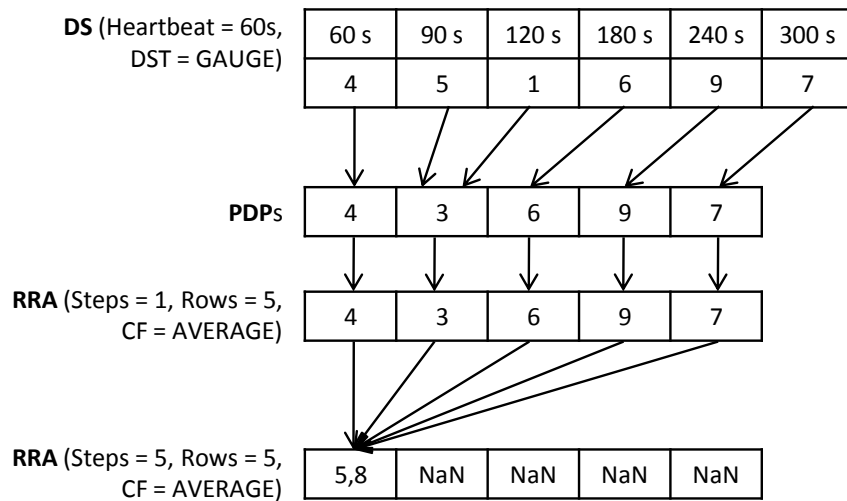
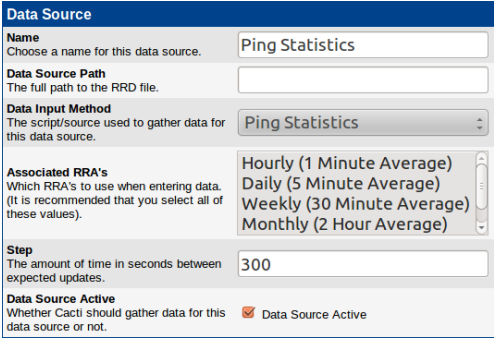


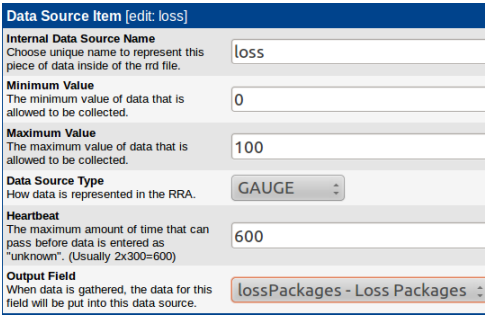
Abbildung 7: Beispiel-RRD mit zwei RRAs und Step = 60 s

4.2 Zusammenhang zwischen RRDTool und Cacti

Die Unterteilung in DS und RRAs findet sich in Cacti wieder; jedoch unter anderen Begriffen. Was in RRDTool mit RRD bezeichnet wird, heißt in Cacti Data-Source. Daher ist jeder Data-Source eine RRD-Datei zugeordnet. Eine Data-Source ist beschränkt auf eine Data-Input-Method, d. h. es können nicht Messwerte aus unterschiedlichen Data-Input-Methods gespeichert werden. Jede Data-Input-Method besitzt mindestens ein Output-Field. Zur Speicherung der Messwerte eines jeden Output-Fields ist ein zugehöriger Data-Source-Item in der Data-Source anzulegen. Ein Data-Source-Item entspricht einer DS in RRDTool. Die zugehörigen Parameter für Data-Source und Data-Source-Item sind in Abbildung 8a bzw. 8b dargestellt.



(a) Data-Source



(b) Data-Source-Item

Abbildung 8: Parameter von Data-Source und Data-Source-Item

Bei „Data-Source-Path“ wird der Pfad zur RRD-Datei angegeben. Der Parameter „Internal-Data-Source-Name“ gibt den Namen der RRDTool-DS an. Auch weitere Parameter finden ihre Entsprechung in RRDTool:

- Associated-RRAs: Entspricht den RRA-Definitionen in einer RRD. Die von Cacti angebotenen RRAs haben den DST = AVERAGE. Jedoch lassen sich weitere RRA-Vorgaben mit anderen Steps-, Rows- und CF-Argumenten anlegen.
- Step: Entspricht dem Step-Parameter in einer RRD.
- Minimum- und Maximum-Value: Entspricht den Parametern Min- und Max-Value einer DS in RRDTool.
- Data-Source-Type: Entspricht dem DST-Parameter einer DS.
- Heartbeat: Entspricht dem Heartbeat-Parameter einer DS.

5 Messdatendarstellung

Neben der Messdatenspeicherung setzt Cacti auch bei der Messdatendarstellung auf RRDTool. In regelmäßigen Abständen werden Diagramme – in Cacti als „Graphs“ bezeichnet – mit RRDTool erstellt und auf der Weboberfläche dargestellt. Als Bildformate werden „Portable Network Graphics“ (PNG), „Graphics Interchange Format“ (GIF) und das Vektorgrafikformat „Scalable Vector Graphics“ (SVG) angeboten. Bei der Erstellung eines Graphs können u. a. folgende Parameter angegeben werden:

- Title: Titel-Text, der oberhalb des Graphs positioniert wird.
- Vertical-Label: Vertikal platzierter Text auf der linken Seite des Graphs.
- Width und Height: Breite und Höhe des Graphs in Pixel.
- Skalierung der Y-Achse: Entweder skaliert RRDTool die Y-Achse automatisch oder man gibt eine untere (Lower-Limit) und/oder obere Grenze (Upper-Limit) an. Desweiteren ist eine logarithmische Skalierung der Y-Werte möglich.

Die Messwerte können als Linien- oder Flächendiagramme dargestellt werden. Jede einzuzeichnende Komponente ist als sogenannter „Graph-Item“ dem Graph hinzuzufügen. Dazu gehört bspw. „LINE“, der Datenpunkte durch eine Linie verbindet (Liniendiagramm). Darüber hinaus ist es möglich, Funktionen auf Datenpunkte anzuwenden. Das wird nötig, wenn bspw. die Übertragungsgeschwindigkeit in Byte/s gegeben ist und im Graph in Bit/s dargestellt werden soll. Hier müssen die Werte erst mit 8 multipliziert werden. Für diese und weitere Berechnungen bietet RRDTool die sogenannten „CDEF-Anweisungen“. Diese führen Berechnungen elementweise auf einer Werteliste aus und weisen das Ergebnis neuen Variablen zu.

Im Folgenden werden Graph-Items und CDEF-Anweisungen vorgestellt.

5.1 Graph-Items

Obwohl Cacti auf Linien- und Flächendiagramme begrenzt ist, bleibt die Gestaltung eines Graphs durch die Kombinationsmöglichkeiten verschiedener Graph-Items flexibel. Graph-Items sind Graphs untergeordnet und modellieren darzustellende Komponenten im Graph. Zu den angebotenen Graph-Item-Typen gehören u. a.:

- LINE: Die Datenpunkte eines Data-Source-Items bzw. einer CDEF-Variablen werden durch eine Linie verbunden (Liniendiagramm). Es kann festgelegt werden, mit welcher Farbe und welcher Dicke (1, 2, oder 3 Pixel) die Linie zu zeichnen ist. Cacti erstellt automatisch einen Eintrag mit Linienfarbe und Text in der Legende des Graphs.
- AREA: Die Fläche zwischen den verbundenen Datenpunkten und der X-Achse wird gefüllt (Flächendiagramm). Auch hier lässt sich die Flächenfarbe angeben und es wird ein Eintrag in der Legende erstellt.

- HRULE: Zu einem festen Y-Wert wird eine horizontale Linie eingezeichnet. Diese kann bspw. einen kritischen Schwellwert (maximale Festplattenauslastung u. ä.) darstellen.
- VRULE: Zu einem festen X-Wert wird eine vertikale Linie eingezeichnet.
- GRPINT: Expliziter Eintrag in der Legende, um den größten, kleinsten, letzten Wert oder den Durchschnittswert eines Data-Source-Items bzw. einer CDEF-Variablen im Graph zu vermerken.

Durch Kombination aus zwei LINE-Graph-Items ist es bspw. möglich, die Sende- und Empfangsgeschwindigkeit einer Netzwerkkarte in einem Graph mit unterschiedlichen Farben darzustellen.

5.2 Berechnungen mittels CDEF-Anweisungen

CDEF-Anweisungen sind mathematische Funktionen, die auf Messwerte elementweise angewandt werden. Das Ergebnis wird in einer Variablen gespeichert. Über den Namen der Variablen kann die neu berechnete Datenreihe in den Graph gezeichnet oder erneut als Argument einer anderen CDEF-Anweisung übergeben werden. Der Berechnungsausdruck wird in der „Umgekehrt Polnischen Notation“ (engl. Reverse Polish Notation, kurz RPN) angegeben. Als Rechenmodell dient hier eine Stackmaschine. Auf den Stack sind zunächst die Operanden abzulegen (Push). Beim Ausführen einer Operation werden die Operanden vom Stack geholt (Pop), die Berechnung durchgeführt und das Ergebnis auf den Stack gelegt (Push). Beispiel: Der Infix-Ausdruck $3 + 4 * 5$ (Operator steht zw. den Operanden) ist äquivalent zum RPN-Ausdruck $4, 5, *, 3, +$ (Operator steht hinter den Operanden).

In Cacti werden diese CDEF-Anweisungen separat von den Graphs verwaltet. Zur Definition einer CDEF-Anweisung gehören ein eindeutiger Name sowie ein oder mehrere „CDEF-Items“. Diese CDEF-Items modellieren die Operanden bzw. Operatoren des RPN-Ausdrucks. Daher ist deren Reihenfolge zu beachten.

Beispiel Umrechnung von Byte/s in Bit/s

Angenommen die Übertragungsgeschwindigkeit einer Netzwerkkarte wird in Byte/s gemessen, soll aber in Bit/s angezeigt werden. Dafür ist eine CDEF-Anweisung mit den folgenden CDEF-Items zu erstellen:

1. CURRENT_DATA_SOURCE: CDEF-Anweisungen werden getrennt

von Graphs verwaltet. Bei der Definition einer CDEF-Anweisung ist daher nicht bekannt, auf welchem Data-Source-Item die Berechnung durchgeführt werden soll. Bei „CURRENT_DATA_SOURCE“ ersetzt Cacti automatisch diesen CDEF-Item durch den in einem Graph-Item angegebenen Data-Source-Item.

2. Custom String „8“: Operand „8“ für die Multiplikation, um Byte in Bit umzurechnen.
3. Operator „*“: Multiplikation mit der 8.

Letztlich wird damit der RPN-Ausdruck „CURRENT_DATA_SOURCE,8,*“ beschrieben, der auf jeden Messwert angewandt wird.

Zu den arithmetischen Operatoren werden auch boolesche Operatoren, trigonometrische Operatoren, Mengenoperatoren und weitere angeboten. Alle Berechnungen auf den Datenpunkten werden unabhängig voneinander durchgeführt.

Beispiel Farbiges hervorheben kritischer Übertragungsraten

Ein Mittel, um auf kritische Werte hinzuweisen, ist das farbige hervorheben dieser. Beispielsweise können „optimale“ Werte grün, normale Werte gelb und kritische Werte rot dargestellt werden. Sei r eine gemessene Übertragungsrate. Im Beispiel sollen Werte $r > 350\text{kb/s}$ dem optimalen Bereich, Werte $100\text{kb/s} < r \leq 350\text{kb/s}$ dem normalen Bereich und Werte $r \leq 100\text{kb/s}$ dem kritischen Bereich zugeordnet werden.

Generell ist das Zeichnen mehrfarbiger LINE- bzw. AREA-Graph-Items nicht möglich. Für die Umsetzung können jedoch mehrere Graph-Items unterschiedlicher Farbe miteinander kombiniert werden. Die Idee ist, Werte außerhalb eines Bereichs zu Null zu setzen. Dazu werden folgende drei CDEF-Anweisungen definiert:

```
optimal = CURRENT_DATA_SOURCE,350000,GT,
          CURRENT_DATA_SOURCE,0,IF
normal = CURRENT_DATA_SOURCE,350000,LE,CURRENT_DATA_SOURCE,
          100000,GT,1,EQ,EQ,CURRENT_DATA_SOURCE,0,IF
kritisch = CURRENT_DATA_SOURCE,100000,LE,
           CURRENT_DATA_SOURCE,0,IF
```

Der RPN-Ausdruck für die CDEF-Variable „normal“ ist wie folgt zu lesen:

1. Lege aktuellen Messwert und 350000 auf den Stack und vergleiche beide mittels „LE-Operator“ (engl. „lower or equal“). Dieser liest zwei Werte a, b und legt eine Eins auf den Stack, wenn $a \leq b$ gilt; sonst eine Null.

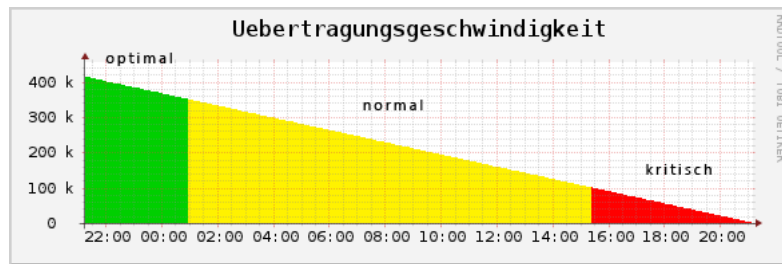


Abbildung 9: Mehrfarbiges Flächendiagramm

2. Analog ist „CURRENT_DATA_SOURCE,100000,GT“ zu lesen, wobei GT (engl. „greater than“) auf $a > b$ prüft.
3. Mittels 1,EQ,EQ wird eine Eins auf den Stack gelegt, wenn beide Bedingungen erfüllt sind; sonst eine Null. Dabei prüft „EQ“ auf $a = b$.
4. Der „IF-Operator“ stellt einen bedingten Ausdruck dar. Es gilt bei A,B,C,IF: Wenn $A = 1$ wird B, sonst C auf den Stack gelegt. Erfüllt hier der Messwert beide Bedingungen, wird er auf den Stack gelegt; sonst eine Null.

Für jede CDEF-Anweisung wird ein AREA-Graph-Item erstellt, und ihm grün für „optimal“, gelb für „normal“ und rot für „kritisch“ zugewiesen. In Abbildung 9 ist dieses Verhalten dargestellt.

6 Anwendungsbeispiel: Privater Homeserver

6.1 Einleitung

In diesem Kapitel soll der praktische Einsatz von Cacti anhand eines privaten Homeservers gezeigt werden. Über Systemprogramme lassen sich Hardwareressourcen, wie RAM-Verbrauch und Prozessorauslastung, abfragen. Auch Serverdienste, wie HTTP und FTP, stellen ihre Metriken typischerweise in Form von Log-Dateien zur Verfügung. Cacti ermöglicht es hier, diese Hardwareressourcen und Metriken zentral und einheitlich über eine Weboberfläche darzustellen. Im Folgenden wird die Konfiguration des betrachteten Homeservers vorgestellt und Anwendungsbeispiele von Cacti beim Überwachen des Homeservers beschrieben.

6.2 Konfiguration des Homeservers

Als Betriebssystem ist Linux auf dem Homeserver installiert. Gesichert wird der Homeserver über die Linux-eigene Firewall „iptables“. Installiert wurden verschiedene Server-Anwendungen. Dazu zählt der Apache2-Webserver, das OpenSSH-Programmpaket für die Unterstützung von SSH, ein FTP-Server, ein E-Mail-Server mit IMAP-Unterstützung, ein Jabber-Server zum Anbieten von Instant-Messaging und ein Ventrilo-Server zum Anbieten von Internet-Telefonie.

Zusätzlich werden weitere Programme für die Installation von Cacti vorausgesetzt: Cacti wurde zu einem Großteil in PHP entwickelt, weswegen ein entsprechender PHP-Interpreter benötigt wird. Ein MySQL-Server wird vorausgesetzt, um Einstellungen in Form von Tabellen zu speichern. Für die Messdatenspeicherung und -Visualisierung wird das RRDTool benötigt.

6.3 Anwendungsbeispiele von Cacti

In den folgenden Unterabschnitten werden Anwendungsbeispiele beschrieben, bei denen Cacti für das Monitoring des Homeservers eingesetzt wird. Alle Anwendungsbeispiele haben gemeinsam, dass die zu überwachenden Metriken aus Log-Dateien ermittelt werden können. Die Formate der Log-Dateien unterscheiden sich von Software zu Software. Cacti erwartet die Messwerte in einem bestimmten Format, wie in Abschnitt 3.1 beschrieben. Entsprechend sind die Messdaten für jedes Log-Datei-Format individuell in das von Cacti geforderte Format zu überführen. Der hier gewählte Ansatz zur Messdatenerfassung verarbeitet die Log-Dateien via Bash-Scripts. Mittels awk, grep, cat etc. werden die relevanten Messdaten ausgelesen und Cacti im gewünschten Format übergeben. Der Poller führt in festen Zeitintervallen diese Scripts aus und speichert die Messdaten in den RRDs ab. Anschließend lassen sich die Messdaten in Form von Diagrammen visualisieren.

Vorgestellt werden folgende Anwendungsbeispiele: Überwachung der Gesprächsteilnehmerzahl (Ventrilo-Server), Ermitteln des Verbrauchs eines Verzeichnisses (samt Unterverzeichnissen), die Zählung vollständiger Downloads, die Zählung missglückter Anmeldeversuche unter SSH, die Zählung vollständiger Downloads unter Berücksichtigung dass Logdateien archiviert werden und das Auslesen des Onlinestatus unter Jabber. Jedem Anwendungsbeispiel ist ein Bash-Script zugeordnet. Dabei sind die erforderlichen Benutzerrechte zum Ausführen der Bash-Scripts zu beachten. Ein Anwendungsbeispiel ist untergliedert in Überblick, Formatbeschreibung und Erläuterung zum Bash-

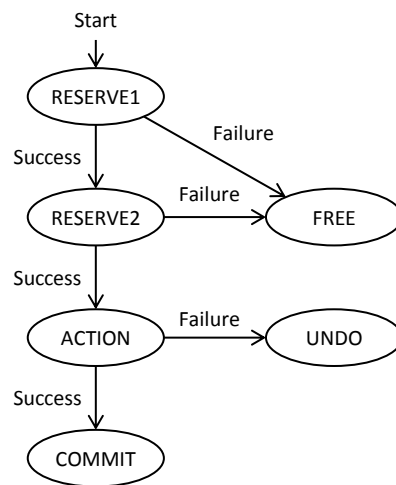


Abbildung 10: Zustandsdiagramm der Modi in Net-SNMP

Script.

6.3.1 Ventrilo: Aktuelle Teilnehmerzahl

6.3.2 Verbrauch eines Verzeichnisses samt Unterverzeichnissen

6.3.3 Apache2: Vollständige Downloads

6.3.4 OpenSSH: Zählen missglückter Anmeldeversuche

6.3.5 Firewall: Blockierte Eingangsversuche

6.3.6 Downloads-Messung unter Berücksichtigung alter Logdateien

6.3.7 Jabber: Onlinestatus eines Accounts

A Beispiel Anhang

```

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

#include "delayed_instance.h"

static u_long delay_time = 1;

```

```

void init_delayed_instance(void)
{
    static oid my_delayed_oid[] =
        { 1, 3, 6, 1, 4, 1, 8072, 2, 1, 2, 0 };
    netsnmp_handler_registration *my_test;

    my_test = netsnmp_create_handler_registration(
        "delayed_instance_example",
        delayed_instance_handler, my_delayed_oid,
        OID_LENGTH(my_delayed_oid), HANDLER_CAN_RWRITE);

    netsnmp_register_instance(my_test);
}

#define DELAYED_INSTANCE_SET_NAME "test_delayed"

int delayed_instance_handler(netsnmp_mib_handler *handler,
    netsnmp_handler_registration *reginfo,
    netsnmp_agent_request_info *reqinfo,
    netsnmp_request_info *requests)
{
    DEBUGMSGTL(("delayed_instance", "Got request, mode = %d:\n",
        reqinfo->mode));

    switch (reqinfo->mode) {
        default:
            requests->delegated = 1;

            snmp_alarm_register(delay_time, 0, return_delayed_response,
                (void *) netsnmp_create_delegated_cache(handler, reginfo,
                    reqinfo, requests, NULL));
            break;
    }
    return SNMP_ERR_NOERROR;
}

void return_delayed_response(unsigned int clientreg, void *
    clientarg)
{
    netsnmp_delegated_cache *cache = (netsnmp_delegated_cache *)
        clientarg;

    netsnmp_request_info *requests;
    netsnmp_agent_request_info *reqinfo;
    u_long *delay_time_cache = NULL;

    cache = netsnmp_handler_check_cache(cache);

```



```

if (!cache) {
    snmp_log(LOG_ERR, "illegal call to return delayed response\n");
    return;
}
reqinfo = cache->reqinfo;
requests = cache->requests;

DEBUGMSGTL(("delayed_instance",
    "continuing delayed request, mode = %d\n", cache->reqinfo->
    mode));

requests->delegated = 0;

switch (cache->reqinfo->mode) {
    case MODE_GET:
    case MODE_GETNEXT:
        snmp_set_var_typed_value(cache->requests->requestvb,
            ASN_INTEGER, (u_char *) & delay_time, sizeof(delay_time))
            ;
        break;
    case MODE_SET_RESERVE1:
        if (requests->requestvb->type != ASN_INTEGER) {
            netsnmp_set_request_error(reqinfo, requests,
                SNMP_ERR_WRONGTYPE);
            netsnmp_free_delegated_cache(cache);
            return;
        }
        break;
    case MODE_SET_RESERVE2:
        memdup((u_char **) & delay_time_cache, (u_char *) &
            delay_time, sizeof(delay_time));

        if (delay_time_cache == NULL) {
            netsnmp_set_request_error(reqinfo, requests,
                SNMP_ERR_RESOURCEUNAVAILABLE);
            netsnmp_free_delegated_cache(cache);
            return;
        }
        netsnmp_request_add_list_data(requests,
            netsnmp_create_data_list (DELAYED_INSTANCE_SET_NAME,
                delay_time_cache, free));
        break;
    case MODE_SET_ACTION:
        delay_time = *(requests->requestvb->val.integer);
        DEBUGMSGTL(("testhandler", "updated delay_time -> %d\n",
            delay_time));

```

```

        break;

case MODE_SET_UNDO:
    delay_time = *((u_long *) netsnmp_request_get_list_data(
        requests, DELAYED_INSTANCE_SET_NAME));
    break;

case MODE_SET_COMMIT:
case MODE_SET_FREE:
    break;
}
netsnmp_free_delegated_cache(cache);
}

```

Literatur

- [1] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction and Applicability Statements for Internet-Standard Management Framework. RFC 3410 (Informational), Dec. 2002.
- [2] J. Conner. Thold – Threshold Alert Module. <http://docs.cacti.net/plugin:thold>, 2011. [Online; accessed 16-Februar-2012].
- [3] D. Mauro and K. Schmidt. *Essential SNMP*. O'Reilly Media, 2001.
- [4] M. Schilli. Messdaten mit RRDtool und Perl verwalten. <http://www.linux-magazin.de/layout/set/print/content/view/full/4996>, 2004. [Online; accessed 16-Februar-2012].
- [5] The Cacti Group. Templates. <http://docs.cacti.net/templates>, 2011. [Online; accessed 16-Februar-2012].
- [6] T. Urban. *Cacti 0.8 Beginner's Guide*. PACKT Publishing, 2011.