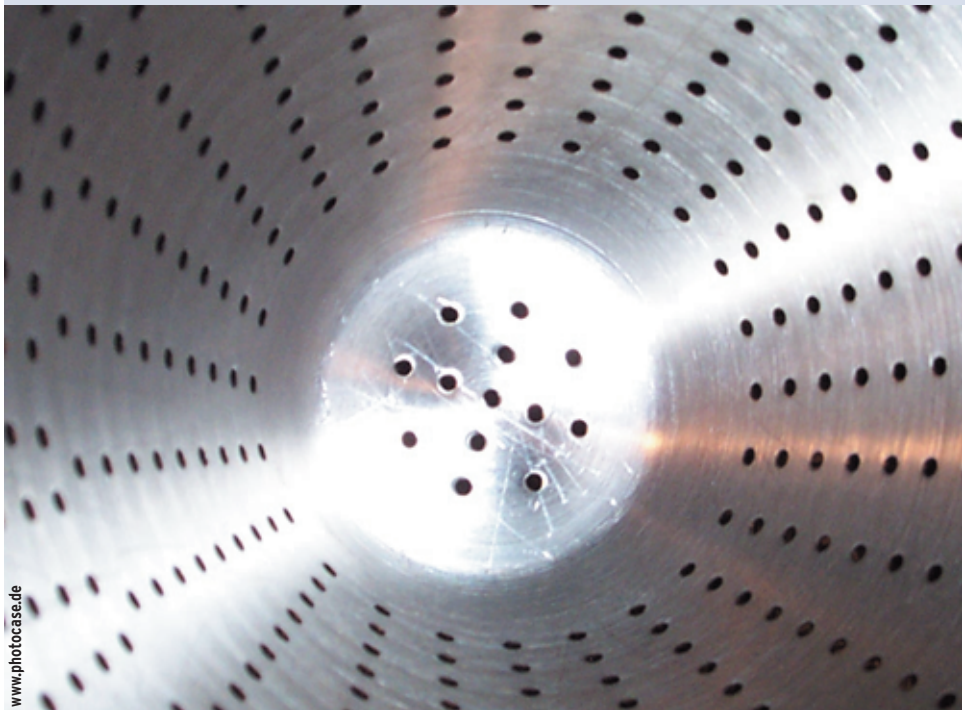
Messdaten mit RRDtool und Perl verwalten

Daten ausgesiebt

Ständig reinrieselnde Messdaten müssen die Festplatte nicht vollstopfen. Bei älteren Werten genügt ein Überblick, während fürs aktuelle Geschehen alle Details wichtig sind. Eine Round-Robin-Datenbank wie RRDtool vergisst unwichtige Werte, das System bleibt bis zum Sankt-Nimmerleins-Tag wartungsarm. Michael Schilli


www.photocase.de

Das von Tobias Oetiker entwickelte Programm RRDtool [2] hat sich zum Quasi-Standard bei der Speicherung von Netzwerk-Überwachungsdaten gemausert. Es legt seine Daten in so genannten Round-Robin-Datenbanken (RRD) ab, die von Frontends wie Cacti [3] genutzt werden. Ein Round-Robin-Archiv (RRA) stellt man sich am besten wie in Abbildung 1 dargestellt vor.

Dort liegen auf einer begrenzten Anzahl von Speicherplätzen die bei einem Webserver ermittelten Lastwerte, angefangen beim Wert 6,1 um 01:00 Uhr (oben Mitte), dann – im Uhrzeigersinn – eine Last von 2,0 um 01:01 Uhr, bis schließlich zu dem um 01:04 Uhr gespeicherten Wert 2,4. Der Zeiger deutet auf den zuletzt aktualisierten Eintrag.

Das Ergebnis der nächsten Messung passt aber nicht mehr ins Archiv – deshalb wird, wie Abbildung 2 zeigt, die Messung von 01:00 Uhr mit dem neuen Wert 4,1 von 01:05 Uhr überschrieben. Nun ist der Admin aber nicht nur an Messwerten der letzten 5 Minuten interessiert, sondern möchte auch sehen, wie sich die Rechnerlast über die letzten dreißig Tage oder die zurückliegenden zwölf Monate entwickelt hat.

Gewollt unscharf

Auch dafür braucht er keine riesigen Datenmengen vorzuhalten, denn über größere Zeiträume hinweg ist eine gewisse Unschärfe akzeptabel. Der Trick ist, weitere RRAs anzulegen, die die Durchschnittslast (oder die Höchstlast, ganz nach Geschmack) pro Stunde für den letzten Tag oder pro Tag für das laufende Jahr aufnehmen. Sind diese Round-Robin-Archive erst einmal in der RRD-Datei angelegt, füttert RRDtool neue Messwerte hinein, und zwar per Kommandozeilenaufruf oder über die mitgelieferte Perl-Schnittstelle.

Der darunter liegende Datenbankmotor sorgt automatisch dafür, dass die Kreise mit den verschiedenen Granularitäten die richtig aufpolierten Daten erhalten. Spätere Abfragen liefern auch die Werte über einen angegebenen Zeitraum in der höchsten verfügbaren Genauigkeit und RRDtool zeichnet davon sogar formschöne Grafiken.

RRDtool in Perl-Skripten verwenden

Die Definition einer Round-Robin-Datenbank besteht aus einer oder mehreren Datenquellen (DS, Data Sources). Für jede einzelne Quelle gibt der RRD-Administrator beim Anlegen der Datenbank vier Parameter an: Einen Namen, einen Datenquellentyp, die Breite des Eingabe-Zeitfensters sowie die minimalen und maximalen Eingangswerte.

Der Name (zum Beispiel »load« oder »mem_usage«) identifiziert die Eingabedatenquelle in der RRD eindeutig. Über den Datenquellentyp (DST, Data Source Type) legt der Admin fest, ob die Eingabewerte einfach übernommen werden (»GAUGE«) oder von einem stetig wachsenden Zähler stammen, den RRDtool bei einem Überlauf sinnvoll behandelt (»COUNTER«). Das Programm fängt den Überlauf dann ab und nutzt den zuletzt gespeicherten Wert, um die Zählreihe fortzuführen.

Purzeln in der durch die Breite des Eingabe-Zeitfensters bestimmten Zeit mehrere Daten herein, wird der Mittelwert

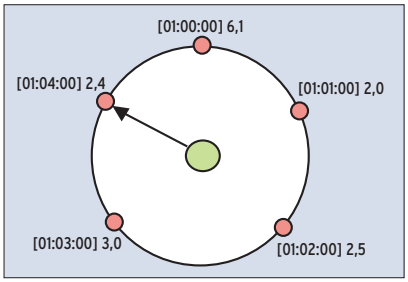


Abbildung 1: Das Round-Robin-Archiv hält eine feste Anzahl von Datenwerten vorrätig und überschreibt alte Werte, um Platz für neue zu schaffen.

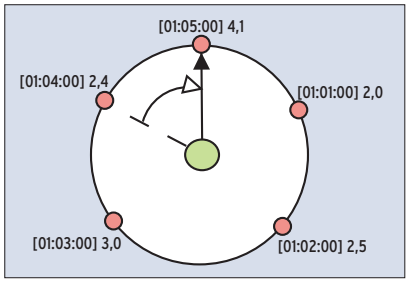


Abbildung 2: Der alte, um 01:00 Uhr gemessene Wert wird durch den um 01:05 Uhr ermittelten ersetzt, der Zeiger rückt weiter.

berechnet und abgespeichert. Trifft in diesem Zeitraum kein einziger Wert ein, speichert RRDtool ein »na« (not available). Alle Messwerte, die jenseits der minimalen und maximalen Eingangswerte liegen, ignoriert es.
Folgender Perl-Code legt eine Datenbank an, die ihre Einträge aus der Eingabequelle namens »load« bezieht, die wiederum alle 60 Sekunden Infos über die aktuelle Rechnerlast liefert:

```
use RRDs;

RRDs::create(
  "/tmp/load.rrd", "--step=60",
  "--start=" . time() - 10,
  "DS:load:GAUGE:90:0:10.0",
  "RRA:MAX:0.5:1:5",
  "RRA:MAX:0.5:5:10");
```

Leider gibt es noch kein intuitives objektorientiertes Interface für RRDtool, der etwas kryptisch anmutende Code ist daher sehr erklärungsbedürftig. In der Datei »/tmp/load.rrd« legt RRDtool die Datenbank ab. Das vorgegebene Eingabeintervall ist 60 Sekunden (über die Option »--step = 60«). In diesen Zeitabständen werden später die Daten in das Archiv eingespeist.

Mein Herz macht bum

Die Startzeit der Datenbank ist auf zehn Sekunden in der Vergangenheit gelegt. Das ist üblich und voreingestellt, wenn man »--start« weglässt, denn RRDtool wird alle Eingaben zurückweisen, die einen Zeitstempel kleiner oder gleich der Startzeit tragen. Die »DS:«-Zeile definiert die einzige Datenquelle der Datenbank mit den oben beschriebenen Parametern: Quellenname »load«, Eingabetyp »GAUGE«, dem so genannten Heartbeat von 90 sowie Minimal- und Maximalwert 0 beziehungsweise 10,0. Die mit 90 angegebene Pulsfrequenz legt fest, dass der Admin auch zufrieden ist, falls die Daten nicht mit der in »--step« angegebenen Rate von 60 Sekunden ankommen, sondern mit bis zu 30 Sekunden Verzögerung. RRDtool lügt dann und interpoliert einfach. Wäre im Extremfall der Heartbeat 24 Stunden und die Schrittrate weiterhin 60, genügte ein einziger Wert pro Tag, auf den RRDtool dann alle Minuteneinträge setzen würde.

Primary Data Points

Ist der Heartbeat auf einen niedrigeren Wert als der Step gesetzt, sind mehrere Messdaten pro Step erforderlich. Dann erwartet RRDtool die Daten im Rhythmus des Herzens und speichert sofort streng »na« für einen Schritt, falls der Herzschlag einmal aussetzt. Liegen ordnungsgemäß mehrere Werte pro Schrittfenster vor, errechnet RRDtool den Mittelwert, bevor es den so genannten Primary Data Point (PDP) speichert.
Im links abgedruckten Code erzeugen die letzten beiden Zeilen die RRAs. Der Zahlenwert in den vorletzten Spalten gibt an, wie viele PDPs das Archiv zu einem Archivpunkt zusammenfassen soll. Das erste Archiv übernimmt nur einen Wert, es entspricht den Round-Robin-Archiven, wie in den Abbildungen 1 und 2 gezeigt.
Das zweite Archiv fasst fünf Messpunkte zu einem Archivpunkt zusammen. Bei einem einzigen PDP muss RRDtool nichts weiter tun, aber bei fünf ist die zweite Spalte der obigen Definition wichtig, in der die Consolidation Function (CF) angegeben ist. »AVERAGE« schnappt sich den Mittelwert aus den PDPs, die oben verwendete Funktion »MAX« nimmt den Höchstwert. Weitere Optionen sind »MIN« für den kleinsten und »LAST« für den zuletzt ermittelten Wert.
Die magische Zahl »0.5« ist der so genannte Xfiles Factor. Er bestimmt, welcher Bruchteil von den PDPs undefiniert (»na«) sein darf, damit das Archiv einen interpolierten Mittelwert als gültigen Eintrag speichert. Wird der Wert unterschritten, steht später »na« im Archiv. Die letzte Spalte bestimmt, wie viele Datenplätze das Archiv bereitstellt. Sind alle aufgefüllt, beginnt es, die ältesten zu überschreiben. Abbildung 3 zeigt, wie RRDtool aus den Werten, die die Datenquelle liefert, PDPs erzeugt, die anschließend in die verschiedenen Round-Robin-Archive wandern.

RRD auf die Finger geschaut

Listing 1 ist ein Testskript, das eine RRD definiert, dann künstlich erzeugte Werte eingibt und die Archivdaten abfragt. Für reproduzierbare Ergebnisse nutzt das Skript statt der Systemzeit den Zeitstempel »1080460200«. Tipp: RRD fängt wild zu runden an, falls keine durch 60 (und für das Fünf-Minuten-Archiv sogar durch 300) teilbare Zahl verwendet wird. Das mittelt sich zwar auf Dauer, aber für Demonstrationszwecke eignen

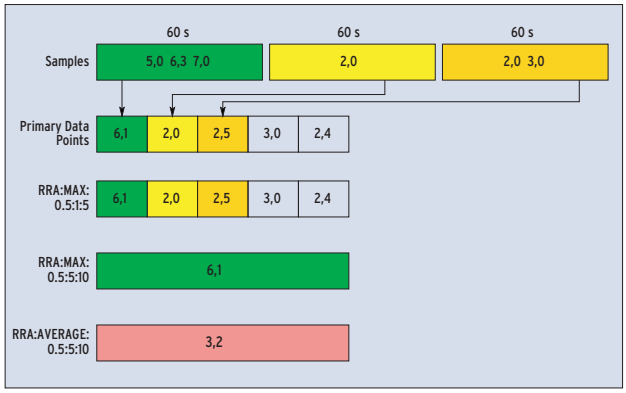


Abbildung 3: Aus den Sample-Werten, die aus einer Datenquelle kommen, erzeugt RRDtool Primary Data Points (PDPs). Mit diesen PDPs füllt RRDtool seine Round-Robin-Archive (RRA).

sich glatte Werte besser. Die Zeilen 17 bis 22 erzeugen die RRD wie oben beschrieben.
Die »for«-Schleife ab Zeile 26 läuft von 0 bis 40 und schiebt mit »RRDs::update()« die folgenden Zeitstempel-Lastwert-Kombinationen als Strings in die RRD:

```
1080460200:2
1080460260:2.1
1080460320:2.2
...
```

Im Normalbetrieb ist es möglich, den Zeitstempel nicht zu übergeben, dann nimmt das »RRDs«-Modul die aktuelle Systemzeit. Bei den übergebenen Werten handelt es sich um künstlich erzeugte Beispielwerte für die Systemlast, das Testskript startet einfach bei »2« und erhöht den Wert pro Schritt um »0.1«.

Auswertung starten

Um ein Archiv abzufragen, nimmt »RRDs::fetch()« das gewünschte Abfrageintervall mit der beim Abspeichern verwendeten CF entgegen und ermittelt daraus das Archiv mit der maximal verfügbaren Auflösung. Ist eine CF angegeben, für die kein Archiv existiert, gibt das Modul eine Fehlermeldung zurück. »RRDs::fetch()« speichert in Zeile 47 die Datenpunkte des Archivs in »$data«, einer Referenz auf ein Array, das wiederum Referenzen enthält, die auf Arrays mit den Floating-Point-Datenwerten zeigen. Weitere von »RRDs::fetch()« zurückgelieferte Werte sind »$dbstart« (Startzeitpunkt dieser RRD), »$step« (Zeitabstand der Datenpunkte im Archiv) und »$names« (Referenz auf ein Array mit den Namen aller Datenquellen).

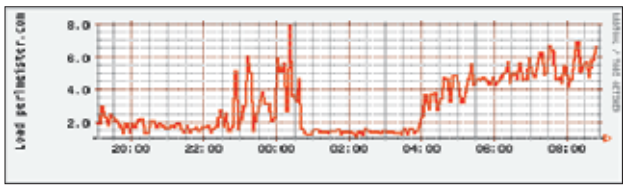


Abbildung 4: Diese mit RRDtool erstelle Grafik zeigt die Last auf dem Internet-Host perlmeister.com, und zwar über eine ganze Nacht verteilt.

»$step« ist übrigens nicht unbedingt der mit »--step« eingestellte Datensammelabstand der Datenbank. Für ein Archiv, das mehrere PDPs zu einem Archivpunkt zusammenfasst, ergibt sich »$step« aus der Multiplikation von Sammelabstand und der Anzahl der pro Archivpunkt erfassten Punkte.
Zeile 36 von Listing 1 startet mit Hilfe der in Zeile 42 definierten Funktion »fetch()« eine Abfrage im Zeitfenster der letzten fünf Minuten vor dem Ende. Sie fördert Folgendes zutage:

```
Last 5 minutes:
1080462300: N/A
1080462360: 5.6
1080462420: 5.7
1080462480: 5.8
1080462540: 5.9
1080462600: 6
```

Das Modul »RRDs« hat hierzu das Kurzzeitarchiv mit 60 Sekunden Datenabstand gewählt. Da es immer nur fünf Werte vorhält, ist der älteste Wert »na«. Fragt der Anwender, wie etwa in Zeile 39, die Funktion »RRDs::fetch()« hingegen nach Werten für ein breiteres Fenster, beispielsweise für die letzten 30 Minuten der Messreihe, enthält das Ergebnis Werte aus dem zweiten Archiv, das die Daten im 300-Sekunden-Abstand speichert:

```
Last 30 minutes:
1080460800: 3
1080461100: 3.5
1080461400: 4
1080461700: 4.5
1080462000: 5
1080462300: 5.5
1080462600: 6
```

Bei den eingetragenen Werten handelt es sich um die im jeweiligen Intervall gemessenen Höchstwerte, da das zweite Archiv mit der CF »MAX« definiert ist. Das Modul »RRDs« wird übrigens nicht versuchen, Werte aus einer Kombination von Archiven darzustellen, es wählt ein passendes Archiv aus und nutzt dessen Granularität für eine Ergebnisreihe mit konstanten Zeitabständen.
Das Skript »rrdload« in Listing 2 zeigt eine praktische Anwendung von RRD-

Listing 1: »rrdtest«

```
01 #!/usr/bin/perl
02 ##############################################
03 # Feed test data to RRD
04 # Mike Schilli, 2004 (m@perlmeister.com)
05 ##############################################
06 use warnings;
07 use strict;
08
09 use RRDs;
10
11 my $DB    = "/tmp/mydemo.rrd";
12 my $start = 1080460200;
13 my $dst   = "MAX";
14 my $nof_iterations = 40;
15 my $end   = $start + $nof_iterations * 60;
16
17 RRDs::create(
18     $DB, "--step=60",
19     "--start=" . ($start-10),
20     "DS:load:GAUGE:90:0:10.0",
21     "RRA:$dst:0.5:1:5",
22     "RRA:$dst:0.5:5:10",
23 ) or
24     die "Cannot create rrd ($RRDs::error)";
25
26 for(0..$nof_iterations) {
27     my $time = $start + $_ * 60;
28     my $value = 2 + $_ * 0.1;
29
30     RRDs::update(
31         $DB, "$time:$value") or
32          die "Cannot update rrd ($!)";
33 }
34
35 print "Last 5 minutes:\n";
36 fetch($end - 5*60, $end, $dst);
37
38 print "Last 30 minutes:\n";
39 fetch($end - 30*60, $end, $dst);
40
41 ##############################################
42 sub fetch {
43 ##############################################
44     my($start, $end, $dst) = @_;
45
46     my ($dbstart, $step, $names, $data) =
47       RRDs::fetch($DB, "--start=$start",
48         "--end=$end", $dst);
49
50     foreach my $line (@$data) {
51         print "$start: ";
52         $start += $step;
53         foreach my $val (@$line) {
54           $val = "N/A" unless defined $val;
55           print "$val\n";
56         }
57     }
58 }
```

tool. Folgende Cronjob-Konfiguration startet es alle fünf Minuten:

```
*/5 * * * * /home/mschilli/bin/rrdload -u
```

Mit der Option »-u« aufgerufen frischt es ein Round-Robin-Archiv mit dem Messwert der aktuellen Systemlast auf und verabschiedet sich dann wieder. Zur grafischen Auswertung ruft der Administrator es mit der Option »-g« auf. Es legt dann eine schöne Grafik wie in Abbildung 4 als PNG-Datei im Dokumentenpfad des Webservers ab. Auf diese Weise kontrolliert der Autor beispielsweise auf dem Shared-System Perlmeister.com die Systemlast.

Graphen zeichnen

Der Code ab Zeile 21 in Listing 2 legt drei Archive an. Das erste nimmt 288 Datenpunkte auf, stellt also genügend Plätze bereit, um die alle fünf Minuten ermittelten Werte einen Tag lang zu speichern (24·12). Das zweite Archiv sucht die Spitze aus zwölf Messpunkten, also eine Stunde (12·5 Minuten = 60 Minuten) lang eingehende Daten, und speichert 168 davon. Später steht die jeweils letzte Woche im Stundentakt zur Abfrage bereit (168 = 24·7). Das dritte und letzte Archiv findet noch die Tagesspitzen und hält 365 davon für die Jahresbilanz vorrätig.

Die erzeugte Grafik im PNG-Format erzeugt das Skript mit Hilfe der Funktion »RRDs::graph()«. Sie erhält über »--vertical-label« noch eine Beschriftung für die Lastachse. Die beiden Argumente

```
"DEF:myload=$DB:load:MAX",
"LINE2:myload#FF0000"
```

bestimmen, dass das »RRDs«-Modul aus der in »$DB« angegebenen Datei Ergebnisdaten bezieht und diese der Graphen-Variablen »myload« zuordnet. Es werden Werte der Datenquelle »load« gesucht, in einem Archiv, das zum vorher angegebenen Zeitraum (»--start« bis »--end«) Daten mit der Consolidation Function »MAX« gewonnen hat.

RRDtool hat die zweifelhafte Angewohnheit, die Datenbank zu Beginn zufällig zu füllen und über die Startzeit eine falsche Auskunft zu geben – daher greift die ab Zeile 46 definierte Funktion »rrd_start_time()« ein und holt so lange Daten heraus, bis etwas Vernünftiges erscheint. Das Datum dieses Messwerts gibt sie zurück und die »graph«-Funktion nimmt es entgegen.

Die in Zeile 50 verwendete Funktion »RRDs::fetch()« geht ohne Angabe einer Startzeit genau einen Tag zurück. Wer einen längeren Zeitraum im Graphen betrachten möchte, bestimmt mit »--start« einen anderen Zeitpunkt. Negative Werte setzen relative Zeitdifferenzen zur gegenwärtigen Uhrzeit. Mit »"--start", -365*24*3600« kommen stets alle verfügbaren Daten zur Anzeige, allerdings in der gröbsten Auflösung. Den Graphen malt die Funktion elegant ganz in Rot (»#FF0000«) und wegen »LINE2« genau zwei Pixel stark.

Das Perl-Modul »RRDs«, das eine Shared Library von »RRDtool« nutzt, gibt es nicht im CPAN, es liegt der RRD-Distribution bei. Um es zu installieren, lädt und entpackt der Admin den neuesten Source-Tarball von [2] und kompiliert ihn per »./configure; make«. Im Unterverzeichnis »perl-shared« findet sich die Distribution von »RRDs.pm«, die Installation erfolgt mit »perl Makefile.PL; make install«. *(mwe)* ■

Infos

[1] Listings zu diesem Artikel: [ftp://ftp.linux-magazin.de/pub/listings/magazin/2004/06/Perl]
[2] RRDtool: [http://www.rrdtool.com]
[3] Achim Schrepfer, „Kurven-Schau“: Linux-Magazin 09/03, S. 54
[4] Charly Kühnast, „Netz-Monitoring“: Linux-Magazin 01/04, S. 28

Der Autor

Michael Schilli arbeitet als Software-Engineer für AOL/Netscape in Mountain View, Kalifornien. Er hat „Goto Perl 5“ (deutsch) und „Perl Power“ (englisch) für Addison-Wesley geschrieben und ist unter [mschilli@perlmeister.com] zu erreichen. Seine Homepage ist [http://perlmeister.com].

Listing 2: »rrdload«

```
01 #!/usr/bin/perl
02 ##########################################
03 # rrdload -- Measure CPU load over time
04 # Mike Schilli, 2004 (m@perlmeister.com)
05 ##########################################
06 use warnings;
07 use strict;
08
09 use RRDs;
10 use Getopt::Std;
11
12 getopts("ug", \my %opts);
13
14 my $DB     = "/tmp/load.rrd";
15 my $SERVER = "/www/htdocs";
16 my $UPTIME = "uptime";
17
18 if(! -f $DB) {
19   RRDs::create($DB, "--step=300",
20     "DS:load:GAUGE:330:U:U",
21     "RRA:MAX:0.5:1:288",
22     "RRA:MAX:0.5:12:168",
23     "RRA:MAX:0.5:288:365",
24   ) or die "Create error: ($RRDs::error)";
25 }
26
27 if(exists $opts{u}) {
28     my $uptime = `$UPTIME`;
29     my ($load) = ($uptime =~ /(\d\.\d+)/);
30
31     RRDs::update($DB, time() . ":$load") or
32         die "Update error: ($RRDs::error)";
33 }
34
35 if(exists $opts{g}) {
36   RRDs::graph("$SERVER/load.png",
37     "--vertical-label=Load perlmeister.com",
38     "--start=" . rrd_start_time(),
39     "--end=" . time(),
40     "DEF:myload=$DB:load:MAX",
41     "LINE2:myload#FF0000") or
42         die "graph failed ($RRDs::error)";
43 }
44
45 ##########################################
46 sub rrd_start_time {
47 ##########################################
48
49     my ($start,$step,$names,$data) =
50                     RRDs::fetch($DB, "MAX");
51
52     foreach my $line (@$data) {
53       if(! defined $line->[0]) {
54           $start += $step;
55           next;
56       }
57       return $start;
58     }
59 }
```