Monitoring: Server- und Netzüberlastungen mit Bordmitteln ermitteln

Selbst ist der Admin

Fällt ein Server aus, sollten nicht die User, sondern Tools dem Admin das mitteilen. Besser wäre, wenn er schon die Frühformen abnormen Serververhaltens erkennt. Das vorgestellte Toolset überwacht Server und Dienste und fertigt Lastanalysen an. Anders als fertige Lösungen sind die gezeigten Skripte sehr flexibel. Charly Kühnast

Wer Daten über den Zustand seines Netzes und der Server sammelt und übersichtlich darstellt wird Engpässe und Überlastungen schnell und ohne manuelles Durchforsten von Logfiles diagnostizieren können. Für dieses Monitoring gibt es eine ganze Reihe guter und nützlicher Werkzeuge. Erst kürzlich hat das Linux-Magazin Nagios und Big Sister vorgestellt [1], [2].

Dieser Beitrag handelt aber davon, wie man diese Werkzeuge nicht benutzt, sondern stattdessen Bordmittel, die zu einem guten Teil jede Linux-Distribution mitbringt. Dazu kommen ein paar einfache, aber trickreiche Bash-Skripte. Das gewährleistet, dass jeder Admin das Vorgehen gut nachvollziehen kann. Somit – und hier liegt der Vorteil gegenüber den großen Netzmanagement-Tools – kann jeder den hier vorgestellten Werkzeugkasten an seine IT-Landschaft und deren Applikationen individuell anpassen. Weitere Vorteile:

- Auch exotische Auswertungen sind möglich.
- Es entstehen nicht nur die Ergebnisse, sondern auch die Rohdaten, die man verschiedenen auswerten kann.
- Hoher Lerneffekt für den Admin.
- Skripte haben für Linux/Unix-Enthusiasten ungleich mehr Charme als fertige Binaries.

Die Nachteile sollen aber auch nicht verschwiegen werden:

- Bis zur Einsatzreife vergeht mehr Zeit als bei einer fertigen Lösung.
- Weniger Features.
- Wenn sich die Toolsammlung bewährt, hat deren Programmierer für den Rest seines Daseins die Programmpflege am Bein.

Livecheck: Antworten die Server und ihre Dienste?

Das Wichtigste zuerst: Der Admin muss schauen, ob alle Server laufen und die Dienste, die auf ihnen laufen sollen, auch verfügbar sind. Dabei wird ihm das kleine Bash-Skript »simple_livecheck.sh« helfen – aber zuvor muss er noch einige Verzeichnisse und Dateien anlegen: Das Arbeitsverzeichnis ist »/usr/local/shellscripts/livecheck«. Darunter gibt es das Unterverzeichnis »etc«, in dem er mit dem Editor seiner Wahl für jeden Server eine Datei anlegt.

Ihr Name folgt zwingend dem Muster »*IP-Adresse_Name.SLD.TLD*«, im Falle des hier vorgestellten Beispiels lautet er »10.0.0.2_funghi.gondor.de«. In die Datei schreibt der Admin untereinander jene Ports, die im normalem Betriebszuständen aktiv sein sollen:

```
25
80
110
```

Für die anderen zu überwachenden Server legt man analog weitere Dateien an, im Beispiel »10.0.0.12_inn.gondor.de«, deren Inhalt nur aus der Portnummer 119 besteht.

Nmap leistet gute Dienste

Im Skript läuft eine Schleife über die Dateien, die per Ping checkt, ob die Server überhaupt antworten. Ist das der Fall – was jeder Admin schwer hofft –, überprüft das Skript die einzelnen Ports per Nmap [3], hier in der Version 3.00. Achtung, Suse Linux 9.0 hat einen nervigen Bug: Nmap funktioniert aus bislang ungeklärter Ursache nicht mit Root-Rechten. Listing 1 zeigt das Bash-Skript. (In Perl wären einige Details eleganter zu lösen – Snapshot-Autor Michael Schilli wird den Kopf oder gar den Autor dieses Beitrags schütteln.) Der Beispielserver fördert folgende Meldung zu Tage:

```
Server 10.0.0.2 (funghi.gondor.de): ping OK
10.0.0.2: Port 25 is up
10.0.0.2: Port 80 is up
10.0.0.2: Port 110 is up
```

Die Gegenprobe nach manuellem Stoppen des Apache befriedigt:

```
Server 10.0.0.2 (funghi.gondor.de): ping OK
10.0.0.2: Port 25 is up
10.0.0.2: Port 80 is down
10.0.0.2: Port 110 is up
```

So soll es sein! Aber selbst dem letzten Frischlings-Admin ist klar, dass ein Kon-

solen-Echo eine suboptimale Form der Alarmierung ist. Besser wäre es, erstens auf die Echos zu verzichten und stattdessen einen Eintrag ins Syslog zu schreiben, denn das Skript wird im wahren Leben nicht von Hand, sondern als Cronjob ausgeführt. Zweitens wäre eine Benachrichtigung per Mail, SMS oder Cityruf praktisch [4].

Viele Alarmierungswege führen zum Server

Die Wahl der Methode hängt davon ab, wie wichtig der jeweilige Server ist. Eine Stolperfalle: Der Alarm per Mail oder Mobilfunk sollte natürlich nur einmal rausgehen, nicht alle fünf Minuten, wenn der Cronjob anspringt. Selbst der coolste Handy-Klingelton verliert durch stete Wiederholung an Reiz. Das macht einige Modifikationen am Skript notwendig. Bei jedem Durchlauf muss es prüfen:

- Falls ein Host oder Port down ist: War er das auch im vorigen Durchlauf schon?
- Falls alles in Ordnung ist: War es das vorher auch schon oder ist ein vormals toter Host oder Port jetzt repariert und meldet sich zurück?

Als Vorbereitung für das verbesserte Skript »alarm_livecheck.sh« legt der Admin unter »/usr/local/shellscripts/livecheck« das Verzeichnis »deadhost« an. Findet das Skript einen toten Host, wird es dort einfach eine Datei mit dessen IP hineinlegen. Bei toten Diensten nennt das Skript die Datei »*IP_Port*«.

Am Vorhandensein dieser Datei (die nicht mal einen Inhalt haben muss) erkennt das Skript bei seinem nächsten Lauf, ob der Server beziehungsweise der Dienst gerade erst gestorben ist oder schon vorher down war, sich zwischenzeitlich zurückgemeldet hat oder sich dauerhaft bester Gesundheit erfreut. Das Skript ist in Listing 2 zu sehen und wartet darauf, ausprobiert und per Cron gestartet zu werden.

Das so gewonnene Wissen um die alltägliche Erreichbarkeit der Server und Dienste inklusive der jeweiligen Vorgeschichte verrät viel über die Stabilität der einzelnen Teile und lässt Schlussfolgerungen auf Schwachstellen zu, denen die Server- und Netzverantwortlichen mit Logfiles näher zu Leibe rücken. Natürlich gibt es bei den Skripten noch Raum für Verbesserungen und Erweiterungen, die der allgemeinen Betriebssicherheit und dem persönlichen Wohlbefinden dienlich wären.

Serverlast und Fehlalarme

Das nächste Interesse gilt der Information, wie stark die Server unter „Strom“ stehen. Als Werkzeuge bieten sich unter anderem MRTG [5], RRDTool und Cacti an. Hier fiel die Entscheidung für Letzteres in der Version 0.6.8 in Zusammenspiel mit RRDTool 1.0.40. Die Konfiguration von Cacti ist ausreichend flexibel und nicht allzu kompliziert; sie wird in [6] ausführlich beschrieben. Cacti liefert sowohl Netz- als auch Systemlast (Load Average) und damit die wichtigsten Parameter, die etwas über den Systemzustand aussagen. Mit diesen Graphen ausgestattet erkennt man anormales Systemverhalten wie zum Beispiel Lastspitzen auf einen Blick.

Dem Autor dieses Artikels hat Cacti erst kürzlich sehr praktische Dienste geleis-

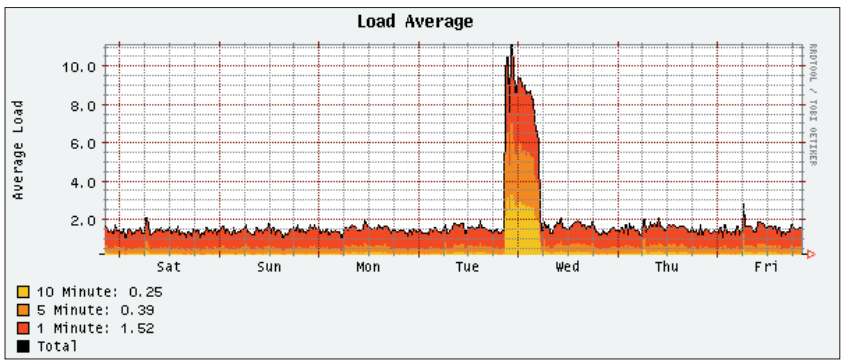

Abbildung 1: Die Lastspitze auf dem Webserver ist ein starkes Indiz dafür, dass das eingesetzte Backup-Tool mit den Systemressourcen aast. Daran sollte der Admin denken, wenn er Alarme setzt: Beispielsweise könnte hier ein »loadavg > 5.0« für „Alarm“ dafür sorgen, dass er bei jedem Backup aus dem Bett telefoniert würde.

tet, nachdem er sich mit einem eigentlich gut gemeinten Skript selbst reingelegt hatte: Auf seinem Webserver verrichtete das besagte Skript Cron-gesteuert brav seinen Dienst. Es las in kurzen Intervallen »/proc/loadavg«, um Alarm zu schlagen, falls die Last im Fünf-Minuten-Mittel höher als 8,0 war. Das tat es dann auch – mitten in der Nacht von letztem Dienstag auf Mittwoch. Der dazu passende Cacti-Graph in Abbildung 1 macht schnell klar, dass die Systemlast nicht langsam, sondern sehr plötzlich angestiegen war.

Der Graph, der die Last auf dem Netzwerk-Interface des Server zeigt, lag allerdings im Normalbereich – eine heftig angestiegene Zahl von HTTP- oder FTP-Zugriffen war als Ursache also von vornherein ausgeschlossen. Da auf dem Webserver außer den benötigten Serverdiensten nichts Nennenswertes läuft, blieb nur ein Kandidat als Verantwortlicher für die Lastspitze übrig: der Backup-Prozess. Inzwischen benutzt der Autor eine Backup-Lösung, die schonender mit den Systemressourcen umgeht und ihn durchschlafen lässt.

Listing 1: »simple_livecheck.sh«

```
01 #! /bin/bash
02
03 # Das Skript prüft, ob Server lebt und
04 # seine Dienste verfügbar sind
05
06 WDIR=/usr/local/shellscripts/lm-livecheck
07
08 for i in `ls $WDIR/etc/`; do
09
10   ## extract IP and fqdn from file name
11   IP=`echo $i|cut -f1 -d"_"`;
12   NAME=`echo $i|cut -f2 -d"_"`;
13
14   ## ping host to see if it's up
15   PING=$(/bin/ping -c2 -q -w2 $IP|grep
   transmitted|cut -f3 -d","|cut -f1 -d","|cut -f 1
   -d"%")
16   if [ $PING -eq " 0" ]; then
17     ## Host is up
18     echo "Server $IP ($NAME): ping OK";
19
20     ## now checking the ports
21     for j in `cat $WDIR/etc/$i`; do
22
23       RET=`/usr/bin/nmap -r --host_timeout 2500
   --initial_rtt_timeout 2000 -p $j $IP|grep
   $j/tcp|cut -f1 -d"/"`;
24
25       if [ -z $RET ]; then
26         echo "$IP: Port $j is down";
27         ## Alarm: Port down ##
28       else
29         echo "$IP: Port $j is up";
30       fi
31     done
32
33   else
34     echo "Server $IP ($NAME): no response";
35   fi
36 done
```

Nur zum Überblick: Netzlast als summierte Größe

Als eine der wichtigen Funktionen ermittelt Cacti die Gesamtlast auf einem oder mehreren Netzwerk-Interfaces – die zu kennen ist sicher gut. Aber besser wäre noch zu wissen, wie sich diese Last auf die einzelnen Dienste verteilt. Als Beispiel dient wieder der Testserver »funghi.gondor.de«: Auf ihm läuft ein Web-, Mail- und POP3-Server. Wenn Cacti jetzt anzeigt, dass die Last auf dem Netzwerk-Interface unnatürlich hoch ist, stellt sich sofort die Frage, welcher der drei Dienste der Schuldige ist.

Nun müsste man sich durchs System wursteln und anhand diverser Logfiles versuchen den verursachenden Dienst zu finden. Besser wäre es, etwas zu haben, das die Netzlast nach Ports sortiert

Listing 2: »alarm_livecheck.sh«

```
01 #! /bin/bash
02
03 # Das Skript prüft, ob Server lebt und
04 # seine Dienste verfügbar sind und alarmiert
05 # bei Störungen per Mail/SMS/Cityruf
06
07 WDIR=/usr/local/shellscripts/lm-livecheck
08
09 for i in `ls $WDIR/etc/`; do
10
11   ## extract IP and fqdn from file name
12   IP=`echo $i|cut -f1 -d"_"`;
13   NAME=`echo $i|cut -f2 -d"_"`;
14
15   ## ping host to see if it's up
16   PING=$(/bin/ping -c2 -q -w2 $IP|grep
   transmitted|cut -f3 -d","|cut -f1 -d","|cut -f 1
   -d"%")
17   if [ $PING -eq " 0" ]; then
18     ## Host is up
19     echo "Server $IP ($NAME): ping OK";
20
21     ## check if host was down and has now
   returned
22     if [ -e $WDIR/deadhost/$IP ]; then
23       echo "Server $IP ($NAME) came back to
   life";
24       rm $WDIR/deadhost/$IP;
25     fi
26
27     ## now checking the ports
28     for j in `cat $WDIR/etc/$i`; do
29
30       RET=`/usr/bin/nmap -r --host_timeout 2500 -
   -initial_rtt_timeout 2000 -p $j $IP|grep
   $j/tcp|cut -f1 -d"/"`;
31
32       if [ -z $RET ]; then
33         echo "$IP: Port $j is down";
34
35         ## check if Port was down before
36         if  [ -e $WDIR/deadports/$IP_$j ]; then
37           echo "Port $j on server $IP ($NAME) ist
   still dead";
38         else
39           echo "Port $j on server $IP ($NAME) has
   just died";
40           touch $WDIR/deadports/$IP_$j;
41           ## place commands for sending alarm
   here ##
42         fi
43       else
44         echo "$IP: Port $j is up";
45         ## check if port was down and has now
   been resurrected ##
46         if  [ -e $WDIR/deadports/$IP_$j ]; then
47           echo "Port $j on server $IP ($NAME)
   came back to life";
48           rm $WDIR/deadports/$IP_$j;
49         fi
50       fi
51     done
52
53   else
54     echo "Server $IP ($NAME): no response";
55
56     ## check if Server has been dead before
57     if [ -e $WDIR/deadhost/$IP ]; then
58       echo "Server $IP ($NAME) is still dead.";
59     else
60       echo "Server $IP ($NAME) has just died.";
61       touch $WDIR/deadhost/$IP;
62       ## place commands for sending alarm here ##
63     fi
64
65   fi
66 done
```

anzeigt. Zwar käme wieder Cacti in Frage, doch ein eigenes Tool verspricht mehr Flexibilität.

Portlast messen mit dem Multitalent Iptraf

Eine gute Lösung ist das Allround-Werkzeug Iptraf [7], denn es liefert umfangreiche Informationen über Netzwerk-Interfaces, darunter nicht nur die aktuelle Netzlast pro Interface, sondern auch Paketgrößen und eine Übersicht über den Traffic, und zwar sortiert nach angesprochenen Ports. Hier eingesetzt wurde Iptraf in der Version 2.7.0.

Die meisten Admins benutzen Iptraf hauptsächlich im interaktiven Modus, um einen schnellen Überblick über den aktuellen Status des Netzverkehrs auf einem Server zu bekommen. Glücklicherweise lässt sich Iptraf auch im Hintergrund als Daemon betreiben. In diesem Modus schreibt es seine Erkenntnisse in eine Logdatei (normalerweise »/var/log/iptraf«), die man auslesen und weiterverarbeiten kann. Zuerst wandert Iptraf also in die Cron-Datei:

```
*/5 * * * * /usr/sbin/iptraf -s eth0 -t 5 ➥
-B -L /var/log/iptraf
```

Der Parameter »-s« weist Iptraf an Informationen über den Traffic nach Ports sortiert zu sammeln. Dabei ist »-t 5« die Laufzeit in Minuten, bis Iptraf sich wieder beendet und seine Erkenntnisse ins Log schreibt, »-B« unterdrückt den Interaktivmodus und startet Iptraf als Daemon. Iptraf schreibt seine Log-Einträge in dieser Art:

```
TCP/25: 169107 packets, 90804448 bytes ➥
total; 96958 packets, 86978452 bytes ➥
incoming; 72149 packets, 3825996 bytes ➥
outgoing
TCP/110: 20174 packets, 7575496 bytes ➥
total; 8251 packets, 360974 bytes incoming;➥
 11923 packets, 7214522 bytes outgoing
```

Von gesteigertem Interesse sind die »bytes total«-Werte, um sie für später zu archivieren. Außerdem sollen sie in einer RRD (Round Robin Database) landen, die als Materiallager für grafische Verlaufsdiagramme dient. Fürs Archivieren legt der Admin das Unterverzeichnis »data« an, in dem pro Dienst und Tag eine Datei entsteht. Der Name des Dienstes und das Datum gehen aus dem Dateinamen hervor. Die Datei »smtp-history.20031115« beispielsweise enthält die von Iptraf ermittelten SMTP-Traffic-Daten vom 15. November 2003.

Messwerte-Datenbank für Cacti anlegen

Nun zur RRD: In dem dafür auserkorenen Arbeitsverzeichnis »/usr/local/shellscripts/iptraf« legt der Admin ein Unterverzeichnis »rrdtool« an. Es wird die Datenbank für den Beispielserver aufnehmen, die ein einziges, etwas längliches Kommando anlegt:

```
rrdtool create /usr/local/shellscripts/➥
iptraf/rrdtool/mailserver.rrd \
  DS:smtp:ABSOLUTE:600:U:U \
  DS:pop3:ABSOLUTE:600:U:U \
  RRA:AVERAGE:0.5:1:600 \
  RRA:AVERAGE:0.5:6:700 \
  RRA:AVERAGE:0.5:24:775 \
  RRA:AVERAGE:0.5:288:797 \
  RRA:MAX:0.5:1:600 \
  RRA:MAX:0.5:6:700 \
  RRA:MAX:0.5:24:775 \
  RRA:MAX:0.5:288:797
```

Ein Detail für Leute mit RRDTool-Kenntnissen: Anders als bei für per SNMP gewonnene Netzdaten ist hier die Datenquelle nicht mit »COUNTER«, sondern mit »ABSOLUTE« qualifiziert. Das be-

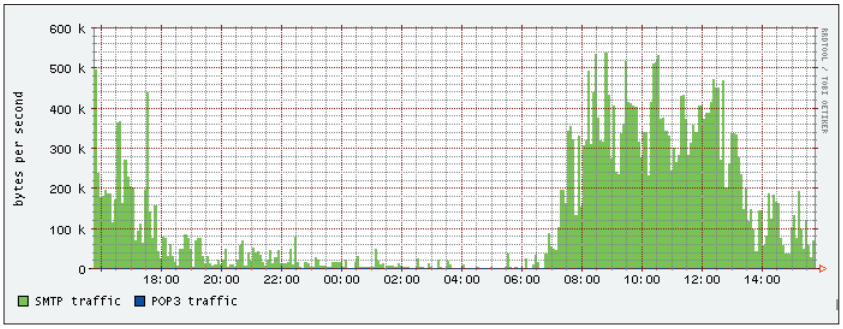

Abbildung 2: Der vom »plot_mailserver.sh«-Skript erzeugte Graph zeigt die Last auf Port 25 – eigentlich auch die von Port 110, wo aber mangels Aktivität kein Plot entsteht.

rücksichtigt auch den Umstand, dass Iptraf alle fünf Minuten von Neuem zu zählen anfängt.

Dem Schema des Beispiel-Mailservers folgend kann der Admin weitere RRDs für seine anderen Server anlegen, indem er lediglich den Dateinamen »mailserver.rrd« und die »DS«-Einträge abändert. Zwar braucht man das lange Kommando von eben nur einmal pro Server, aus praktischen Gründen macht sich das Ganze aber als Shellskript recht gut, weil ja gern mal neue Server angeschafft werden, die nach sofortiger Überwachung verlangen.

Das Archivieren und Plotten der Grafiken besorgt das kleine Skript »plot_mailserver.sh« aus Listing 3, Abbildung 2 zeigt das Ergebnis. Für andere überwachungswerte Dienste wie HTTP, FTP oder NNTP sind eigene Skripte analog schnell geschrieben. Damit und mit den oben beschriebenen Werkzeugen ausgerüstet besitzt der Admin eine solide Datenbasis, um bei auftretenden Verstopfungen im Netz den Verursacher schnell zu finden. Beim Autor jedenfalls ließ der erste Praxistest für das Toolset nicht lange auf sich warten.

Der Tod kam leise

Der Fall: In unregelmäßigen Abständen starb ein Mailserver ab. Der Ablauf war immer gleich: Zuerst schlug »alarm_livecheck.sh« Alarm, dass der SMTP-Port nicht mehr reagiere, kurz darauf der POP3-Port und etwas später antwortete der Server nur noch sporadisch auf Ping, dann gar nicht mehr. Der mit Cacti erhobene Netzlast-Graph zeigte etwa 30 Minuten vor dem Exitus zwar steigende Aktivität auf dem Interface, aber nicht in einem Maße, dass es den treuen Postfix hätte beeindrucken dürfen.

Der Loadavg-Graph dagegen wusste zu beeindrucken: Ein kontinuierlicher Lastanstieg der Last bis über 40, dann nichts mehr. Diese Symptome sind typisch für Maschinen, deren Speicher so voll läuft, dass sie sich ins Nirwana swappen. Tatsächlich besaß der betagte Mailserver nur 64 MByte RAM und 128 MByte Swapspace. Von der anderen Seite betrachtet ist Postfix aber nicht durch ausufernden Ressourcenverbrauch bekannt. Somit geriet ein anderer Kandidat ins Visier, der kürzlich aufgespielte Spamassassin [8].

Den Beweis für die These lieferte ein Test, der bei laufendem Top und von einer zweiten Maschine rund hundert Mails in den Mailserver einliefern ließ. Bingo: Nach kurzer Zeit trieb Spamassassin den Mailserver über die Swap-Grenze. Nur, was tun? Zwei Lösungen lagen nahe:

- Man bremst Postfix künstlich aus, um die Mailfrequenz zu verringern. Dann können sich die Spamassassin-Prozesse nach getaner Arbeit in Ruhe beenden und ihren Speicher freigeben. Das wäre in Postfix' »main.cf« schnell konfiguriert, ist aber unelegant.
- Der Server bekommt RAM spendiert.

Listing 3: »plot_mailserver.sh«

```
01 #! /bin/bash
02
03 sleep 5 # give iptraf time to write its data
   into the log file
04
05 TRAFLOG=/var/log/iptraf
06 WDIR=/usr/local/shellscripts/iptraf
07 TODAY=$(/bin/date +%s)
08 UDATE=$(/bin/date +%Y%m%d)
09
10 SMTP=$(grep "TCP/25" $TRAFLOG|tail -n1|cut -f2
   -d","|cut -f2 -d" ")
11 POP=$(grep "TCP/110" $TRAFLOG|tail -n1|cut -f2
   -d","|cut -f2 -d" ")
12
13 echo "smtp: $SMTP"
14 echo "pop3: $POP"
15
16 if [ -z $SMTP ]; then
17   SMTP="0";
18 fi
19
20 if [ -z $POP ]; then
21   POP="0";
22 fi
23
24 # archive results
25
26 echo $SMTP >> $WDIR/data/smtp-history.$UDATE
27 echo $POP >> $WDIR/data/pop-history.$UDATE
28
29 rrdtool update $WDIR/rrdtool/mailserver.rrd
   $TODAY:$SMTP:$POP3
30
31 # draw the graph
32
33 rrdtool graph /usr/local/httpd/htdocs/
   protostats/mailserver.gif \
34 --start -86400 \
35 --vertical-label "bytes per second" \
36 -w 600 -h 200 \
37 DEF:smtp=$WDIR/rrdtool/mailserver.rrd
   :smtp:AVERAGE \
38 DEF:pop3=$WDIR/rrdtool/mailserver.rrd
   :pop3:AVERAGE \
39 AREA:smtp#00ff00:"SMTP traffic" \
40 LINE1:pop3#0000ff:"POP3 traffic"
```

Listing 4: »meminfo.sh«

```
01 #! /bin/bash
02
03 # Das Skript ermittelt den freien Speicher
   (RAM und Swap)
04 # und lässt diese Werte von RRDTool plotten.
05
06 WDIR=/usr/local/shellscripts/iptraf
07 TODAY=$(/bin/date +%s)
08
09 ## extract mem values from /proc/meminfo
10
11 RAM=`grep MemFree /proc/meminfo|tr -s
   [:blank:]|cut -f2 -d" "`
12 SWAP=`grep SwapFree /proc/meminfo|tr -s
   [:blank:]|cut -f2 -d" "`
13
14 ## write data into the RRD
15
16 rrdtool update $WDIR/rrdtool/mailmemory.rrd
   $TODAY:$RAM:$SWAP
17
18 ## draw the graph
19
20 rrdtool graph /usr/local/httpd/htdocs/
   protostats/mailmemory.gif \
21 --start -86400 \
22 --vertical-label "kBytes free" \
23 -w 600 -h 200 \
24 DEF:ram=$WDIR/rrdtool/mailmemory.rrd:ram:
   AVERAGE \
25 DEF:swap=$WDIR/rrdtool/mailmemory.rrd:swap:
   AVERAGE \
26 AREA:ram#00ff00:"RAM" \
27 LINE1:swap#0000ff:"Swap"
```

Die Entscheidung fiel zugunsten der zweiten Lösung. Mit 512 MByte RAM und 1 GByte Swap waren die Probleme vom Tisch. Das Beispiel lehrt: Wer auch den Speicherverbrauch in die Überwachung einbezieht, tut sich bei Diagnosen leichter. Das soll jetzt geschehen.

Speicherverbrauch protokollieren

In bewährter Do-it-yourself-Manier wird ein kleines Skript den Füllpegel des RAM und des Swap aus »/proc/meminfo« lesen, in eine RRD schreiben und daraus einen Graphen plotten. Aber zunächst erzeugt folgender Befehl die RRD:

```
rrdtool create /usr/local/shellscripts/➥
iptraf/rrdtool/mailmemory.rrd \
  DS:ram:GAUGE:600:U:U \
  DS:swap:GAUGE:600:U:U \
  RRA:AVERAGE:0.5:1:600 \
  RRA:AVERAGE:0.5:6:700 \
  RRA:AVERAGE:0.5:24:775 \
  RRA:AVERAGE:0.5:288:797 \
  RRA:MAX:0.5:1:600 \
  RRA:MAX:0.5:6:700 \
  RRA:MAX:0.5:24:775 \
  RRA:MAX:0.5:288:797
```

Die Datenbank residiert der Ordnung halber neben den Netzlastgraphen in dem eben schon benutzten »rrdtool«-Verzeichnis, obgleich diesmal nicht Iptraf die Daten ermittelt. Achtung: Die Datenquelle ist hier als »GAUGE« definiert, weil sie im Gegensatz zu den Iptraf-Daten nicht nach jedem Lesevorgang auf null zurückfällt.

Nachdem die Datenbank angelegt ist, beginnt das Skript »meminfo.sh« aus Listing 4 mit dem Sammeln der Daten und erzeugt einen Graphen. Damit hat der Admin-Werkzeugkasten einen durchaus zufrieden stellenden Füllgrad erreicht.

Der Blick in die Glaskugel

Die Archiv-Funktion aus dem Iptraf-Skript existiert natürlich nicht als Selbstzweck. Vielmehr möchte sie der Autor später heranziehen, um mittelfristige Prognosen über das Netzlast-Aufkommen zu errechnen. Dafür gibt es eine Vielzahl mathematischer Modelle [9]. Bereits mit bloßem Auge ist am RRDTool-Graphen erkennbar, dass sich die Netzlastdaten über einen langen Zeitraum ungefähr linear steigend entwickeln.

Das macht Vorhersagen leicht: Zuerst bestimmt man die Trendgerade, eine gedachte Gerade, die so durch den Graphen führt, dass die einzelnen Messpunkt ihr in y-Richtung möglichst nahe sind. Dann berechnet man die Steigung dieser Gerade und nimmt nassforsch an, dass die künftig ermittelten Messwerte ebenfalls möglichst nahe an dieser Gerade liegen. Diese Methode taugt für kurzfristige Prognosen unter der Prämisse, dass sich die Nebenbedingungen nicht nennenswert ändern und – ganz wichtig – dass die Systeme nicht an technische Limits stoßen.

Übrigens: Theoretisch kann die Netzlast auch sinken – theoretisch –, dann weist die Trendgerade abwärts. Mit Ausnahme einiger Fälle von Dotcom-Firmen in der Spätphase mit stark degressiver PC-Nutzerzahl sind aus der Praxis keine Fälle von Netzlast- oder Bandbreiten-Rückgängen glaubhaft überliefert. *(jk)* ■

Infos

[1] D. Ruzicka, „ Netzmanagement mit Nagios, dem Nachfolger von Netsaint“: Linux-Magazin 3/03, S. 66

[2] J. Fritsch, Th. Aeby, „ Network Monitoring mit Big Sister“: Linux-Magazin 12/03, S. 54

[3] Nmap: [http://www.insecure.org/nmap]

[4] Ch. Kühnast, „ Aus dem Alltag eines Sysadmin – Yaps“: Linux-Magazin 4/03, S. 53

[5] W. Boeddinghaus, „ Netzüberwachung mit MRTG“: Linux-Magazin 9/02, S. 49

[6] A. Schrepfer, „ Systemdaten grafisch überwachen mit Cacti, dem Webfrontend für RRDtool“: Linux-Magazin 9/03, S. 54

[7] Iptraf: [http://iptraf.seul.org]

[8] Spamassassin: [http://eu3.spamassassin.org]

[9] Prof. Dr. J. Kopf, Arbeitspapiere zur Zeitreihen-Analyse: [http://www.wifak.uni-wuerzburg.de/ewf/doku/zra/ap-zra.htm]

Der Autor

Charly Kühnast administriert Unix-Betriebssysteme im Rechenzentrum Niederrhein in Moers. Zu seinen Aufgaben gehören die Sicherheit und Verfügbarkeit der Firewalls und der DMZ (demilitarisierte Zone). In seiner Freizeit lernt er Japanisch, um endlich die Bedienungsanleitung seiner Mikrowelle lesen zu können.