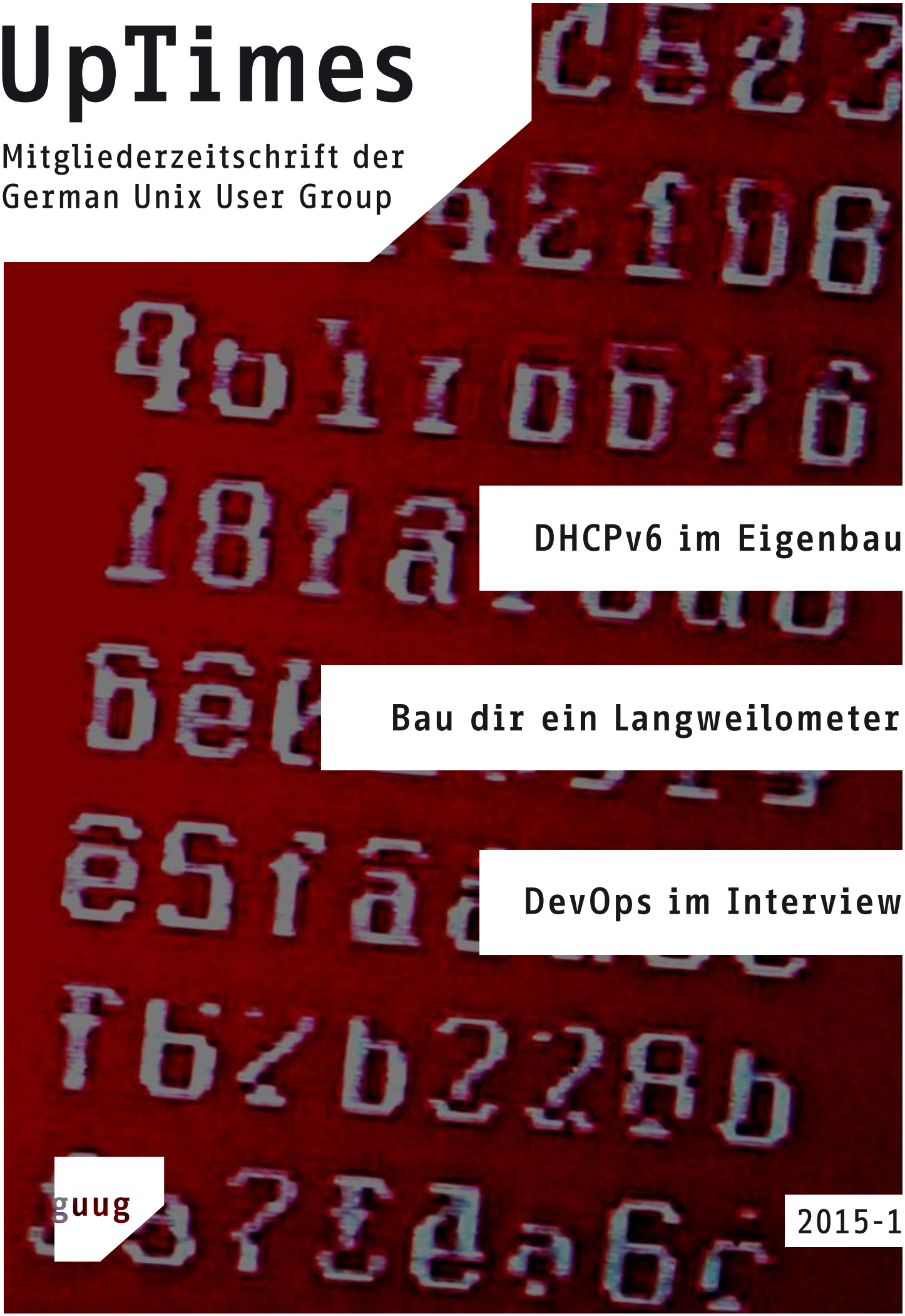
UpTimes

Mitgliederzeitschrift der
German Unix User Group



DHCPv6 im Eigenbau

Bau dir ein Langweilometer

DevOps im Interview

guug

2015-1

Inhaltsverzeichnis

Hallo miteinander!
Gruß aus der Redaktion

Viel lässt sich über Hindernisse, Schwierigkeiten, Wenns und Abers sagen. Hier ist das Allheilmittel: Loslegen. Artikel schreiben ist einfacher, als du denkst.

von **Anika Kehrer, Chefredakteurin**

Verbringe die Zeit nicht
mit der Suche nach einem Hindernis.
Vielleicht ist keines da.

Franz Kafka

Ich habe eine Vermutung. Sie betrifft dich. Meine Vermutung ist, dass du als GUUG-Mitglied oder GUUG-affiner Leser viel zu sagen hast, was all die anderen GUUG-Mitglieder und GUUG-affinen Leser brennend interessiert. Deswegen tingelt man auf Konferenzen, lungert auf Mailinglisten herum und schielt nach Kommentaren, die Buletten erklären. (Also Gehacktes. Also Getipptes. Klar?)

Bei „Artikel schreiben" ergreifst du aber die Flucht. „Keine Zeit", rufst du als erstes. „Zu anstrengend", murmelst du als zweites. „Kannichnich", denkst du als drittes vielleicht. Kein Wunder. Manche Leute halten ganze Vorträge darüber, was dazu gehört, Fachartikel zu schreiben [1]. Sowas aber auch. Und jetzt guck nochmal in den Vorspann dieses Artikels hier. Mach mal.

Jetzt passiert nämlich Folgendes: Du öffnest deinen Lieblingstexteditor. Schreib ihm einen Brief. Nicht komisch gucken, **loslegen** sollst du! Von mir aus schreibst du den Brief auch an dich selbst, an Linus Torvalds, deinen Hamster oder die UpTimes-Redaktion. Ist mir egal. Auf jeden Fall schreibst du jetzt mal jemandem – sofort und so konkret wie es nur geht –, welcher Problemfall oder Lösungsweg rund um Unix, Linux, Programmieren oder Administrieren dich zuletzt beschäftigt hat. Schreib ihn auf. Erzähl ihn einfach mal.

Einfach braindumpen

Ich vermute: Was du so beschreibst, ist verdammt interessant. Autor Henri Wahl hat auch einfach mal aufgeschrieben, was er sich so ausgedacht hat. Und findest du den Artikel *Dualstack dank eigenem DHCPv6-Server* von Henri Wahl etwa uninteressant? Na also.

Den Anfang macht ein schlichter Braindump. Manchmal hilft es, ein Gegenüber zu haben, damit man im eigenen Dump nicht versinkt – daher die Briefgeschichte als schräge Anregung. Manchmal mag etwas aber auch einfach rausfließen, etwa weil es nagt oder verwirrt. So ist der Artikel *Identifikation, Authentisierung, Autorisierung* von Kristian Köhntopp entstanden: Er war ein klassisches "Mir-fällt-da-was-auf-ich-sortier-das-jetzt-mal". Er war ein Google-Plus-Posting, wurde zum Fundstück der Redaktion und ist jetzt ein Artikel.

Manchmal reicht aber auch das nicht. Du musst etwas nachschlagen, weil du sonst zu allgemein bleibst? Rumeiern ist tötlich – auch gegenüber dir selbst. Also eiere nicht, sondern schlag es nach. Jetzt gleich. Du hast mal eine Skizze gekritzelt? Erzähle davon und beschreibe sie in Worten. Du musst in eine alte Konfig-Datei gucken? Such sie sofort und kopier gleich die betreffenden Zeilen in deinen Brief. Direkt dazu schreibst du, was daran doof oder nützlich war. Schwups, bist du beim Artikelschreiben.

Die Wahrscheinlichkeit ist nicht gering, dass das, was du darlegst, auch für andere interessant ist. Etwas knifflig wird es lediglich, wenn Listings ins Spiel kommen. Davon zeugt der Bastel-Artikel *Bau dir ein Langweilometer* von Jürgen Plate, dem Autoren, der glücklicherweise viel spaßprogrammiert, die Redaktion aber regelmäßig vor Formatierungsprobleme stellt. Das darf man aber. :-)

Hack dich selbst: Reverse Writing

Vielleicht ist einfach mal erzählen aber kein Motiv für dich. Code ist konkreter. Er eignet sich hervorragend, um einen UpTimes-Artikel drum herum zu bauen – sozusagen als Reverse Writing eines Artikels. Code muss man nämlich allermeistens erklären. Man darf ihn nicht einfach für sich selbst sprechen lassen – sonst verpufft sein Sinn.

Schon mal über fehlende Doku geschimpft? Siehste.

Es liegt doch bestimmt ein Skript, eine Konfigurationsdatei oder ein Programm bei dir herum, das nicht unbedingt trivial ist. Und das Dokumentation mal gut gebrauchen könnte. Mach das doch mal, das Dokumentieren. Und zwar gleich so, dass ein Artikel dabei entsteht. Der Weg zu einem Programmier-, Sysadmin- oder Netzwerkartikel in der UpTimes besteht aus fünf Schritten:

- Kopier die **Quelltextdatei** und leg sie bei den eigenen Dokumenten ab, um sie aus dem surrenden Maschinenpark in den sonnigen Hinterhof zu holen.
- Auch wenn du sie sonst nicht leiden kannst: Bringe **Zeilennummern** an alle Code-Zeilen an. So etwas wie Header und Variablendeklarationen lässt du aus, nur der eigentliche Programm-Code zählt. Es sei denn natürlich, du benutzt besondere Headerdateien, Variablen oder Konstanten. – Ach, was solls, nummeriere einfach alle Zeilen.
- Schreib jetzt ganz oben drüber in die Datei, warum und mit welchem Ergebnis der ganze Kram entstanden ist. Mindestens zehn Zeilen, darf aber mehr sein: Erklär herum, diskutiere mit dir selbst oder argumentiere mit Aussagen Dritter. Sieh es als ausführliches *tl;dr*, als Abstract, als **Plädoyer**, warum dieser Code für deinen speziellen Fall wichtig ist. Für wen er vielleicht noch wichtig sein könnte. Es wird dir später dienlich sein, um den Einstieg zu finden, dem roten Faden zu folgen oder Schlussfolgerungen zu identifizieren. Spätestens jetzt meldest du dich bei der UpTimes-Redaktion, dass du da was ausbrütest. Auch, um dir Enttäuschung zu ersparen: Was einen Artikel relevant macht, lässt sich besser im Mehraugenprinzip klären.
- Jetzt kommt das Herzstück: Guck dir an, welche Teile des Codes besonders wichtig sind. Wo der Hund begraben liegt. Des Pudels Kern. Weißt schon. Der Trick ist: Diese **Ausschnitte** und Schnippsel werden später zu den Listings für Deinen Artikel, um die du die Geschichte strickst. Den kompletten Code stelle besser als Download bereit – Programmtext nur so abzudrucken, überlastet einen Artikel. Indem du Ausschnitte auswählst und begründest, machst du den Schritt vom technischen Dokumentieren zum redaktionellen (Be-)Schreiben.
- Jetzt erklärst du einzeln alle deine ausgewählten Zeilen. Zunächst klassisch als **Kommentare**: Schreib in den Code, was genau eine Zeile tut, welche anderen Zeilen sie braucht, welche Ideen ihr zu Grunde liegen. Diskussionspunkte, Alternativen, Wenns und Abers dürfen sein: Wenn du ins Schwafeln kommst, schwafle. Offenbar gibt es dann viel zu zu sagen. Du wirst sehen, dass du dir bald einen eigenen Erklärtext wünschst, anstatt zwischen den Zeilen zu reden. Und prompt freust du dich über die Zeilennummern, mit denen du alle Zeilen fein säuberlich referenzieren kannst.

Ich tippe, du merkst durch das Kommentieren und Erklären, wie dein Code eine Metaebene erhält. Das ist die Ebene des Artikels. Irgendwann benennst du dann die Datei in ein TXT um. Wenn du nämlich genug in den Kommentaren geschwafelt hast, ziehst du sie aus dem Code heraus und mergst sie zu einem eigenen Text (den so genannten Fließtext, den die beschriebenen Listings ergänzen). Daran bindest du das Warum-und-Wieso an, das du oben drüber geschrieben hast. Dazu kommen dann sicher noch weitere Gedanken – wirst schon sehen. Nochmal: Behalte unbedingt das mit den Zeilennummern bei. Ohne Zeilennummern kannst du nicht vernünftig den Code referenzieren.

Probiere mal, was da so passiert. Am Ende stellst du fest: Dein Programmier-, Sysadmin- oder Netzwerkartikel ist quasi fertig. Ja: Im Nachhinein besehen, wird das Ganze aufwändig gewesen sein. So aufwändig, dass du vermutlich nie angefangen hättest, wäre das Ganze absehbar gewesen. Wäre aber schade drum. Und einfach mal hinten anzufangen, macht doch auch viel mehr Spaß, oder?

[1] Anika Kehrer, Vortrag „Fachartikel schreiben“, FFG 2015:
https://www.torial.com/anika.kehrer/portfolio/78266

Über Anika

Anika Kehrer arbeitet seit acht Jahren als Technikjournalistin und ist seit der Wiederauflage der UpTimes im Sommer 2012 von der GUUG als Chefredakteurin beauftragt. Ihr Ziel ist, die UpTimes als Vereinszeitschrift der GUUG in Zusammenarbeit mit dem Redaktionsteam aus GUUG-Mitgliedern als hochwertiges, transparentes und partizipatives Fachmagazin zu gestalten.

Das Was-ist-was des Wer-ist-wer
Identifikation, Authentisierung, Autorisierung

Nach Vorträgen ergeben sich oft längere Diskussionen über Anmeldeverfahren oder Passworte. Oft scheint es an Nuancenwissen zu fehlen. Der folgende Text entstand, um abends den Kopf frei zu bekommen. Da er zum Wegschmeißen zu schade war, landete er bei Google Plus – und von dort auf Anfrage in der UpTimes.

von **Kristian Köhntopp**

Fürchte dich nicht
vor der Verwirrung außer dir,
aber vor der Verwirrung in dir.

Friedrich Schiller

Wenn jemand mit einem Computersystem arbeitet, geschieht das nicht direkt, sondern der Rechner repräsentiert den Anwender intern mit einer Kennung, einer User-ID oder einem Namen. So ein Anwender wird also zum Beispiel mit der UID *500* und dem Namen *kris* zum User. Programme, die der Anwender startet, laufen unter dieser Kennung und führen in seinem Auftrag Operationen im System aus.

Die Identifier, Usernamen und User-IDs sind systemweit eindeutig, damit das System User voneinander unterscheiden kann. Sie sind natürlich nicht global eindeutig – Anwender A kann auf seinem System UID *500* haben und *kris* heißen, und Anwenderin B auf ihrem. Aber sie müssen innerhalb eines Namensraumes und einer Verwaltungseinheit eindeutig sein, innerhalb der so genannten *Administrative Domain*.

Laufen beide Rechner allein und tauschen keine Medien miteinander, entspricht dem User *kris* hier der Anwender A, dort die Anwenderin B. Alles fein. Sobald beide aber ihre Rechner zusammenschalten und etwa ein gemeinsames Netzwerklaufwerk verwenden, sind beide Teil desselben, größeren Systems. Sie müssen sich entscheiden, wer von beiden nun die UID *500* hat und wer stattdessen die UID *501*, und wer weiter *kris* heißt und wer ab jetzt zum Beispiel *kkoehntopp*.

Solche Umbenennungen und Umnummerierungen kommen häufig vor, wenn mehrere Verwaltungseinheiten zu einer werden, etwa bei Firmenaufkäufen oder Systemvernetzungen. Sie sind problematisch, weil sich eine unbekannte Menge an Dateien im Bestand befindet, die zu finden und anzupassen sind. Wenn außerdem zum Beispiel *Access Control Lists* im Einsatz sind, sind diese auch alle zu finden und zu ändern. Die Systeme dürfen erst verschmolzen werden, nachdem beide zusammenzuführenden *Administrative Domains* überlappungsfrei sind. Denn sind sie erst einmal verschmolzen, ist nicht mehr entscheidbar, ob eine Datei mit der UID *500* zu dem User-Namen *kris* von Anwender A oder Anwender B gehört.

Vielfach notiert man daher auch nicht nur den unqualifizierten User-Namen, sondern qualifiziert ihn mittels der Verwaltungseinheit – je nach Systemkonvention mit Punkten, Strichen oder Klammeraffen abgetrennt. Anwender A heißt dann zum Beispiel `KOEHNTOPP\\KRIS` und Anwenderin B `ANDEREFAMILIE\\KRIS`, oder der eine ist `kris@koehntopp.de` und die andere `kris@anderefamilie.ch`.

Meldet einer der beiden sich bei seinem System an, dann nutzt er diesen Namen. Er behauptet also, *kris* zu sein. Das ist seine Identifikation. (Oder ihre.)

Von Identifizierung zur Authentifizierung

Die Karre glaubt das aber nicht einfach – weder dem einen noch der anderen. Sie will, dass der Anwender nicht nur sagt, dass er der User *kris* ist, sondern es beweist. Der Anwender muss also zeigen, dass er der wahre, echte und reine *kris* ist und nicht irgendein zweitklassiger Ersatz. Das ist die Authentisierung.

Er zeigt das, indem er ein Geheimnis verwendet, das nur er kennt – das Passwort, die PIN, das Sesam-öffne-Dich, das Zauberwort. Das ist das übliche Login mit einem Passwort.

Alternativ zeigt Anwender *kris* etwas vor, das nur er hat, und das schwer zu duplizieren ist. Das kann eine Chipkarte sein, oder ein Schlüssel. Wer keines von beiden hat, kommt nicht hinein: Auch der wahre, echte und reine *kris* muss sich an seiner Wohnungstür durch Besitz des Haustürschlüssels authentisieren. (Und durch dessen korrekten Gebrauch, du Saufnase.)

Die Wohnungstür unterscheidet allerdings nicht zwischen verschiedenen Identitäten – Anwender A und seine Frau haben identische Haustürschlüssel. Die Tür öffnet sich für jeden, der einen passenden Schlüssel hat. Bei Türen an einer Uni oder in einem Hotel ist das zum Beispiel anders. Hier kommen Chipkarten zum Einsatz, die alle unterschiedlich sind. Jede Tür kennt eine Liste von passenden Schlüsseln. Und jede Tür kann eine andere Liste von passenden Schlüsseln haben.

Eine weitere Möglichkeit zu beweisen, wer man ist, sind persönliche Eigenschaften. Etwa das blendende Aussehen, die strahlenden Augen, die markanten Linien des Kinns oder die schmutzigen Fingerabdrücke machen einen Anwender kenntlich und unterscheiden ihn von anderen, billigen Imitationen.

Damit haben wir unterschiedliche Faktoren zur Authentisierung: geheimes Wissen, Besitz schwer fälschbarer Dinge, und einmalige Eigenschaften.

Einfaktor- und Zweifaktor-Authentisierung

Meldet sich Anwender A nun also bei einem Rechner durch Behauptung („Ich bin *kris*!“) und Beweis („Das geheime Passwort heißt *Flipperwaldt gersput*!“) an, ist das eine so genannte Einfaktor-Authentisierung. Braucht er darüber hinaus noch ein Dings, das er in den Rechner steckt, dann ist das eine Zweifaktor-Authentisierung (Abbildung 1).

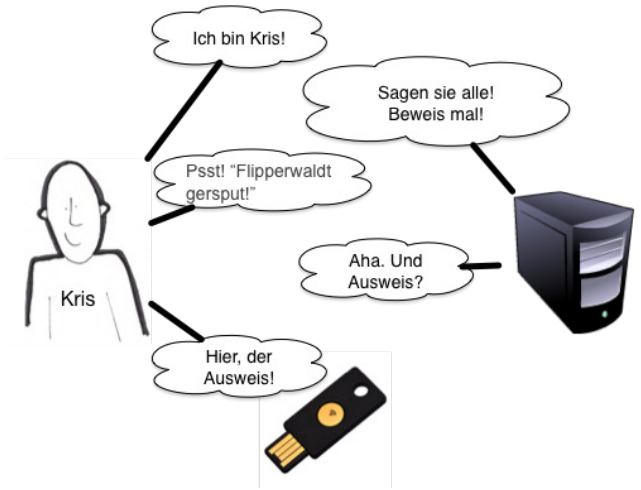


Abbildung 1: Die Zweifaktor-Authentisierung fügt dem einen Faktor – dem Passwort – einen zweiten hinzu, zum Beispiel ein Schlüssel-Device.

Letztere ist eine feine Sache, weil sie verhindert, dass jemand anders mit Kenntnis von User-Name und Passwort einen Account übernimmt. Stattdessen bräuchte der Angreifer dazu noch das Dings (zum Beispiel einen *Fido-Key* [1]) Das ist insbesondere für einen physisch nicht anwesenden Angreifer schwer. Das bedeutet, dass Zweifaktor-Authentisierung insbesondere vor Phishing schützt.

Wer Google-Dienste verwendet und keinen Dingens besitzt, pausiert ihr hier mal bitte mit dem Lesen und kauft eben schnell einen Fido-Key [2]. Er ist mit sechs Euro zuzüglich Versand spottbillig und verhindert sicher, dass jemand den Account abphisht. Der Text wartet solange.

Wieder da? Gut. Weiter im Text.

Von Authentisierung zur Autorisierung

Hat Anwender A-oder-B jetzt also behauptet, *kris* zu sein, und die Möhre glaubt das auch endlich, dann muss sie sich entscheiden, was ihr das wert ist. Sie muss wissen, wo der User *kris* heran darf und was er machen darf. Das ist seine Autorisierung.

Die User-Berechtigungen lassen sich prima in Dreiergruppen notieren, also in Sätzen mit Subjekt, Prädikat und Objekt. Da heißt es dann zum Beispiel: ‘*kris* darf */notizen.txt* mit dem Editor *vi* bearbeiten‘, oder kürzer: ‘*kris, vi, ~notizen.txt*‘. Daraus ist ersichtlich: Subjekte sind immer User-Identitäten, Objekte sind Dinge im Rechner – meist Dateien –, und die Verben des Prädikats entsprechen Programmen, mit denen der Zugriff darauf erfolgen darf.

In den meisten Fällen ist das Verb ein Wildcard, berechtigt also zu jedem beliebigen Programm. Die in Satz gebrachte Berechtigung heißt dann: ‘*kris* darf mit *~notizen.txt* machen, was immer er will‘. Ein Blankoscheck ist das jedoch nicht. Der Anwender kann zum Beispiel mit Vi sein User-Passwort in *etcshadow* nicht ändern. Das liegt nicht am dazu benutzten Programm, sondern daran, dass der User *kris* die Datei *etcshadow* weder lesen noch schreiben darf.

Er kann sein Passwort aber mit dem Programm *passwd* ändern. Es gilt also: “*kris, *, ~notizen.txt*‘, aber nur ‘*kris, binpasswd, etcshadow*‘. Und wenn User *kris* das Programm *binpasswd* aufruft, dann wird das Programm auch nur das Passwort von *kris* ändern, aber nicht dasjenige von anderen Usern. Mit Vi könnte ansonsten jeder in der Shadow-Datei herumtreckern und Passwörter ändern.

Die Rechte, über die User *kris* verfügt, sind entweder am Objekt notiert. Dann sind das so genannte Permissions (Abbildung 2). Die Datei *~notizen.txt* besitzt dann die Eigenschaft ‘*kris* darf

sie lesen und schreiben, und sonst keiner'. Analog kennt die Türsteuerung eines Türschlosses eine Liste von Schlüsselcodes, die die Tür öffnen (die Schlüsselkarte selbst entspricht nur einer Identität – zum Beispiel als Türkartennummer).

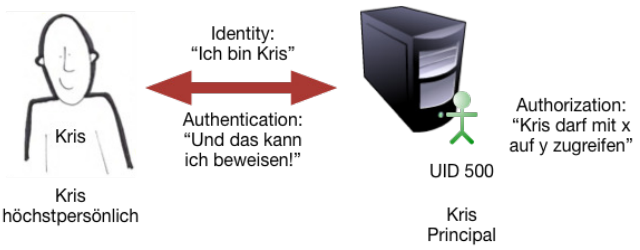


Abbildung 2: So gehören Identifikation, Authentisierung und Autorisierung zusammen.

Oder die Rechte sind am User notiert. Dann sind es so genannte Capabilities. User *kris* besitzt dann die Eigenschaft '*kris* darf die folgenden Dateien lesen: $LISTE'. Analog steht auf der Schlüsselkarte von *kris* eine signiert und unfälschbare Liste der Türnummern als Objektidentitäten, durch die der Karteninhaber hindurchgehen darf (der dann hoffentlich der User *kris* ist). An der Tür muss nichts hinterlegt sein außer dem Root-Zertifikat der Signatur, damit die Daten auf der Karte geprüft werden können.

Wer noch nicht genug hat: Das frei erhältliche Buch Security Engineering (zweite Auflage) [3] ist spannend, lehrreich und gut geschrieben, was bei diesem Thema in dieser Combo eher selten ist.

Links

[1] *FIDO* (Fast IDentity Online) Alliance: https://fidoalliance.org/about/overview/
[2] Fido U2F Security Key bei Amazon:
http://www.amazon.de/Plug-up-FIDO-U2F-U2F-SK-01-Security/dp/B00OGPO3ZS
[3] Ross J. Anderson: Security Engineering. A Guide to Building Dependable Distributed Systems, Wiley 2008 (2nd edition). http://www.cl.cam.ac.uk/~rja14/book.html

Über Kristian

Kristian Köhntopp arbeitet seit 2014 als Cloud Architect bei der SysEleven GmbH in Berlin. Vorher war er als Senior Scalability Engineer bei Booking.com in Amsterdam tätig, wo er die Datenbank-Architektur über ein Wachstum um den Faktor 100 begleitet hat. In früheren Leben war er außerdem als Consultant für MySQL AB und als Senior Security Engineer bei web.de tätig, hat an PHP mitentwickelt und ist Autor und Contributor verschiedener Linux-HOWTOs. Sein Stream ist unter http://plus.google.com/+KristianKöhntopp zu lesen.

Von einem, der auszog, IPv6-only zu leben
Hindernisse im IPv4-Netz

Wer in einer IPv6-Welt leben möchte, sieht sich an so manchen Ecken und Enden vor ungeahnte Hindernisse gestellt. Beispiele: das Nagios-Plugin *check_mk* und Puppet.

von **Jens Link**

Die Berge,
über die man am schwersten hinwegkommt,
häufen sich immer
aus Sandkörnchen auf.

Christian Friedrich Hebbel

Wir schreiben das Jahr 2015. IPv6 wird dieses Jahr 20 Jahre alt. Die Server, die man im Internet mieten kann, werden immer leistungsfähiger – Virtualisierung im Eigenbetrieb bietet sich an. Das Problem: Die Server bieten von der Hardware her genug Leitung für viele virtuelle Maschinen, aber man hat zu wenig IPv4-Adressen, um diese auch zu nutzen.

IPv6-Adressen sind hingegen meist mehr als genug erhältlich. Ausnahmen bestätigen zwar die Regel, aber es zwingt einen ja niemand, IPv6-befreite Provider zu nutzen. Also schaut man, welche VMs und welche Dienste nicht zwingend IPv4 brauchen. Hier war das zum Beispiel der Puppet-Master. Der IPv6-Betrieb brachte allerdings einige Probleme mit sich.

Erstes Problem: Monitoring

Der Autor nutzt *check_mk* [1] und hat auch schon bei dem einen oder anderen Vortrag begeistert davon berichtet. Er mag diesen Agent so sehr, dass er ihn auch bei Kunden einsetzt. Nur kann das Ding kein IPv6. Ein kurzer Blick in die Sourcen verrät, warum – er verwendet die Funktion `gethostbyname()` anstelle von `getaddrinfo()`:

```
ipa = socket.gethostbyname(hostname)
```

Der korrigierende Patch von Stefan Neufeind hat ganze drei Zeilen: Er ersetzt im Prinzip `gethostbyname()` durch `getaddrinfo()`. Der zweite Teil des Patches kümmert sich darum, dass IPv6 und IPv4 jeweils den passenden Sockettype erhalten (siehe Kasten *IPv6-Patch für Check_mk*).

Der Patch fand seinen Weg schon vor drei Jahren auf die Check_mk-Mailingliste [2]. Aber in den Code hat er es noch immer nicht geschafft. Das Check_mk des Autors spricht nun dank des Patches IPv6. Dafür fällt aber flach, einfach Updates einzuspielen.

Wer aber dachte, mit den drei Zeilen wäre es getan, der irrt. Check_mk ruft intern nämlich *check_icmp* des Monitoring-Plugins-Projekts auf. Der Leser darf raten, was Check_icmp nicht kann. Genau: IPv6!

Wenigstens verfügen die Entwickler über ein funktionierendes Bug-Tracking IPv6-Support für Check_icmp steht für Version 2.4 auf der Roadmap [3]. Als Quick-and-dirty-Lösung lässt sich in der Konfigurationsdatei *check_mk_templates.cfg* das Kommando *check_icmp check-mk-ping* durch *check_ping* ersetzen:

```
define command {
command_name check-mk-ping
command_line /usr/lib/nagios/plugins/check_ping \
  -H $HOSTADDRESS$ -w 120,90% -c 150,95%
}
```

Zweites Problem: Software-Download

Ab und zu will man fertige Puppetmodule oder anderen Code auf seinen Systemen verwenden. Doch weder Github noch Forge.puppetlabs.com ist per IPv6 erreichbar. Nachfragen ergaben, dass man sich bei Github irgendwann mal damit beschäftigen will – die Antworten schwanken zwischen „in den nächsten Monaten“ und „nächstes Jahr “. Bei Puppetlabs hat der Hoster wohl noch kein IPv6 [4].

Das Problem ließ sich mit einem Squid-Proxy auf einem Dualstack-Host lösen. Unschön, aber es ging wenigstens.

Drittes Problem: Wlan

Wer viel Bahn fährt, freut sich über kostenloses Wlan, allerdings leider nur mit dem altem IP. Ist

man dann IPv6 gewöhnt, kommt es schonmal vor, dass man sich fragt, warum SSH nicht geht. Gleiches gilt natürlich für das Wlan in Hotels, bei Konferenzen oder Gastzugängen beim Kunden. Der Workaround ist hier, einfach einen Dualstack-Host als Proxy zu verwenden.

Dieser Artikel soll ein paar der Fallen aufzeigen, in die man mit IPv6 stolpert. Bei kommerzieller oder selbstentwickelter Software wird es sicher noch viel mehr Probleme geben. Es empfiehlt sich also, endlich IPv6 zu implementieren, um in Ruhe lernen und testen zu können. Wer seine Funde zur Verbesserung der Lage dokumentieren möchte, trägt sie zum Beispiel hier [5] ein.

Übrigens: Zu der Zeit, als die letzten Korrekturen an diesem Artikel geschahen, gingen der *ARIN* (Vergabestelle für IPs in Nord-Amerika) die vorletzten IPv4 Adressen aus [6].

IPv6-Patch für *check_mk*

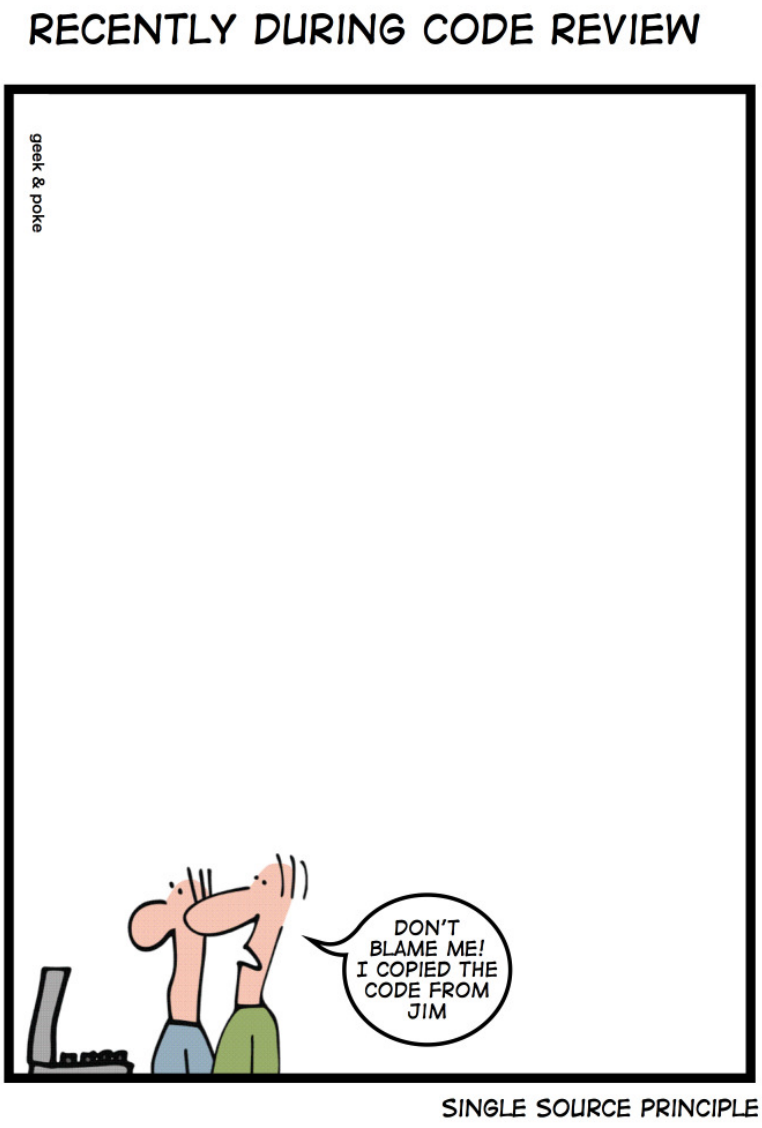
```
diff --git a/modules/check_mk.py b/modules/check_mk.py
index f79f889..5dc770b 100755
--- a/modules/check_mk.py
+++ b/modules/check_mk.py
@@ -1031,7 +1031,8 @@ def lookup_ipaddress(hostname):

     # To do the actual DNS lookup
     try:
-        ipa = socket.gethostbyname(hostname)
+        ipaddrinfo = socket.getaddrinfo(hostname, None)
+        ipa = ipaddrinfo[0][4][0]
     except:
         g_dns_cache[hostname] = None
         raise

diff --git a/modules/check_mk_base.py b/modules/check_mk_base.py
index d87eee8..d09a0f7 100755
--- a/modules/check_mk_base.py
+++ b/modules/check_mk_base.py
@@ -460,7 +460,11 @@ def get_agent_info_tcp(hostname, ipaddress):
     if not ipaddress:
         raise MKGeneralException("Cannot contact agent: host '%s' has
no IP address." % hostname)
     try:
-        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
+        if ":" in ipaddress:
+            socktype = socket.AF_INET6
+        else:
+            socktype = socket.AF_INET
+        s = socket.socket(socktype, socket.SOCK_STREAM)
         try:
             s.settimeout(tcp_connect_timeout)
         except:
```

Links

[1] Projektseite des Monitoring-Tools *check_mk*: https://mathias-kettner.de/check_mk.html
[2] E-Mail mit Patch für *check_mk* auf der Check_mk-Mailingliste:
https://www.mail-archive.com/checkmk-en@lists.mathias-kettner.de/msg06908.html
[3] Bugreport und Roadmap-Aufnahme von IPv6-Support in Check_icmp:
https://github.com/monitoring-plugins/monitoring-plugins/issues/1291
[4] Bugreport und Antwort bei Puppetlabs: https://tickets.puppetlabs.com/browse/FORGE-260
[5] Sammelstelle für IPv6-Hindernisse: http://wiki.test-ipv6.com/wiki/Main_Page
[6] „Nordamerikas IPv4-Adresspool fast leer “ (25.06.2015): http://heise.de/-2729025

Über Jens

Jens Link ist freiberuflicher Computer-Geek, der sich auf Netzwerke hauptsächlich von Cisco spezialisiert. Zum Monitoren nutzt er gern freie Software wie Icinga oder Cacti. Besonders interessiert er sich für IPv6.



Unterwegs nach morgen
Dualstack dank eigenem DHCPv6-Server

Das Leibniz-Institut für Festkörper- und Werkstoffforschung in Dresden implementiert IPv6, wie in den meisten Fällen, durch die Koexistenz von IPv4 und IPv6 (Dualstack). Dieser Artikel zeigt, wie sich dank des DHCPv6-Servers *dhcpy6d* die mit IPv4 genutzte Clientverwaltung auch mit IPv6 verträgt.

von **Henri Wahl**

Je länger man vor der Tür zögert,
desto fremder wird man.

Franz Kafka

Im Institut sind die Hosts des LAN verschiedenen Rollen zugeordnet. Den Rollen wiederum sind verschiedene IPv4-Adressbereiche zugeteilt, die der DHCP-Server vergibt. Die Identifikation der Hosts erfolgt über ihre MAC-Adresse, die sie bei ihren DHCP-Requests übermitteln. Zudem erledigt der DHCP-Server zentral die Aktualisierung der Hostnamen im DNS. Dieses klassische Verfahren funktioniert mit IPv4 hervorragend. Für den Einsatz auch mit IPv6 waren jedoch einige Stolpersteine aus dem Weg zu räumen. Unter anderem war ein geeignetes Verfahren für die automatische Adressverteilung zu finden, und der Umgang mit den für die Identifizierung der Clients wichtigen MAC-Adressen war zu klären.

Schritt 1: DHCPv6 statt SLAAC

Für die automatische Verteilung von IPv6-Adressen für Clients gibt es derzeit zwei Verfahren: *Stateless Address Autoconfiguration* (SLAAC) [1] und das *Dynamic Host Configuration Protocol version 6* (DHCPv6) [2]. SLAAC bedarf zwar weniger Aufwand als DHCPv6, erlaubt allerdings auch weit weniger Kontrolle. Im Heimbereich mag SLAAC also gute Dienste leisten, in einem größeren Netz ist es nur bedingt eine Option. Stattdessen bietet sich Stateful DHCPv6 an:

- Die Zuordnung der Clients je nach Rolle zu verschiedenen Adressbereichen ist mit SLAAC unmöglich. Denn hier gibt der Router in seinen *Router Advertisements* (RA) lediglich das `/64`-Präfix vor, aus dem sich die Clients ihre eigene IPv6-Adresse erstellen. Welche Rolle sie aus Sicht einer Adressverwaltung haben, ist ihnen egal. Ein vollwertiger DHCPv6-Server sollte aber imstande sein, verschiedenen Clients Adressen aus verschiedenen Bereichen zuzuteilen.
- Bei den vielfach längeren IPv6-Adressen ist ein konsistentes DNS notwendig. Aufgrund seiner dezentralen Natur ist bei SLAAC keine gesteuerte Aktualisierung des DNS möglich. Wenn überhaupt, könnten die Clients diese Aufgabe übernehmen, was aber meist nicht gewünscht ist, da nicht jeder das DNS modifizieren darf. Im Gegensatz dazu sollte ein DHCPv6-Server das DNS mit den von ihm verteilten Adressen zentral aktualisieren können.

Somit ist die einzige Chance, die bestehende Verwaltung der Clients von IPv4 nach IPv6 zu übertragen, eine Stateful-DHCPv6-basierte Lösung. Stateful DHCPv6 ist insofern mit DHCPv4 vergleichbar, als dass es zentral Clients und ihre Adressen verwaltet. Es gibt auch Stateless DHCPv6, welches als Ergänzung zu SLAAC die Clients nur mit Informationen etwa zu Nameservern beliefert, aber nicht mit Adressen. Im weiteren Text ist der Einfachheit halber mit DHCPv6 immer Stateful DHCPv6 gemeint.

Schritt 2: MAC statt DUID

Laut RFC 3315 werden DHCPv6-Clients durch ihren *DHCP Unique Identifier* (DUID) identifiziert. Dieser sieht zum Beispiel so aus: `00043e1cd8e1646011c148eceaef3ab7922c`. Er lässt sich auf verschiedene Arten erzeugen, die jedoch einige unangenehme Eigenschaften gemeinsam haben:

- Das Betriebssystem erstellt den DUID. Installiert man es neu, ändert sich auch der DUID.
- Der DUID gestattet keinen sicheren Rückschluss auf die Hardware. Es wird irgendeine MAC-Adresse des Clients in ihm eingebettet - das muss nicht zwangsläufig die des aktuellen Netzwerkinterfaces sein. Zudem bleibt der

DUID auch bei Hardware-Änderungen erhalten, wodurch gar keine Aussage über die tatsächliche MAC-Adresse mehr möglich ist.

- Die Erfassung und Darstellung des DUID, um ihn beispielweise in einer zentralen Datenbank zur späteren Client-Identifikation zu speichern, ist in verschiedenen Betriebssystemen und Client-Geräten für den Admin im Vergleich zu herkömmlichen MAC-Adressen aufwändig.

Im genannten RFC finden MAC-Adressen nur Verwendung beim Generieren des DUID. Der RFC untersagt ausdrücklich ihre Ableitung aus dem DUID. Allmählich setzt sich die Erkenntnis durch, dass einige Aspekte der DHCPv6-Spezifikation etwas praxisfern sind. Gerade das Fehlen der MAC-Adresse als Identifikationsmerkmal diskutieren Admins seit Jahren immer wieder zum Beispiel in Mailinglisten [3].

Abhilfe verspricht der RFC 6939 [4]. Er stellt die so genannte *Client Link-Layer Option* vor, die die MAC-Adresse des Clients enthält. Diese Option wird allerdings von einem Relay-Agenten zu einer weitergeleiteten Nachricht hinzugefügt, was den RFC für Netze ohne Relays nutzlos macht. Überdies unterstützen bisher nur wenige Hersteller diesen RFC.

Im Ergebnis machen gemäß RFC 3315 sämtliche gängigen DHCPv6-Server – etwa *ISC DHCP*, *WIDE-DHCPv6* oder *Dibbler* – einen Bogen um die MAC-Adressen ihrer Clients. Damit waren sie allesamt nicht geeignet für das hier gestellte Ziel, die Clients analog zu IPv4 anhand ihrer Hardware zu identifizieren und mit ihrer Rolle entsprechenden IPv6-Adressen zu versehen. Die Lösung dieses Dilemmas war, selbst einen DHCPv6-Server zu entwickeln: *dhcpy6d*.

Neighbour Cache als Deus ex machina

Der Ausweg führt über den *Neighbour Cache*. Dieser ist bei IPv6 das Pendant zum *ARP Cache* bei IPv4. Hier werden die Link-Local- und MAC-Adressen paarweise zwischengespeichert (Abbildung 1).

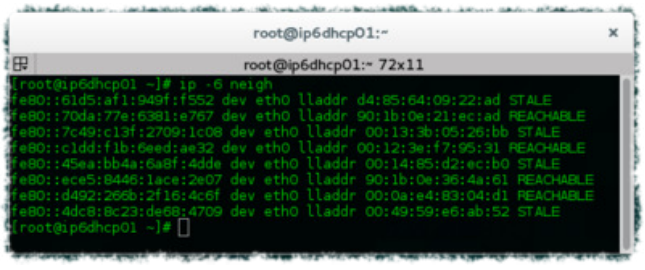


Abbildung 1: Anzeige des *Neighbour Cache* in Linux mittels `ip -6 neigh` [6]

Eine Anfrage des Clients an den DHCPv6-Server enthält immer auch die Link-Local-Adresse des Clients. Diese selbst kann nicht als dauerhaftes Identifikationsmerkmal dienen, da sie sich bei den zunehmend verwendeten Privacy-Extensions der Clients ständig verändert. Sie kann jedoch zum Nachschlagen im Neighbour Cache verwendet werden, um dort die entsprechende MAC-Adresse herauszufinden.

Ist ein Paar aus Link-Local- und MAC-Adresse noch nicht im Neighbour Cache vorhanden, so ermittelt es der Server, indem er eine Dummy-Antwort an die Link-Local-Absenderadresse des Clients zurückschicken lässt. Diese Antwort löst zwei Aktionen aus:

- Auflösen der MAC-Adresse durch das *Neighbour Discovery Protocol*. Es ist Teil von IPv6 und wird somit von allen Clients verstanden. Anhand dieses Protokolls ermittelt das Betriebssystem selbständig die zur Link-Local-Adresse des Clients gehörige MAC-Adresse, und fügt diese neu gewonnenen Information dem Neighbour Cache hinzu.
- Der Client wiederholt seine Anfrage an den DHCPv6-Server.

Bei der erneuten Anfrage des Clients ist er anhand seiner nun bekannten MAC-Adresse identifizierbar. Der Server kann ihn entsprechend der ihm laut Adressverwaltung zugewiesenen Rolle mit IPv6-Adressen versehen (Abbildung 2).

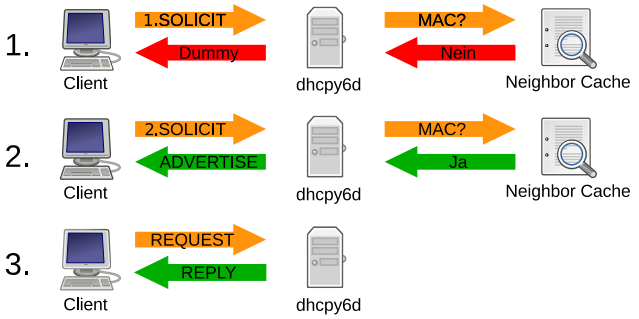


Abbildung 2: Abfrage des *Neighbour Cache* bei unbekannten Clients.

Der DHCPv6-Server *dhcpy6d*

Der vom Autoren geschriebene MAC-verarbeitende DHCPv6-Server *dhcpy6d* läuft im Leibniz-Institut für Festkörper- und Werkstoffforschung mit hunderten Clients seit über fünf Jahren. An der Verwaltung der IPv4-Adressen hat

sich nichts geändert. Die MAC-basierte Identifikation und Rollenzuordnung der Clients ließ sich für IPv6 ohne Probleme übernehmen.

Dadurch, dass Dhcpy6d inhäusig entsteht, ließen sich bisher nicht verfügbare Funktionen ergänzen, beispielsweise die Ausgabe von zufälligen Adressen analog zu Privacy-Extensions, oder direkte Datenbankanbindung. Neue Features gelangen gleich in den Praxistest, sodass sich auftretende Probleme sofort beheben lassen. Dhcpy6d 0.4.1 verfügt derzeit über folgende Features:

- Adressen lassen sich aus einem Bereich, aus individuellen Client-IDs, aus der MAC-Adresse oder zufällig generieren.
- Sowohl die MAC-Adresse, als auch DUID, als auch Hostname, oder eine Kombination aus allen drei Attributen identifizieren Clients.
- Clients können mehrere Adressen mit verschiedenen Präfixen erhalten.
- Clients lassen sich in Klassen organisieren.
- Client-Informationen können aus einer Datenbank oder einer Konfigurationsdatei stammen.
- Der Admin kann Clients anhand von Regular Expressions nach MAC, DUID oder Hostname filtern und Klassen zuordnen.
- Der Admin kann DNS-Updates an einen Bind-Server senden.
- Der Server unterstützt *Rapid Commit*.
- Der Server kann Anfragen an mehreren Netzwerkschnittstellen entgegen nehmen.

Die Versionsnummer zeigt, dass die Entwicklung noch nicht abgeschlossen ist. Mit dem jetzigen Stand hat sich Dhcpy6d aber bereits als praktisch einsetzbar erwiesen.

Dhcpy6d konfigurieren

Standardmäßig konfiguriert die Datei */etc/dhcpy6d.conf* den Dhcpy6d-Server. Sie definiert unter anderem Lifetimes, Nameserver, Logging, Adressen und Klassen. Gelangen Client-Informationen nicht aus einer Datenbank zum Server, so können sie auch einer Textdatei entstammen. In der Regel ist das */etc/dhcpy6d-clients.conf*.

Die Konfigurationsdateien liegen im altbekannten INI-Format vor und bestehen aus verschiedenen Sektionen. Die Sektion *[dhcpy6d]* in `/etc/dhcpy6d.conf` enthält grundsätzliche Einstellungen wie Nameserver, Lifetimes und Interfaces. Auch sind hier Logging, Methoden zur Identifizierung der Clients und dynamische DNS-Updates festgelegt.

Verschiedene Client-Klassen konfigurieren die Abschnitte in *[class_<name>]*. Notwendig ist hier die Angabe der in der Klasse verwendeten Adressen. Optional sind verschiedene Eigenschaften wie Interface, Filter, Lifetimes und von der Standardeinstellung abweichende Nameserver.

Die in den Klassen verwendeten Adressen legen die *[address_<name>]*-Abschnitte fest. Ein solcher Abschnitt muss eine der Kategorien MAC, Range, Random oder ID enthalten sowie ein Muster, um die Adresse zu bilden. In dem Muster steht die Variable *$<category>$* als ein für diese Kategorie typischer Wert. Optional sind Angaben für dynamische DNS-Updates und Lifetimes möglich, die dann nur für diese Adresse gelten.

Zwei Sektionen sind im Server immer vorhanden, auch wenn sie nicht in der Konfigurationsdatei */etc/dhcpy6d.conf* stehen müssen: *[class_default]* und *[address_default]*. Die Klasse *default* trifft für alle Clients zu, die zu keiner anderen Klasse gehören. Diese Clients erhalten dann Adressen vom Typ *default*. Die Konfigurationsdatei kann *[class_default]* und *[address_default]* aber auch explizit angeben, um hier eigene Werte einzustellen.

Die minimale Konfigurationsdatei `dhcpy6d.conf` für den Dhcpy6d-Server sieht beispielsweise so aus:

```
01 [dhcpy6d]
02 really_do_it = yes
03 interface = eth1
04 store_config = file
05 store_file_config = dhcpy6d-clients.conf
06 store_volatile = sqlite
07 store_sqlite_volatile = volatile.sqlite
08 server_duid = 000100013442452123456816 61
09 server_preference = 255
10 domain = exampledomain.org
11 nameserver = fd00::53
12
13 [address_global]
14 category = random
15 pattern = 2001:db8::$random64$
16
17 [address_local]
18 category = range
19 range = 1000-1fff
20 pattern = fd00::$range$
21
22 [class_full-access]
23 addresses = global local
24
25 [class_only-local]
26 addresses = local
```

Wichtig ist die Einstellung `really_do_it` in Zeile 02, damit Clients tatsächlich eine Antwort bekommen. Diese Sicherung verhindert Chaos im Netz durch unkonfigurierte Server. Zeile 03 nennt das Interface, auf dem der Server Anfragen erwartet. Die Zeilen 04 bis 05 geben die Herkunft der

Client-Informationen an, in diesem Fall aus der Datei *dhcpy6d-clients.conf*. Zeilen 06 und 07 speichern die Leases in einer SQLite-Datenbank.

Zeile 08 enthält den Server-DUID. Steht hier nichts, so würfelt sich Dhcpy6d beim nächsten Start einen neuen. Es ist jedoch empfehlenswert, einen Server-DUID festzulegen, da Clients mitunter irritiert auf wechselnde Server-DUIDs reagieren. Durch Zeile 09 erhält dieser DHCPv6-Server im Netz die höchste Priorität. Die Domain aus Zeile 10 wird an die Clients übergeben, ebenso wie der Nameserver aus Zeile 11.

Die Zeilen 13 bis 15 definieren den Adresstyp *global* als Privacy-Extension-artige Zufallsadresse. Zeilen 17 bis 20 bestimmen den Adresstyp *local* als fortlaufende Adresse aus dem Bereich, den die Zeilen 19 bis 20 festlegen.

Die Zeilen 22 und 23 bestimmen die Klasse *full-access*, die die beiden Adresstypen *global* und *local* enthält. Clients aus dieser Klasse bekommen also zwei Adressen zugeteilt. Laut der Zeilen 25 und 26 bekommen Clients der Klasse *only-local* hingegen nur eine Adresse.

Eine beispielhafte Konfigurationsdatei *dhcpy6d-clients.conf* für die Client-Informationen sieht so aus. Zeile 01 legt den Client *exampleclient* fest. Er ist anhand der MAC-Adresse aus Zeile 02 zu identifizieren und gehört laut Zeile 03 zur Klasse *full-access*. Ein weiterer Client *anotherexampleclient* gehört zur Klasse *only-local* (Zeilen 05 bis 07):

```
01 [exampleclient]
02 mac = 00:08:15:12:34:56
03 class = full-access
04
05 [anotherexampleclient]
06 mac = 12:34:56:78:90:10
07 class = only-local
```

Abbildung 3 zeigt nun, wie der Server anhand dieser Konfigurationen die Adresse für einen Client generiert. Er identifiziert den Client anhand seiner MAC-Adresse als *exampleclient*, welcher – laut eben gezeigter Konfiguration – der Klasse *full-access* zugeordnet ist und somit zwei Adressen erhält, nämlich eine *global* und eine *local*. Die globalen Adressen gehören zur Kategorie *random* (siehe oben, Zeile 14). Der Server erzeugt sie also zufällig aus dem Bereich `2001:db8::0000:0000:0000:0000 – 2001:db8::ffff:ffff:ffff:ffff`, analog zu Privacy-Extension-Adressen. Die lokalen Adressen gehören zur Kategorie *range* (siehe oben, Zeile 18) und stammen hier im Beispiel aus dem Bereich `fd00::1000 – fd00::1fff`. Dabei werden sie fortlaufend vom Server vergeben.

Das Ergebnis ist wie gewünscht: Der Client erhält auf seine DHCP-Anfrage die zwei IPv6-Adressen *2001:db8::3425:6ad3:7b25:9a72* und *fd00::1000*.

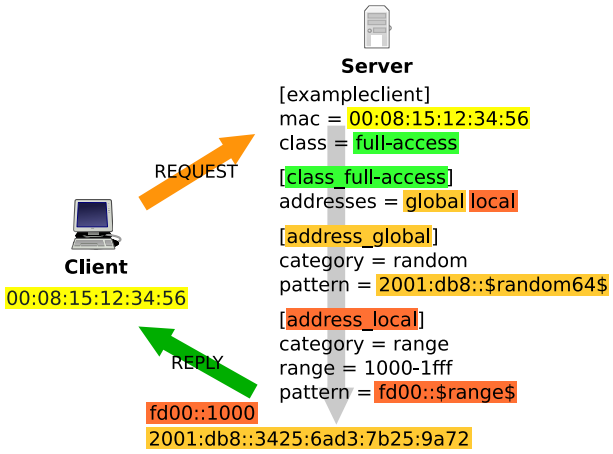


Abbildung 3: Generierung von Adressen für einen Client aus einer Klasse und deren Adressdefinition.

Potenzial und Grenzen

Dhcpy6d ist das Ergebnis davon, dass für den Dualstack-Betrieb brauchbare DHCPv6-Server fehlten. Der neue Server ermöglicht, die für IPv4 vorhandene Infrastruktur zur Clientverwaltung auch für IPv6 zu nutzen. In Netzwerken, in denen sich der Server zusammen mit seinen Clients in demselben Netzsegment befindet, kann Dhcpy6d dank der Auswertung des Neighbour Cache IPv6-Adressen MAC-basiert verteilen und somit sogar einige Unzulänglichkeiten des RFC 3315 ausgleichen. Mindestens in kleinen und mittleren Netzen ist Dhcpy6d dann eine Option, wenn die automatische Adressverteilung an Clients die Einführung von IPv6 ausbremst.

Im Alltag gibt bei der DHCPv6-Fähigkeit der Clients große Unterschiede. Am besten schlagen sich folgende Desktop-Betriebssysteme:

- Windows-PCs ab Windows 7 spielen vorbildlich mit und nehmen klaglos auch mehrere Adressen an. Sie bevorzugen per DHCPv6 übermittelte Einstellungen, wie den DNS-Server.
- Mac-OSX-Geräte verstehen ab Version 10.8 Stateful DHCPv6. Leider akzeptieren sie nur eine Adresse, ganz gleich, wie viele sie angeboten bekommen.
- Linux-Desktops könnten ein besseres Bild abgeben, wenn der oft verwendete Network Manager die guten DHCPv6-Fähigkeiten des ihm

zu Grunde liegenden ISC-Dhclients nutzen würde. Stattdessen wählt der Network Manager wie bei MacOSX scheinbar willkürlich nur eine Adresse unter den angebotenen aus [5].

Andere Geräte wie Drucker verhalten sich zum Teil unvorhersehbar. Selbst beim selben Hersteller treten die unterschiedlichsten und merkwürdigsten Anfragen und Fehler auf, so dass in diesem Bereich ein praktischer Einsatz noch nicht absehbar ist.

Dhcpy6d ist in Python geschrieben und unter der GPL2 lizensiert, also frei verfügbar. Getestet ist es unter Linux, OpenBSD, FreeBSD und NetBSD. Die Projektseite [7] enthält die vollständige Dokumentation und Downloads. Eine weitere Möglichkeit, auf den Code zuzugreifen, ist das Repository auf Github [8]. Seit Version 8 *Jessie* ist Dhcpy6d auch in Debian enthalten [9].

Links

[1] RFC 4862 *IPv6 Stateless Address Autoconfiguration* von 2007:
https://tools.ietf.org/html/rfc4862
[2] RFC 3315 *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)* von 2003:
https://tools.ietf.org/html/rfc3315
[3] Diskussion über fehlendes Identifikationsmerkmal MAC-Adresse:
https://lists.isc.org/pipermail/dhcp-users/2012-January/thread.html#14708
[4] RFC 6939 *Client Link-Layer Address Option in DHCPv6* von 2013:
https://tools.ietf.org/html/rfc6939
[5] Bugreport und Diskussion beim Network Manager:
https://bugzilla.gnome.org/show_bug.cgi?id=681764
[6] Manpage zu `ip-neighbour`:
http://www.dsm.fordham.edu/cgi-bin/man-cgi.pl?topic=ip-neighbour&ampsect=8
[7] Projektseite von Dhcpy6d: https://dhcpy6d.ifw-dresden.de
[8] Github-Repo von Dhcpy6d: https://github.com/HenriWahl/dhcpy6d
[9] Debian-Paket von Dhcpy6d (ab Debian Jessie): https://packages.debian.org/jessie/dhcpy6d

Über Henri

Henri Wahl ist seit 2007 Systemadministrator für Unix/Linux im Leibniz-Institut für Festkörper- und Werkstoffforschung (IFW) in Dresden. Schwerpunkte seiner Arbeit sind die Betreuung von Mailservern, Webproxies, Firewalls, Intrusion Detection und Monitoring auf Linux- und OpenBSD-Systemen. Aus Mangel an vorhandenen Lösungen und aus Begeisterung für Open Source und Python schuf er Projekte wie *Nagstamon* (https://nagstamon.ifw-dresden.de) und Dhcpy6d.

Ist dir langweilig?
Bau dir ein Langweilometer

Das Langweilometer war ein Spaßprojekt während der Leitung eines Projektpraktikums an der Hochschule München. Es hilft zwar nicht gegen Langeweile, kann diese aber dokumentieren.

von **Jürgen Plate**

Bekanntlich werden Übel
dadurch erleichtert,
dass man sie
gemeinschaftlich erträgt.

Arthur Schopenhauer

Die Idee des Langweilometers ist schnell erklärt: Per Webinterface melden Teilnehmer einer Veranstaltung ihren aktuellen Eindruck. Dies geschieht durch Eingabe eines prozentualen Langeweilepegels zwischen Null („Super spannend und unterhaltsam!“) und 100 („Schlaftablette!“). Die Webseite meldet den aktuellen Stand nach der Eingabe grafisch zurück. Damit nicht jede Eingabe zur totalen Schwankung der Anzeige führt, zeigt das Langweilometer den Medianwert über die letzten elf Eingaben an (die Dämpfung lässt sich auch auf mehr als elf Eingaben erweitern). Außerdem gibt es eine analoge Ausgabe des Messpegelstandes.

Als Webserver kommt ein Raspberry Pi zum Einsatz, der klein und kostengünstig ist [1]. Er steuert die analoge Anzeige und alles andere über seine digitale Schnittstelle, das *General Purpose Input Output* (GPIO). Neben den typischen Schnittstellen zeichnet sich der kleine Computer durch eine Steckerleiste aus, die Digitalwerte einliest und ausgibt. Gibt er auf einer Leitung ein Signal aus, so sind dort entweder 3,3 Volt messbar (logisch *1*) oder null Volt (logisch *0*). Bei der Eingabe ist das im Prinzip genauso: Liegt dort eine Spannung von mehr als zwei Volt an, ist das eingelesene Zeichen eine logische 1, liegt die Eingangsspannung unter 0,8 Volt, erhält man eine logische 0. Legt man mehr als 3,3 Volt an einen Pin an, muss man einen neuen Raspberry Pi kaufen.

Messpegel-Anzeige in Hardware basteln

Da nur derjenige Veranstaltungsteilnehmer das Webinterface sieht, der eine Bewertung abgibt, soll noch eine externe Anzeige hinzukommen. Die Wahl fiel hier nicht auf das übliche Display oder eine Siebensegment-Ziffernanzeige, sondern es sollte etwas Analoges sein.

Glücklicherweise stand ein altes Voltmeter aus dem Messtechniklabor zur Verfügung. Es sitzt in einem etwa 40 Zentimeter hohen Gehäuse aus Holz und Glas, wie es früher im Physikunterricht üblich war. Ein riesiges Drehspulinstrument bewegt den Zeiger, der auch aus der Entfernung noch erkennbar ist. Das Gehäuse bietet nach dem Ausbau der Vorwiderstände und des Messbereichs-Wahlschalters noch Platz für den Raspberry Pi und vermittelt gleichzeitig einen Hauch von Steampunk (bis auf etwas zuwenig Messing am Gehäuse).

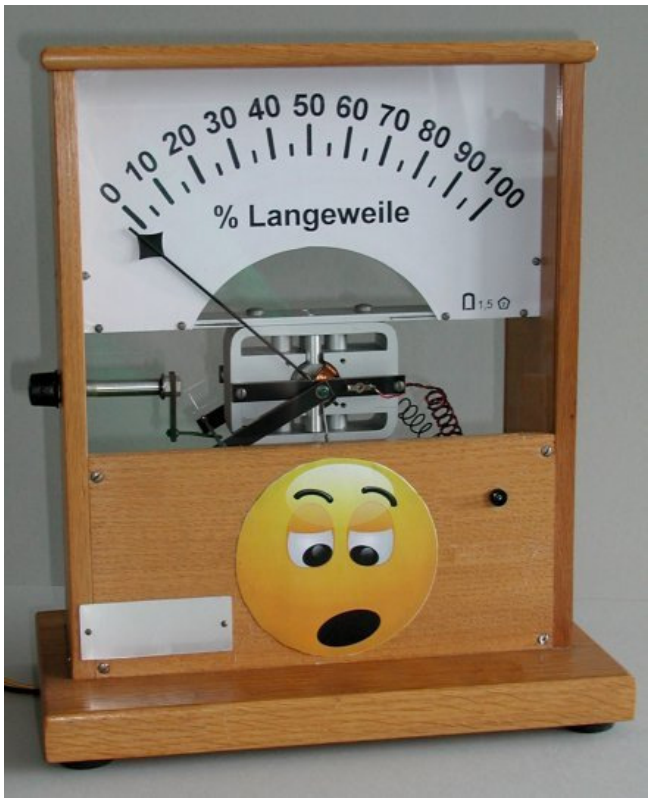


Abbildung 1: Ansicht des umgebauten Messinstruments von vorn.

Die zweite Idee, per Schrittmotor und Laserpointer einen Punkt auf einer an die Wand ge-

malten Skala zu markieren, scheitert an der Machbarkeit: Das Bemalen einer Wand im Foyer findet doch eventuell nicht ganz die Zustimmung der Chefs. Die dritte Idee, als Skala ein Plexiglasrohr zu nehmen, das bis zum entsprechenden Anzeige-Pegel mit gefärbtem Wasser gefüllt wird, scheidet ebenfalls als zu kompliziert aus. Also bleibt nur das Messgerät. Wer noch ein altes Spiegelgalvanometer auftreibt, kann auch das verwenden. Hauptsache groß und imposant.

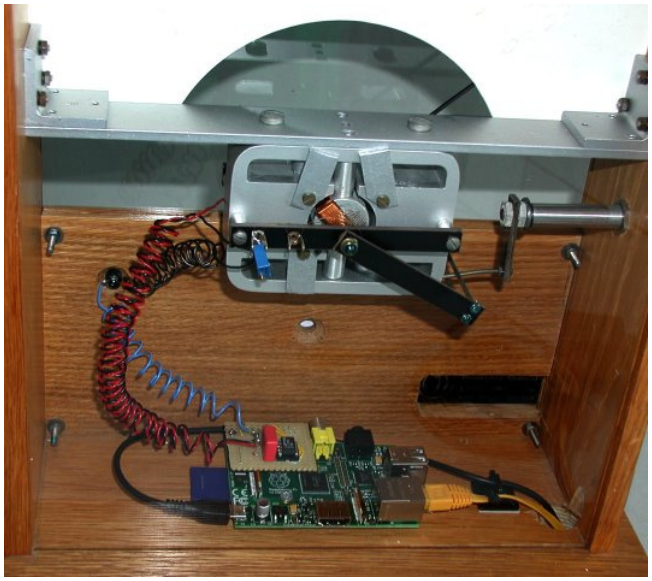


Abbildung 2: Einbau des Raspberry Pi in das Messgerät.

Als erstes ersetzt eine Prozent-Skala die Volt-Skala. Dann gehören die Bohrungen abgedeckt und das Typenschild umgedreht (dahinter war leider ein großes Loch für Sicherungen). Die große Skala des Drehschalters für die Messbereiche wird entfernt und durch ein Smiley ersetzt. Abbildung 1 zeigt das Ergebnis dieser Vorarbeiten.

Der Raspberry Pi und eine kleine Zusatzplatine für den Digital-Analog-Wandler finden wunderbar im Gehäuse Platz. Für das Stromversorgungskabel und das Netzwerkkabel kommt ein Loch in den Boden des Messgeräts. Für die Netzwerkverbindung eignet sich ein dünnes und flaches Kabel, das besonders flexibel ist. Solche Kabel lassen sich sogar unter dem Teppich verlegen.

Der Blick durch die hintere Scheibe zeigt nun unten den Raspi und das Messwerk (Abbildung 2). Der blaue Knubbel links am Messgerät ist ein Vorwiderstand (Mehrgangpoti), mit dem sich der Vollausschlag des Messinstruments auf die maximalen 3,3 Volt des Digital-Analog-Wandlers einstellen lässt. Das blaue Kabel führt zu einer ebenfalls blauen LED, die in die Öffnung des ehemaligen Umschalters gewandert ist. Sie zeigt nicht nur an, ob der Raspberry noch lebt, sondern auch, wann ein neuer Datenwert eintrifft. Die Spiralform der Kabel hat nur einen optischen, keinen elektrischen Zweck.

Analog-Interface für den Raspberry Pi

Der Raspberry Pi verfügt leider über keine analogen Ein- und Ausgänge, und der Ausgang für ein pulsweit moduliertes Signal (PWM) verträgt sich irgendwie nicht mit dem relativ niederohmigen Messwerk. Also musste eine kleine Schaltung für einen Digital-Analog-Wandler her. Keine Angst, die Schaltung in Abbildung 3 ist so einfach, dass sie auch von jemandem zusammengebraten werden kann, der mit dem Lötkolben ungeübt ist. Man sollte nur nicht zu lange an einen Pin herumlöten.

Die Anode der LED (positiver Pol, längerer Anschlussdraht) an der Gerätefront ist direkt über einen 120-Ohm-Vorwiderstand an GPIO 27 angeschlossen. Die Kathode (kürzerer Anschlussdraht) ist mit Masse verbunden. Der Wandlerbaustein ist auf einer Lochrasterplatte verdrahtet. Auf der Unterseite der Lochrasterplatte sitzt eine doppelreihige Buchsenleiste, die auf den GPIO-Anschluss des Raspberry Pi passt. Dann kann man die wenigen Verbindungen löten und die Platine auf den Raspi stecken.

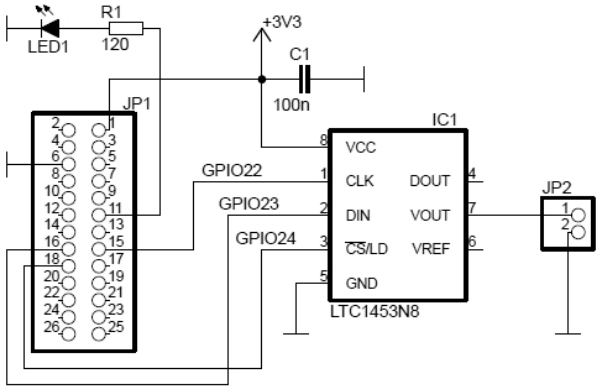


Abbildung 3: Die Schaltung des Digital-Analog-Wandlers.

Als Digital-Analog-Wandler dient der Baustein *LTC1453* von *Linear Technology* im achtpoligen, lötfreundlichen *DIL*-Gehäuse (Dual In-Line) [2]. Der Chip ist für den 3,3-Volt-Betrieb ausgelegt und hat eine 12 Bit Auflösung, was für die geplante Anwendung mehr als ausreichend ist. Auf den ersten Blick schien der LTC1453 ein ganz normaler *SPI*-Chip zu sein (Serial Peripheral Interface) – also null Probleme bei der Ansteuerung, denn der Raspberry hat schon ein passendes Treibermodul. Leider stellt sich heraus, dass es sich um einen nur

fast SPI-kompatiblen Chip handelt. Dazu später mehr.

Das SPI ist ein von *Motorola* entwickeltes System für einen synchronen, seriellen Datenbus. Er verbindet digitale Schaltungen nach dem Master-Slave-Prinzip. Im Grunde wandern die Daten von einem Takt gesteuert Bit für Bit vom Controller in den Peripheriebaustein und umgekehrt. Insofern ist SPI ein entfernter Verwandter der guten, alten seriellen Schnittstelle – nur ohne Start- und Stop-Bit, und mit getrennter Taktleitung. Das SPI besteht aus drei gemeinsamen Leitungen:

- SCLK (Serial Clock) – Takt, den der Master erzeugt.
- MOSI (Master Output, Slave Input) – dient der bitweisen Datenübertragung zum Peripheriebaustein.
- MISO (Master Input, Slave Output) – empfängt die Bits vom Peripheriebaustein

Damit sich mehrere Slaves ansprechen lassen, verfügt jeder Baustein über einen digitalen *Select*-Eingang, der über Null oder Eins den Chip aktiviert oder deaktiviert. Da beim Digital-Analog-Wandler nur Daten zum Peripheriebaustein wandern, sind sogar nur zwei Leitungen (und natürlich Chip-Selects) nötig.

Wie angedeutet, outet sich der LTC1453 als nur fast-SPI, denn er enthält einen Hardware-Bug. Bug und Workaround sind im Datenblatt versteckt [3]. Dort steht der kleine Satz: „Note: CLK must be low before CS/LD is pulled low to avoid an extra internal clock pulse. “ Übersetzt heißt das:

- Den Chip-Select-Pin CSLD auf 1 setzen, dann elf (!) Bit ganz normal eintakten (positiver Impuls am Takteingang CLK)
- Beim letzten Bit den Takteingang CLK auf 1 setzen, dann den CSLD-Pin auf 0 und danach CLK auf 0.

`lwm_setdac.c`: Pin aktivieren

Die Funktion `GPIO_Export()` aktiviert einen Pin des GPIO, indem man die Nummer des gewünschten GPIO in die Pseudovariable */sys/class/gpio/export* schreibt (Zeile 08):

```
01 int GPIO_Export(int pin)
02  {
03   char buffer[MAXBUF]; /* Output Buffer */
04   ssize_t bytes; /* Datensatzlaenge */
05   int fd;  /* Filedescriptor */
06   int res;  /* Ergebnis von write */
07
08   fd = open("/sys/class/gpio/export",
09             O_WRONLY);
10  if (fd < 0)
11     {
12     perror("Fehler bei open (export)!\n");
13     return(-1);
14     }
15   bytes = snprintf(buffer, MAXBUF, "%d", pin);
16   res = write(fd, buffer, bytes);
17   if (res < 0)
18     {
19     perror("Kann Pin nicht aktivieren!\n");
20     return(-1);
21     }
22   close(fd);
23   return(0);
24  }
```

Genauso lässt sich das GPIO wieder deaktivieren. Nur heißt das Ziel in Zeile 08 dann `/sys/class/gpio/unexport`. Dafür gibt es die Funktion `GPIO_Unexport()`, die aus dem Listing ensteht, in dem man den Funktionsnamen in Zeile 01 entsprechend ändert und `export` in Zeile 12 durch `unexport` ersetzt.

Also steht kein geeigneter Gerätetreiber zur Verfügung. Die Bits sind von Hand in den LTC1453 zu schieben. Nachdem im hier berichteten Fall der Baustein schon verlötet war, entstand die Software eben ohne SPI-Treiber, was auch nicht weiter schlimm war.

`lwm_setdac.c`: Analogwert auf AD-Wandlerbaustein bringen

```
01 void SetData(unsigned int value)
02   {
03   int i;
04  /* CS muss anfangs LOW sein */
05   GPIO_Write(CS, LOW);
06  /* shift MSB ... LSB (MSB zuerst) */
07   for(i = 11; i >= 0; i--)
08     {
09    /* set data bit, check MSB  */
10     if (value & 0x800)
11       GPIO_Write(DIN, HIGH);
12     else
13       GPIO_Write(DIN, LOW);
14     value = value << 1;
15    /* CLK-Impuls senden und bei Bit 0 CS/LD-Impuls auf HIGH */
16     GPIO_Write(CLK, HIGH);
17     if (i == 0) GPIO_Write(CS, HIGH);
18     GPIO_Write(CLK, LOW);
19     }
20   GPIO_Write(CS, LOW);
21  }
```

Die Funktion `void SetData()` schiebt den Analogwert als 12-Bit-Zahl bitweise in den LTC1453-Baustein, wobei das MSB (*most significant bit*, höchstwertiges Bit) als erstes kommt (Zeilen 06/07).

`lwm_setdac.c`: Analogwert setzen ...

Das C-Programm *lwm_setdac.c* ([4], [5]) setzt einen Analogwert über den Baustein *LTC1453*. Nach getaner Arbeit macht das Programm mittels einer `delay`-Funktion eine kleine Pause von 0,2 Sekunden, damit auch ein amok laufendes Skript (oder ein DAU) nicht zu wild mit dem Gerätezeiger wedelt. Die Funktion rechnet Millisekunden in Sekunden und Nanosekunden um und ruft dann die Library-Fnktion `nanosleep()` auf.

Der einzige Kommandozeilenparameter des Programms ist ein Prozentwert (`int`) zwischen null und 100. Damit taugt das Programm nicht nur für das Langweilometer, sondern auch für andere Anwendungen oder analoge Anzeigezwecke. Auch ein Aufruf per Shellskript ist möglich: Der Aufruf erfolgt dann zum Beispiel als `./lwm_setdac 45` für eine Anzeige von 45 Prozent.

Das Kompilieren erfolgt mit dem Kommando:

```
gcc -Wall -o lwm_setdac lwm_setdac.c
```

Damit unprivilegierte User (wie *pi* oder *www-data*) das Programm aufrufen können, muss es auf die SUID *root* lauten – oder man gibt den GPIO generell für den entsprechenden User frei:

```
01 sudo chown root.root lwm_setdac
02 sudo chmod 4711 lwm_setdac
```

Das Programm definiert folgende Konstanten, die für die Kommunikation mit dem GPIO notwendig sind:

```
01 /* Datenrichtung */
02 #define IN  0
03 #define OUT 1
04
05 /* Binärdatenwerte */
06 #define LOW  0
07 #define HIGH 1
08
09 /* großzügig bemessene */
10 /* maximale Datenpuffergröße */
11 /* fuer die GPIO-Funktionen */
12 #define MAXBUF 100
13
14 /* Namen der LTC1453-Pins */
15 #define CLK 22 /* Clock Input */
16 #define DIN 23 /* Data Input  */
17 #define CS 24 /* Enable */
```

Kern des Ganzen ist die Funktion `void SetData()`. Sie schiebt den Analogwert als 12-Bit-Zahl bitweise in den LTC1453-Baustein, wobei das höchstwertige Bit als erstes kommt (siehe Kasten „`void SetData()`: Analogwert auf AD-Wandlerbaustein bringen“).

Beim letzten Bit ist in dieser Funktion ab Zeile 10 der Workaround für den Hardware-Bug er-

sichtlich. Da wir es mit einer 12-Bit-Zahl zu tun haben, ermittelt eine *AND*-Verknüpfung mit `0x800` (= 1000 0000 0000) das höchstwertige Bit und setzt die Datenleitung zum LTC1453 dementsprechend auf `0` oder `1`. Danach wandert die Zahl um ein Bit nach links. Für den Takt wird bei den ersten elf Bits ein `1`-Impuls erzeugt, und beim letzten Bit die CS-Leitung vor dem Wechsel des Taktsignals auf `1` gezogen.

`lwm_setdac.c`: Datenrichtung festlegen

Die Funktion `Funktion GPIO_Direction()` legt die Datenrichtung eines GPIO-Pins fest. Der Wert `0` bedeutet, dass der GPIO ein Eingabepin ist, und `1` bedeutet Ausgabepin:

```
01 int GPIO_Direction(int pin, int dir)
02  {
03   char path[MAXBUF]; /* Buffer fuer Pfad */
04   int fd;  /* Filedescriptor */
05   int res;  /* Ergebnis von write */
06
07   snprintf(path,
08            MAXBUF,
09            "/sys/class/gpio/gpio%d/direction",
10            pin);
11   fd = open(path, O_WRONLY);
12   if (fd < 0)
13     {
14     perror("Fehler direction (open)!\n");
15     return(-1);
16     }
17   switch (dir)
18     {
19     case IN : res = write(fd,"in",2); break;
20     case OUT: res = write(fd,"out",3); break;
21     }
22   if (res < 0)
23     {
24     perror("Fehler direction (write)!\n");
25     return(-1);
26     }
27   close(fd);
28   return(0);
29  }
```

. . . und digital den Raspi ansteuern

Nach der Umwandlung spricht das C-Programm über das */sys*-Dateisystem die digitale Schnittstelle *General Purpose Input Output* (GPIO) des Raspberry Pi an. Die GPIO-Pins lassen sich damit direkt über ein System virtueller Dateien steuern. Auch hier gilt, dass dies nur mit Root-Rechten gelingt. Schreiben oder Lesen auf die virtuellen Dateien aktiviert über den Linux-Treiber zunächst den entsprechenden Port-Pin und stellt die Datenrichtung ein. Danach spricht Lesen oder Schreiben dann den Port an.

Die folgenden Funktionen dienen dem Ein- und Ausschalten der Ports und dem Schreiben auf eine Portleitung. Bei allen Funktionen bedeutet der Rückgabewert `0`, dass die Operation erfolgreich war, und `-1`, dass ein Fehler aufgetreten ist.

Die erste Funktion `GPIO_Export()` aktiviert einen Pin des GPIO. Dies geschieht, indem man die Nummer des gewünschten GPIO – nicht etwa die Pinnummer der Steckerleiste! – in die Pseudovariable */sys/class/gpio/export* schreibt (siehe Kasten „`lwm_setdac.c`: Pin aktivieren“).

Durch den Export entsteht ein neues Verzeichnis mit dem Namen `gpio`*XX*. In diesem Verzeichnis befinden sich unter anderem zwei Dateien namens *direction* und *value*. Diese Dateien dienen zum Ansteuern des GPIO-Pins. *direction* legt die Datenrichtung fest, also die Ein- oder Ausgabe. *value* dient zum Lesen und Schreiben von `0` oder `1`.

Die zweite Funktion `GPIO_Direction()` legt

die Datenrichtung eines GPIO-Pins fest (siehe Kasten „`lwm_setdac.c`: Datenrichtung festlegen“). Ist ein GPIO-Pin als Ausgabe geschaltet, erzeugt das Schreiben von Null oder Eins am entsprechenden Pin einen Pegel von 0 Volt bzw. 3,3 Volt. Die dritte wichtige Funktion `GPIO_Write()` setzt schließlich einen GPIO-Pin (siehe Kasten „`lwm_setdac.c`: Pin setzen“).

`lwm_setdac.c`: Pin setzen

Die Funktion `GPIO_Write()` setzt einen GPIO-Pin, nachdem `GPIO_Direction()` die Datenrichtung festgelegt hat. Als Werte (Parameter `value`, Zeile 09) kommen nur `LOW` und `HIGH` zum Einsatz:

```
01 int GPIO_Write(int pin, int value)
02  {
03   char path[MAXBUF]; /* Buffer fuer Pfad */
04   int fd; /* Filedescriptor */
05   int res; /* Ergebnis von write */
06
07   snprintf(path,
08            MAXBUF,
09            "/sys/class/gpio/gpio%d/value",
10            pin);
11   fd = open(path, O_WRONLY);
12   if (fd < 0)
13     {
14     perror("Fehler GPIO_Write (open)!\n");
15     return(-1);
16     }
17   switch (value)
18     {
19     case LOW : res = write(fd,"0",1); break;
20     case HIGH: res = write(fd,"1",1); break;
21     }
22   if (res < 0)
23     {
24     perror("Fehler GPIO_Write (write)!\n");
25     return(-1);
26     }
27   close(fd);
28   return(0);
29  }
```

Das komplette C-Programm vereint alle Funktionen. Der Vorspann mit seinen Header-Dateien und den Konstanten ist eigentlich für alle Programme verwendbar, die auf die Ein- und Ausgabe-Ports des Raspberry Pi zugreifen. Lediglich die Namen der Pins ändern sich. Die Hauptfunktion *main()* prüft zu Beginn, ob jemand mit Root-Rechten das Programm aufgerufen hat. Wenn nicht, bricht es die Ausführung ab. Danach initialisiert es die benötigten GPIO-Ports und prüft den Kommandozeilenparameter.

Die eigentliche Arbeit zeigt der Ausschnitt im Kasten „`lwm_setdac.c`: Richtige Ganzzahlen errechnen“: Da der Digital-Analog-Wandler 12 Bit verarbeitet, muss der an den Wandler gesendete Wert zwischen Null und 4095 liegen. Der hereinkommende Wert liegt zwischen Null und Hundert – es geht ja um den prozentualen Langeweilepegel.

`lwm_frontled.c`: LED erzählt, was geschieht

Die LED an der Gerätefront ist eigentlich eine Notlösung. Nach dem Ausbau der Messgeräte-Innereien hinterlässt der Umschalter für Gleich- und Wechselspannung nämlich ein unschönes Loch, in das aber genau eine fünf-Millimeter-LED mit Fassung passt. Fehlt nur noch eine Aufgabe für die LED.

Bei Mikrocontrollern implementiert der Autor nun immer gern eine blinkende LED als so genannten Heartbeat, um zu sehen, ob der Controller noch am Leben ist. Beim Raspberry kommt

die Tatsache gelegen, dass Linux ein Multitasking-Betriebssystem ist. Die LED arbeitet also völlig unabhängig vom Webinterface oder anderen Programmen – getreu dem Motto: Ein Programm, eine Aufgabe, die aber richtig. Das Beispiel besitzt aber sogar zwei Aufgaben. Im Ruhezustand arbeitet die LED als Heartbeat und zeigt an, dass das System aktiv ist. Und wenn eine neue Eingabe über die Webseite eintrifft, flackert sie kurz und heftig auf.

`lwm_setdac.c`: Richtige Ganzzahlen errechnen

```
01  if (argc == 2)
02    {
03    /* String lesen, nach int umwandeln */
04     value = atoi(argv[1]);
05    /* Wertebereich 0 .. 100 */
06     if ((value >= 0) || (value <= 100))
07       {
08      /* Prozent nach 12-Bit-int wandeln */
09       value = (unsigned int)(40.95 * value);
10       SetData(value);
11       }
12     else
13       {
14       fprintf(stderr,
15       "Argument must be between 0 and 100!\n");
16       return 3;
17       }
19    }
```

Da der Digital-Analog-Wandler 12 Bit verarbeitet, muss der an den Wandler gesendete Wert zwischen Null und 4095 liegen. Der hereinkommende Wert liegt zwischen Null und Hundert – es geht ja um den prozentualen Langeweilepegel (vgl. Zeile 05/06). Also multipliziert Zeile 09 den von der Kommandozeile hereinkommenden Wert mit 40,95 – so erhält der Wandler den richtigen Ganzzahlenwert.

Für die LED gibt es auch wieder ein C-Programm: `lwm_frontled.c` [6]. Das Kompilieren erfolgt wieder ganz schlicht mit dem Kommando

```
gcc -Wall -o lwm_frontled lwm_frontled.c
```

Das Programm muss natürlich ständig laufen, sonst blinkt nix. Damit der Bootvorgang es aktiviert, steht es sinnvollerweise in der Datei */etc/rc.local*. Damit der Zeiger des Messinstruments nach dem Booten nicht bei Null stehen bleibt, kommt auch ein Aufruf von `lwm_setdac.c` in diese Datei. Somit müssen zwei Zeilen das `exit` in */etc/rc.local* ergänzen:

```
01 /home/pi/lwm_setdac 50 &
02 /home/pi/lwm_frontled &
```

Das Ampersand-Zeichen hinter den Kommandos ist wichtig, um die Programme in den Hintergrund zu legen. Sonst unterbrechen sie den Ablauf der Startskripte, weil die Ausführung bei *lwm_frontled* hängen bleibt.

Und Action!

Das Programm *lwm_frontled.c* [5] implementiert die zwei Aufgaben der LED folgendermaßen:

- Der Heartbeat lässt die LED für 50 Millisekunden aufleuchten. Darauf folgen 100 Millisekunden Pause, wieder 50 Millisekunden Aufleuchten, dann eine Pause von einer Sekunde.
- Erfolgt eine neue Eingabe über das Webinterface, blinkt die LED 50 mal mit je 50 Millisekunden Periodendauer. Sie flackert also für 2,5 Sekunden.

Der Programmcode ist relativ einfach. Die Delay-Funktion stammt aus *lwm_setdac.c*. Auch `lwm_frontled.c` ist für andere Aufgaben verwendbar – sofern es nur darum geht, auf die Änderung einer Datei zu reagieren. Der Vorspann ist der gleiche wie beim vorhergehenden Programm, lediglich die Definitionen des LED-Pins und der Datendatei sind unterschiedlich. Die LED selbst ist an GPIO 27 angeschlossen, das ist Pin 13 der Steckerleiste:

```
...
01 /* maximale Datenpuffergröße */
02 /* fuer die GPIO-Funktionen */
03 /* (großzügig bemessen) */
04 #define MAXBUFFER 100
05
06 /* LED-Pin, Stiftleiste Pin 13 */
07 #define LED 27
08
09 /* Datei für Werteübergabe */
10 #define FILENAME "/home/pi/boring"
...
```

lwm_frontled.c: Datei-Modifikationsdatum für LED-Flackern

```
01   for(;;)
02     {
03    /* Lesen des Modifikationsdatums */
04     stat(FILENAME, &attr);
05     epoc = attr.st_mtime;
06     if (epoc > oldepoc)
07 /* Datei geändert, Lightshow :-) */
08       {
09       for (i = 0; i < 50; i++)
10         {
11         GPIO_Write(LED, 1);
12         delay(50);
13         GPIO_Write(LED, 0);
14         delay(50);
15         }
16       oldepoc = epoc;
17       }
18     else
19 /* normaler Heartbeat: */
20 /* zweimal blinken, dann Pause */
21       {
22       GPIO_Write(LED, 1);
23       delay(50);
24       GPIO_Write(LED, 0);
25       delay(100);
26       GPIO_Write(LED, 1);
27       delay(50);
28       GPIO_Write(LED, 0);
29       delay(1000);
30       }
31     }
32   return 0; /* never reached */
33   }
```

Das Heartbeat-Programm merkt sich in den Zeilen 03 bis 05 das Dateidatum und fragt es mittels `stat()` nach jedem Herzschlag-Zyklus ab (Zeile 06). Ist das neue Dateidatum größer, ist die Datei geändert – die Lightshow beginnt (Zeilen 07 bis 17), und das Programm speichert das neue Datum (Zeile 15). Günstig ist, nicht Datum und Uhrzeit zu speichern, sondern die sogenannte Unix-Epoche, also die Anzahl der Sekunden, die seit dem 1.1.1970, null Uhr vergangen sind (Zeilen 04 und 21). Das ist nämlich ein monoton steigender 32-Bit-Wert, der sich ideal für den geplanten Zweck eignet.

Neu hinzu kommt eine Signal-Handler-Funktion für die Tastenkombination *[Strg]+[C]*. Sie räumt auf, wenn das Programm bei manuellem Aufruf von Hand wieder zu beenden ist und sitzt vor der Hauptfunktion:

```
01 void beenden(int dummy)
02   {
03   GPIO_Write(LED, 0);
04   GPIO_Unexport(LED);
05   delay(100);
06   printf("\nHasta la vista, Baby!\n");
07   exit(0);
08   }
```

Nach der Initialisierung begibt sich das Programm in eine Endlosschleife. Wichtig ist bei solchen Programmen, dass in jedem durchlaufenen Programmzweig irgendwo ein Delay auftaucht – sonst nimmt das Programm den Prozessor unnö-

tig stark in Beschlag.

Das Blinken ist einfach: Delay, LED an, Delay, LED aus. Wie erfährt aber das LED-Programm von einer Neueingabe über das Webinterface? Da die Realisierung über eine der zahlreichen Möglichkeiten der Prozesskommunikation zu aufwendig erscheint, tut es ein einfacher Trick: Jeder neue Wert, der vom Webinterface kommt, wandert ja in die Datei */home/pi/boring*. Dadurch ändert sich das Modifikationsdatum der Datei (siehe Kasten "`lwm_frontled.c`: Datei-Modifikationsdatum für LED-Flackern").

`lwm_index.php`: Einfügefunktion

```
01 function insert($newvalue, $datafile)
02   {
03   $lines = array();
04   if (is_readable($datafile))
05     { $lines = file($datafile); }
06   while (count($lines) < MAXDATA)
07     { $lines[] = 50; }
08   for ($i = MAXDATA - 1; $i > 0; $i--)
09     { $lines[$i] = $lines[$i-1]; }
10   $lines[0] = $newvalue;
11   if (is_writable($datafile))
12     {
13     $handle = @fopen($datafile, "w");
14     for ($i = 0; $i < count($lines); $i++)
15       { fwrite($handle,
16                trim($lines[$i]) . "\n"); }
17     fclose($handle);
18     }
19   }
```

Die Funktion `insert()` liest die Datei in ein Array (Zeile 03) und schiebt alle Datenwerte um eine Indexposition weiter: Der letzte und damit älteste Datenpunkt fällt weg (Zeile 08 und 09), der neue Wert wird zum ersten Wert (Zeile 10). Sie führt in den Zeilen 06 und 07 auch eine kleine Korrektur durch: Da am Anfang noch keine Werte vorhanden sind, füllt die Routine das Array gegebenenfalls mit 50-Prozent-Werten auf. Schließlich schreibt sie die Daten zurück in die Datei, wobei sie in den Zeilen 11 bis 18 gleich noch Leerzeichen, Tabs und Newlines entfernt:

`lwm_index.php`: Das Webinterface

Für das Webinterface *lwm_index.php* [7] bilden – neben den notwendigen Webseiten-Geschichten wie Header, Body, Eingabeformular und Anzeige – zwei Funktionen den Kern der Langweilometer-Anwendung. Es handelt sich um eine Einfügefunktion für neue Daten sowie eine Funktion, welche die letzten n Eingaben mittelt. Sinn von Letzterem ist, dass nicht wegen einer einzigen Bewertung der Zeiger extrem ausschlägt.

Wie bei C, ermöglicht auch bei PHP die Definition von Konstanten leichte Anpassung. Das Programm legt den Binary-Pfad, eine Datendatei und die Anzahl Daten in der Datendatei fest:

```
01 define("SETDAC", "/home/pi/lwm_setdac");
02 define("DATAFILE", "/home/pi/boring");
03 define("MAXDATA", "11");
```

Die Einfügefunktion `insert()` soll die Datendatei bei der Datenspeicherung einerseits nicht zu groß werden lassen, andererseits Aktualität widerspiegeln. Daher speichert sie Daten rollierend: Wenn einen neues Datum hinzukommt, löscht sie das älteste. Dieser Algorithmus ist ein einfacher Ringpuffer, realisiert auf einem Array (siehe Kasten „`lwm_index.php`: Einfügefunktion“).

Wie lange das Ganze gut geht, hängt davon ab, wie viele Schreibzyklen die SD-Karte verkraftet. Dieser Wert hängt von der Qualität der SD-Karte ab – ein Chip-Speicher ist halt doch kein vollwertiger Ersatz für eine Festplatte. Deshalb ist zu empfehlen, sobald alles läuft, die SD-Karte in eine Image-Datei auf dem PC oder Laptop zu kopieren. Dann beschränkt sich die Wartungsarbeit beim Versagen der SD-Karte auf das Kopieren des Images auf eine neue SD-Karte.

Bei der zweiten Funktion hat der Autor zum Glätten der Werte mit zwei Verfahren experimen-

tiert. Zum einen bot sich wegen der sowieso schon rollierenden Datei der gleitende Mittelwert an: Da beim Eintreffen eines neuen Werts jeweils der älteste wegfällt, ergibt sich immer ein Mittel über die letzten n Werte. Dieses Verfahren läuft zwar zufriedenstellen, lässt sich aber von Ausreißern beeinflussen.

`lwm_index.php`: Median-Berechnungsfunktion

```
01 function calculate_median($datafile)
02   {
03   $lines = array();
04   if (is_readable($datafile))
05     { $lines = file($datafile); }
06   while (count($lines) < MAXDATA)
07     { $lines[] = 50; }
08   $aktual = $lines[0];
09   for ($i = 0; $i < MAXDATA; $i++)
10     { $lines[$i] = trim($lines[$i]); }
11   sort($lines);
12   $min = $lines[0];
13   $max = $lines[MAXDATA-1];
14   if(MAXDATA % 2 == 0)
15     {
16     $mean = ($lines[(MAXDATA/2) - 1]
17              + $lines[MAXDATA/2])/2;
18     }
19   else
20     {
21     $mean = $lines[MAXDATA/2];
22     }
23   return array ($aktual, $mean, $min, $max);
24   }
```

Die Funktion `calculate_median()` hat als Parameter den Namen der Datendatei. Sie liest wieder zunächst die Datei in ein Array (Zeile 03), füllt die Daten gegebenenfalls mit 50-Prozent-Werten auf (Zeilen 06 bis 08) und schmeißt Leerzeichen weg (Zeilen 09 und 10). Anschließend sortiert sie die Werte aufsteigend und holt sich die Maxima (Zeilen 11 und 12). Eine Wenn-Funktion unterscheidet: Ist die Zahl der Werte ungerade, berechnet sie den Median als das arithmetische Mittel der beiden mittleren Zahlen (Zeilen 16 und 17). Andernfalls, bei einer ungeraden Zahl der Werte, gilt der wert in der Mitte als Meridian (Zeile 21). Schließlich gibt die Funktion eine Liste von vier Werten zurück – den aktuellen Wert, das Minimum, das Maximum und den Median (Zeile 23).

Resistenter gegen Ausreißer ist der Median. Im Gegensatz zum Mittelwert teilt der Median die Wertemenge in zwei gleich große Teile auf. Daher eignet er sich bei dem Verdacht auf Ausreißer. Für den Betrachter ergibt sich lediglich der Nachteil, dass seine Eingabe nicht immer sofort eine Änderung der Anzeige zur Folge hat, wie es bei gleitenden Durchschnitt normalerweise geschieht. Der Median lässt sich so bestimmen (siehe Kasten „`lwm_index.php`: Median-Berechnungsfunktion“):

- Alle Werte aufsteigend sortieren.
- Ist die Anzahl der Werte gerade, gilt als Median das arithmetische Mittel der beiden mittleren Werte.
- Ist die Anzahl der Werte ungerade, ist die mittlere Zahl der Median.

Der Rest des PHP-Programms ist unspektakulär. Einige Funktionen versuchen, unsinnige Daten bei der Eingabe auszufiltern, andere melden sich, welcher Langeweilepegel gerade erreicht ist (nicht jeder kann ja das Instrument sehen). Die Anzeige ist als waagerechter Balken realisiert, dessen Breite das CSS an den aktuellen Prozentwert anpasst, und der seine Farbe von grün über gelb, orange und rot nach violett ändert (Abbildung 4).

Damit nun nicht Hinz und Kunz in aller Welt, sondern nur aktuell Betroffene einen Wert eingeben, untersucht das Programm die IP-Adresse des Webclient. Es dürfen nur diejenigen eine Wertung

abgeben, deren Client aus dem internen Netz des Veranstaltungsortes kommen. Im Anwendungsfall des Autoren war das das Netz der Fakultät, dessen IP-Adressen mit `10.27.` beginnen. Alle anderen, die zum Beispiel per Wlan im Netz herumgeistern, müssen das geheime Passwort kennen, das als SHA1-Hashwert im Programm gespeichert ist:

```
01 $Client = $_SERVER['REMOTE_ADDR'];
02 if (preg_match('/^10\.27\./', $Client))
03   { $Passwort = $PW; }
```

Schließlich setzt PHP die Instrumentenanzeige per System-Call:

```
01 $cmd = SETDAC . ' ' . $Mittel;
02 system($cmd, $retval);
```

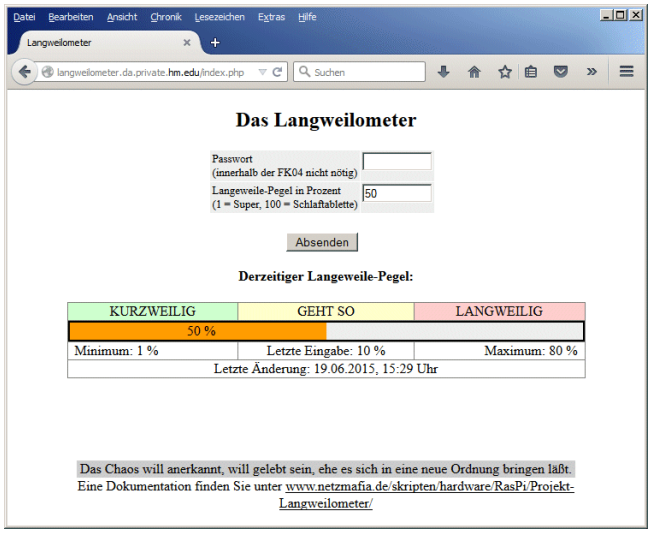


Abbildung 4: Das Webinterface mit Eingabemaske und Langweilometer-Balken.

Epilog

Dieser Artikel ist ein Versuchsballon. Die Mehrheit der UpTimes-Leser sind vermutlich eher dem Software- oder Administrationsbereich zuzurechnen und mit Hardware das letzte Mal möglicherweise vor mehr als 25 Jahren beim Zusammenschrauben eines PCs in Berührung gekommen. Daher kann es gut sein, dass dieser Artikel vollkommen neben der Zielgruppe liegt – noch dazu, weil das Thema auch nicht allzu ernsthaft ist.

Aber auch Claude Shannon, Mathematiker und Begründer der Informationstheorie, hat kleine mechanische Clowns gebastelt. Außerdem keimt derzeit ja das zarte Pflänzchen *Bastler* rund um preiswerte Boards wie Arduino oder Raspberry Pi. (Der Autor mag übrigens den neudeutschen Begriff *Maker*, der heute oft dafür verwendet wird, überhaupt nicht.)

Die Qualität dessen, was da so im Netz, aber auch in Zeitschriften zu finden ist, schwankt extrem. In den 1980er Jahren rekrutierten sich viele Computerbastler noch aus den Radiobastlern, Funkamateuren, HiFi-Enthusiasten und Modellbauern. Ohm, Watt und Volt waren bekannte Begriffe, der Übergang zum Computer nicht so schwer. Die Älteren unter uns erinnern sich vielleicht, dass die ersten Mikrocomputer wie *Altair* oder *Apple 1* anfangs als Bausatz erhältlich waren.

Der Autor stellt sich einfach die Frage, ob von dem geneigten Leser weitere Artikel zum Raspberry Pi gewünscht sind, und ob damit verbundene Hardware-Basteleien eher interessieren oder eher abschrecken. Eine kurze Mail an <***@netzmafia.de> mit dem Betreff *UpTimes* wäre schön. Auch Anregungen sind willkommen. *(Anm. d. Red.: Ja, die E-Mail-Adresse stimmt so und funktioniert, grinst der Autor auf unsere verwunderte Nachfrage, verweisend auf [8]. :-))*

Links

[1] Webserver auf dem Raspberry Pi einrichten:
http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_http.html
[2] Digital-Analg-Umwandler *LTC1453*: http://www.linear.com/product/LTC1453
[3] Datenblatt des LTC1453 mit Hardware-Bug: http://www.linear.com/docs/3265
[4] Quelltext *lwm_setdac.c* für den Digital-Analog-Umwandler:
http://www.netzmafia.de/skripten/hardware/RasPi/Projekt-Langweilometer/lwm_setdac.c
[5] Dokumentation "GPIO beim Raspberry Pi per Programm ansteuern":
http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_GPIO_C.html
[6] Quelltext *lwm_frontled.c* für die LED:
http://www.netzmafia.de/skripten/hardware/RasPi/Projekt-Langweilometer/lwm_frontled.c
[7] Quelltext *lwm_index.php* für das Webinterface:
http://www.netzmafia.de/skripten/hardware/RasPi/Projekt-Langweilometer/lwm_index.php.txt
[8] Jürgen Plate: Shellskripte mit Aha-Effekt; Erlaubte Zeichen in E-Mail-Adressen. In: UpTimes 2013-02, S. 22: https://www.guug.de/uptimes/2013-2/index.html

Über Jürgen

Jürgen Plate ist Professor für Elektro- und Informationstechnik an der Hochschule München. Er beschäftigt sich seit 1980 mit Datenfernübertragung und war, bevor der Internetanschluss für Privatpersonen möglich wurde, in der Mailboxszene aktiv. Unter anderem hat er eine der ersten öffentlichen Mailboxen – TEDAS der *mc*-Redaktion – programmiert und 1984 in Betrieb genommen.

Shellskripte mit Aha-Effekt V
Sirenengeheul und Viertelstundentöne aus der Shell

Was bisher geschah: Die Winterausgabe 02-2014 der UpTimes stellte allerhand Spielereien mit dem CLI-Sound-Exchanger *SoX* vor. Zwei haben wir noch. Und der in *Shellskripte mit Aha-Effekt* VI erbastelte Shell-Shuffle-Soundserver ist auch wieder im Spiel.

von **Jürgen Plate**

Töne sind höhere Worte.

Robert Schumann

Mit dem Kommandozeilentool SoX lassen sich für Messzwecke unter anderem Wav-Dateien mit weißem Rauschen erzeugen [1]. Folgendes Shellskript hingegen eignet sich zum Aufwecken der Teilnehmer einer Power-Point-Präsentation – nicht umsonst haben Power-Point und Power-Nap den gleichen Wortanfang.

Beim Seefunk geht einem Notruf eine 30 Sekunden lange Alarmsequenz voran, sodass die nachfolgende Meldung die nötige Aufmerksamkeit erlangt – so ähnlich, wie die Sirenentöne bei Odysseus. Unsere neuzeitliche Alarmsequenz besteht nun aus zwei Tönen von 1300 und 2100 Hertz, die sich mit einer Rate von 4 Hertz abwechseln. Das folgende Bash-Skript erzeugt eine Wav-Datei mit dieser Alarmsequenz.

Sirenengeheul mit SoX

Es generiert in den Zeilen 04 bis 07 die Töne mit einer Rate von 8 kHz in zwei temporäre Dateien im Raw-Format. Sie dauern jeweils 0,25 Sekunden, was der Wechselfrequenz von 4 Hertz entspricht. Diese Töne kopiert die For-Schleife der Zeilen 09 bis 12 60 mal hintereinander in eine weitere Datei. Da die beiden Töne hintereinander eine halbe Sekunde dauern, ergibt sich die Gesamtdauer von 30 Sekunden. Das funktioniert einwandfrei, da die Dateien nur Sound-Samples sind und keinerlei Verwaltungsinformationen enthalten. Zum Schluss macht das Programm eine Wav-Datei aus den Raw-Daten (Zeile 14) und räumt hinter sich auf (Zeile 15).

```
01 #!/bin/bash
02 $SOX = '/usr/local/bin/sox';
03
04 $SOX -U -r 8000 -n -t raw - synth 0.25 \
05   sine 1300 gain -3 > t.ul
06 $SOX -U -r 8000 -n -t raw - synth 0.25 \
07   sine 2100 gain -3 >> t.ul
08
09 for i in $(seq 1 60)
10  do
11  cat t.ul >> alert.ul
12  done
13
14 $SOX -c 1 -r 8000 alert.ul alert.wav
15
16 rm t.ul alert.ul
```

Wer sich gar nicht darum schert, was für ein Sound entsteht, probiert mal aus, etwas aus eindeutig nicht audiophilen Dateien herauszuholen. Es ist dann nur notwendig, SoX etwas anzulügen und mittels `-t raw` zu behaupten, es kämen Raw-Daten. Dass aber jetzt bloß niemand auf die Idee kommt, den Output von Tcpdump zu Musik zu machen, oder alle bisher bekannten Stellen von Pi in den SoX einzufüttern:

```
cat /dev/urandom |\
 sox -s -b -t raw -r 44100 - ausgabe.ogg

ls -lR / |\
 sox -s -b -t raw -r 44100 - ausgabe.ogg

cat masterarbeit.tex |\
 sox -s -b -t raw -r 44100 - ausgabe.ogg
```

SoX als Glockenturm

Das folgende SoX-Beispiel ist ein Skript, das in der Datei `/etc/crontab` alle Viertelstunde gestartet wird. Es liefert – etwa mangels nahe gelegenem Kirchturm – den Stunden- und Viertelstundenschlag. Grundlage ist der im Shell-Shuffle-Artikel gebaute Sound-Server [2]. SoX dient als Abspielprogramm:

```
01 #!/bin/bash
02
03 BELL1=/usr/local/big-bell.wav
04 BELL2=/usr/local/small-bell.wav
05
06 PLAY=/usr/bin/play
07 export SOX_OPTS="-v0.5 -V0"
08
```

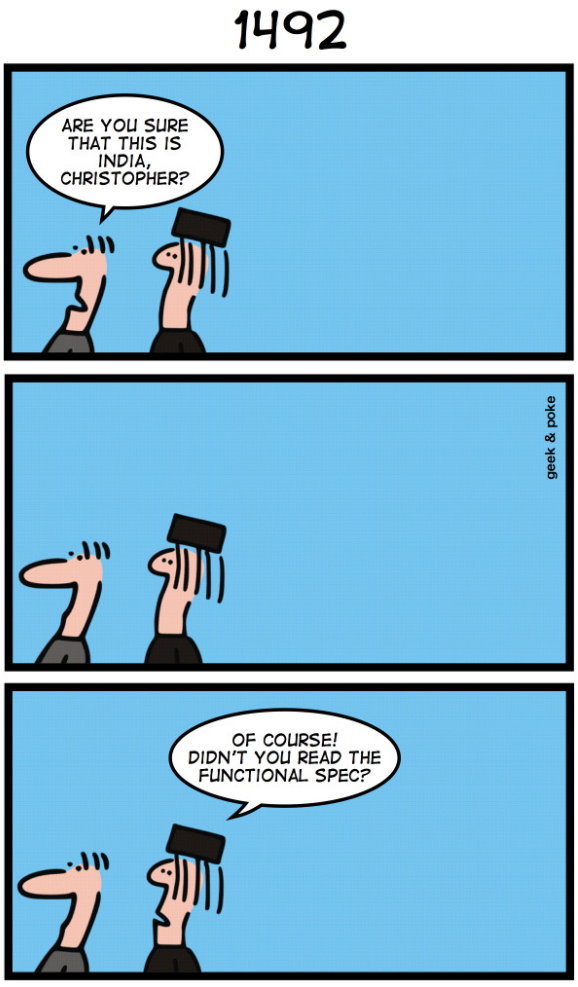
```
09 MINUTE=$(expr $(date "+%M") + 0)
10 HOUR=$(expr $(date "+%I") + 0)
11
12 if [ $MINUTE -eq 0 ]; then
13  BELLS="$BELL2 $BELL2 $BELL2 $BELL2"
14  while [ $HOUR != 0 ]
15    do
16    BELLS="$BELLS $BELL1"
17     HOUR=$(expr $HOUR - 1)
18     done
19  $PLAY $BELLS
20
21 elif [ $MINUTE -eq 15 ]
22  then  $PLAY $BELL2
23 elif [ $MINUTE -eq 30 ]
25   then  $PLAY $BELL2 $BELL2
26 elif [ $MINUTE -eq 45 ]
27   then  $PLAY $BELL2 $BELL2 $BELL2
28 fi
```

Zeilen 03 und 04 speichern die Sounddateien für Stunden und Viertelstunden. In den Zeilen Zeilen 06 und 07 gibt es eine kleine Neuerung gegenüber den anderen Beispielen: Diesmal speichert eine Shell-Variable die Kommandozeilenoptionen für SoX, was nicht nur die Kommandos kürzer macht, sondern auch Änderungen erleichtert.

Anschließend ermittelt das `date`-Kommando die aktuelle Minute und Stunde. Weil es führende Nullen liefert, wird der Zahlenstring durch Addieren von 0 zu einer ganzen Zahl normiert (Zeilen 09 und 10). Der Rest des Skripts baut die Stundenschläge zur vollen (Zeilen 12 bis 20) und zur viertel Stunde zusammen (Zeilen 22 bis 27). Im Ergebnis wird entsprechend der Stunden- oder Minutenzahl gebimmelt.

Links

[1] Jürgen Plate, Mixpult per Tastatur: Der CLI-Sound-Exchanger SoX. In: UpTimes 02-2014, S. 30
http://www.guug.de/uptimes/2014-2/
[2] Jürgen Plate, Shellskripte mit Aha-Effekt IV: Ein Interface für das Shell-Shuffle. In: Ebd, S. 36.



Change Management
Interview zur Umstellung auf DevOps

Das Ops-Team von System Engineer Henning Henkel ist dabei, auf DevOps umzustellen. Was das genau heißt und wie die Erfahrungen sind, beantwortet er hier.

Interview: **Anika Kehrer**

Verschieden gestimmte Saiten ergeben erst Harmonie.

Joseph von Eichendorff

Henning, was genau ist Deine Position bei Deinem Arbeitgeber?
Henning: Ich arbeite als Senior System Engineer in der Konzern-IT der Basler Versicherung. Mein derzeitiges Team wurde früher, zu Zeiten von Solaris, als 'Unix-Team, bezeichnet. Über die Zeit hat sich unsere Rolle gewandelt, und wir heißen inzwischen 'Application-Services-Team'. Wir decken ein weites Feld von Anwendungen ab, engineeren und betreiben mehr oder weniger alles von Java-Applikationsservern über Monitoring und Logging bis hin zu SAP. Seit etwa einem Jahr ist das Engineering und der Betrieb der Linux-Plattform ebenfalls Bestandteil unserer Aufgaben.

Was sind Deine Aufgaben?
Henning: Ich gehöre zum Middleware-Team innerhalb des Application-Services-Teams. Einer meiner bisherigen Schwerpunkte war Engineering und Betrieb von Weblogic-Servern. Seit Linux auch Bestandteil unserer Team-Aufgaben ist, hat sich meine Rolle verschoben, und eine meiner Hauptaufgaben ist jetzt die Entwicklung einer zukunftsfähigen Linux-Plattform.

„DevOps hat viel mit Kommunikation und Kultur zu tun"

Du und Dein Team seid gerade im Prozess, Euch hin zu DevOps zu entwickeln. Seit wann genau?
Henning: Einen genauen Termin kann ich nicht nennen. Es war mehr eine Entwicklung über die Zeit, in der in unserem Team die Erkenntnis gereift ist, dass wir wohl darauf zu steuern.
Was bedeutet ‚DevOps'?
Henning: Das ist ein schwer zu fassender Begriff. Nach meinem Verständnis ist es ein Annähern der Software-Entwickler (Dev) und der Personen, die die Systeme betreiben (Ops). Es gibt immer wieder Stellenangebote für DevOps-Positionen, aber diese gibt es nach meinem Verständnis gar nicht. Für mich hat DevOps viel mit Kommunikation, Kultur und Vorgehensmodellen zu tun. Unter DevOps kann man sicherlich auch eine Industrialisierung der Infrastrukturbereitstellung verstehen, analog zur Industrialisierung der Produktion im 19. Jahrhundert.
Woher kommt Deiner Meinung nach diese Entwicklung?
Henning: Zwei Entwicklungen kommen zusammen: Software-Entwickler greifen vermehrt zu agilen Vorgehensmodellen wie Scrum oder Kanban und den damit verbundenen kürzeren Entwicklungszyklen. Dazu kommt der Trend zur Automatisierung, um Continuous Integration oder sogar Continuous Deployment umzusetzen. Beides zusammen ergibt höhere Anforderungen an die Bereitstellung der Infrastruktur. Schließlich wecken Dienste wie Amazon Web Services oder Microsoft Azure das Bedürfnis, ähnliche Dienstleistungen in der internen IT anzubieten.

Den Trend zur Automatisierung nehme ich seit einigen Jahren wahr: Vielfach soll die gleiche Anzahl Personen mehr Systeme betreuen. Das spiegelt sich meiner Meinung nach in der Popularität von Konfigurationsmanagement-Systemen wie *Puppet*, *Chef*, *Ansible* oder *SaltStack*. Die Einführung von DevOps ist hier einfach ein weiterer Schritt, indem zum Beispiel Scrum für Infrastruktur-Projekte zum Einsatz kommt, oder CI-Lösungen automatisiert Konfigurationen der Infrastruktur testen. Meiner Wahrnehmung nach nähern sich Devs und Ops hier immer mehr an.
Wie kam es konkret bei Euch dazu, das DevOps-Modell umzusetzen?
Henning: Als Linux im Oktober 2013 als Thema zu uns ins Team kam, hatten wir lediglich die Idee, mit mehr Automatisierung der gestiegenen Anzahl Systeme Herr zu werden. Im Januar 2014

hatten wir dann einen Austausch auf technischer Ebene mit einer Firma, die schon viel Automatisierung umgesetzt hatte. Durch die sehr interessanten Gespräche und anschließenden internen Diskussionen merkten wir, dass wir nicht einfach nur ein Tool einführen können, sondern auch ein Kulturwandel nötig wird. Dies ist der Tatsache geschuldet, dass sich die Arbeitsweise bei konsequenter Automatisierung stark verändert. So zögerten wir anfangs zum Beispiel, unser Automatisierungsvorhaben den internen Kunden gegenüber zu erwähnen. Wir befürchteten, sie würden uns mit Anforderungen überrennen, sodass wir Systeme ausliefern müssten, bevor wir sie als fertig ansehen. Das ist natürlich nicht Sinn der Sache.

Was es nutzt

Erscheint Dir rückblickend als nützlich, die DevOps-Herangehensweise auszuprobieren?
Henning: Ich denke, strukturiertes und standardisiertes Vorgehen hilft, eine große Anzahl an Systemen zu pflegen. Auch der Ansatz, die Kommunikation zwischen Devs und Ops zu verbessern, ist sehr gut, da das frühzeitig Missverständnisse ausräumt und Verständnis für das Gegenüber entstehen lässt.
Wenn Du Dir mal verschiedene Situationen vorstellst, wann wäre sie eher nicht nützlich?
Henning: Man sollte nicht versuchen, alles blind umsetzen zu wollen. Dennoch kann meiner Meinung nach jedes Unternehmen von strukturiertem Vorgehen profitieren. Wenn man keine eigene Software-Entwicklung im Unternehmen hat, wird man sich an die Release-Zyklen des Software-Herstellers halten müssen. Die Kommunikation gestaltet sich dann sicherlich schwieriger. Doch niemand hält einen davon ab, beispielsweise einen internen Lifecycle für die Kaufsoftware zu definieren.
Wie habt zum Beispiel Ihr einen internen Lifecycle erstellt?
Henning: Wir definierten für uns intern, dass die erste Release unserer Linux-Installation im Januar 2015 bereitstehen soll. Wir setzten einen Backlog auf mit Features, die wir bis zur Release gebaut und automatisiert haben möchten. Weiter legten wir fest, dass wir ungefähr sechs Monate nach der ersten Release eine weitere zur Verfügung stellen möchten. Diese zweite Release wird Fixes und weitere Automatisierungen enthalten. Falls zu diesem Zeitpunkt verfügbar, wird sie dann zum Beispiel auf RHEL 7.1 aufbauen.
Du meinst also, das Vorgehen eignet sich für jeden?
Henning: Bei kleinen Unternehmen mit geringer Anzahl von Servern muss man sicherlich schauen, dass das Kosten-Nutzen-Verhältnis im vernünftigen Rahmen bleibt. Dann gilt es, nur Dinge umzusetzen, die den größten Nutzen bringen. Doch gerade in diesem Fall ist es meiner Meinung nach wichtig, den Zeitpunkt der Einführung von Automatisierung nicht zu verpassen.

Was dabei hilft

Erfordert das DevOps-Vorgehen neue oder modifizierte Qualifikationen?
Henning: Die Analogie der Industrialisierung von IT führt auch hier weiter. Ich halte es für sehr wahrscheinlich, dass auf lange Sicht höher qualifizierte Mitarbeitende nötig werden. Ich denke, dass es in Zukunft nicht ausreicht zu wissen, wie man etwas manuell aufsetzt. Sondern es wird wichtig sein, dies zu automatisieren. Bei Configuration-Management-Systemen wie Puppet wird gern von *Infrastructure as a Code* gesprochen. Aus diesem Grund wird in Zukunft notwendig sein, mehr Software-Entwicklungs-Knowhow in Infrastruktur-Abteilungen zu haben.
Siehst Du Problempunkte bei diesem Verfahren?
Henning: Der Begriff ‚DevOps' ist unscharf definiert, aber aktuell sehr hip. Meiner Beobachtung nach gibt es bereits Vorreiter. Im Mainstream angekommen ist es aber noch nicht. Das hat zur Folge, dass es keine Best Practices gibt und das Abwägen der Möglichkeiten viel Zeit benötigt. Ich habe allerdings die Hoffnung, dass sich das schnell ändert, da immer mehr Firmen tätig werden. So baut meines Wissens nach die Firma Red Hat bei mehreren Kunden ähnliche Umgebungen auf.
Was hilft einem DevOps-Team?
Henning: Es hilft, wenn Kolleginnen und Kollegen in der Vergangenheit erste Erfahrungen mit Software-Entwicklung gemacht haben. Auch Erfahrung mit agilen Arbeitsmethoden entschärfen die Situation. In meinem Fall unterstützen uns unsere Entwicklern sehr gut, welche schon seit Jahren nach Scrum vorgehen. Man sieht an diesem Beispiel, dass DevOps erste Früchte trägt.
Erzähle bitte mal von dem DevOps-Summit, zu dem Ihr extra hingefahren seid – wie es Euch nicht nur in der Sache etwas gebracht hat, sondern als Nebeneffekt vor allem die Kommunikation zwischen Devs und Ops förderte …

Henning:Zwei Entwickler, mein Teamleiter und ich hatten die Möglichkeit, am DevOps-Summit in Amsterdam teilzunehmen. Es gab dort einzelne interessante Vorträge, über die wir abends diskutierten. Zudem diskutierten wir über die speziellen Situationen in unserem Unternehmen. Rückblickend haben wir dann festgestellt, dass uns dieser zwanglose Austausch am meisten gebracht hat. In der Folge gab es einen gemeinsamen internen Vortrag vor Publikum aus Dev und Ops. Da dieses Thema nur durch Annähern der Software-Entwickler (Dev) und der Personen, die die Systeme betreiben (Ops) bearbeitet werden kann, wurde ein weiteres Treffen für interessierte Entwickler und Betriebsangehörige organisiert. Dabei wurden dann konkrete Ideen und Bedürfnisse ausgetauscht.

Über Henning

Henning Henkel hat Computer Networking an der Hochschule Furtwangen studiert und dort sein Studium als Diplom-Informatiker (FH) abgeschlossen. Er beschäftigt sich schon seit über zehn Jahren mit Linux und Open-Source-Software. Bei der Basler Versicherung war er die ersten Jahre im Bereich Middleware tätig. Seit eineinhalb Jahren ist er für die Automatisierung der Linux-Plattform sowie der Einführung von agilen Arbeitsmethoden in diesem Bereich verantwortlich.

Angriffsvektor Stromnetz
Buchbesprechung: Blackout

Der spannende und gut recherchierte Thriller erzählt von einem Totalstromausfall in Europa, der über zwei Wochen dauert.

von **Hartmut Streppel**

Die ewigen Sterne
kommen wieder zum Vorschein,
sobald es finster genug ist.

Thomas Carlyle

Abbildung 1: Marc Elsberg:
Blackout; Morgen ist es zu spät. (Roman)
München/Blanvalet 2012.
Taschenbuch: 800 Seiten, 10 Euro, ISBN 978-3-442-38029-9.
E-Book: 9 Euro, ISBN 978-3-641-07431-9. Hörbuch (ungekürzt): 30 Euro, München/Random House Audio, ISBN 978-3-8371-1393-8.

Auf dem BCI-Kongress 2014 fand ich einen Flyer, der auf eine Lesung in München hinwies: Blackout (Stromausfall) hieß das Buch, von dem ich bis dahin nichts gehört hatte. Einige Wochen später bei einem Training zum selben Thema in Hamburg stand das Buch im Regal des Schulungsraums, und ich warf einige Blicke hinein. Da mich das Thema *Business Continuity* mit all seinen Facetten sehr interessiert und das Buch packend geschrieben war, kaufte ich es – und hatte es einige Tage später trotz seiner fast 800 Seiten ausgelesen.

Marc Elsberg, Österreicher (damit entfällt die Frage, ob man sich das Original besorgt), beschreibt in seinem Buch die Entwicklung Europas in den Tagen nach einem totalen Stromausfall, der auch nach über zwei Wochen noch nicht behoben ist. Die Folgen sind zunächst gering: ein kleines Verkehrschaos hier, ein kleiner Versorgungsengpass dort. Nach einigen Tagen aber herrscht in ganz Europa ein katastrophales Chaos. Elsberg beschreibt in zwei parallelen, teilweise miteinander verwobenen Erzählsträngen die Auswirkungen auf die Gesellschaften und die Infrastruktur und den Versuch der Behörden, den Angriff zu verstehen und die Ordnung wieder herzustellen.

Die Geschichte

Einem italienischen Informatiker und Aktivisten fallen bei einem Stromausfall einige seltsame Meldungen im Display des intelligenten Stromzählers seiner Wohnung auf. Nachdem kurze Zeit später das Stromnetz komplett zusammengebrochen ist, beginnt er zu recherchieren. Ihm fällt auf, dass der Ausfall in Italien und Schweden begann, den beiden Ländern, in denen Smart Meter am weitesten verbreitet sind. Aus den Meldungen, die er auf seinen Stromzählern gesehen hat, schließt er, dass hier womöglich ein koordinierter Angriff auf die Stromnetze und Kraftwerke vorliegt.

Die meisten Infrastruktur-Einheiten funktionieren nicht ohne Strom: Tankstellen, Kernkraftwerke, Wasserversorgung, Krankenhäuser. Die wich-

tigen haben zwar eine Notstromversorgung, aber die reicht nur ein bis zwei Tage. Wenn dann der Strom nicht wieder da ist, geht fast nichts mehr. Auch die Gesellschaftsstrukturen, heute vollständig abhängig vom Strom, brechen teilweise zusammen. Die humanen, westeuropäischen Gesellschaften zeigen sich dann von ihrer hässlichsten Seite.

Der italienische Aktivist versucht, die Behörden, von seinem Verdacht zu überzeugen, die natürlich zunächst nur lokal denken. Er bietet seine Hilfe an. Das ist ein langwieriger Prozess, der nur gelingt, weil er private Verbindungen in höchste Ebenen in Brüssel hat. Und weil man ihm, der natürlich überwacht wurde, wegen seiner politischen Vergangenheit nicht so richtig traut, gerät er unter Verdacht, mit den Urhebern der vermuteten Anschläge unter einer Decke zu stecken. Während er dann in einer behördlichen Hochsicherheitsumgebung mit seinem Laptop am Netz ist, wird dieser von außen gekapert, und es werden ihm verdächtige Daten und Emails untergeschoben. Zum Schluss werden die Angreifer enttarnt, weil sie in ihrer – natürlich auch überwachten – Kommunikation etwas unvorsichtig sind.

Reflexion

Eine Frage, die sich mir als Informatiker bald stellte: Warum sind Kraftwerksbetreiber nicht in der Lage, zumindest die nicht beschädigten Kraftwerke wieder ans Netz zu bringen? Hier wirkt in der Geschichte der langfristige und umfassende Plan der Angreifer: Die Steuerungssoftware ist infiziert, die bei fast allen Kraftwerken von demselben Hersteller kommt. Die Schadinhalte wurden dann von außen aktiviert. Die Software liefert dem Betrieb also einfach falsche Daten für die überwachten Komponenten. Und die Experten in den Kraftwerken, die auf Grund jahrelanger Erfahrungen im Betrieb in der Lage sind, den Status ihrer Turbinen auch ohne Software zu bestimmen, folgen den Anweisungen der infizierten Steuerungssoftware.

Blackout ist ein sehr spannendes und lehrreiches Buch, nicht nur für die, die sich mit dem Thema Business Continuity auseinandersetzen. Obwohl das Buch fiktiv ist, scheinen viele Ereignisse gar nicht mehr so unwahrscheinlich, nachdem wir durch Stuxnet und Fukushima gelernt haben, wie verwundbar moderne Gesellschaften sind. Elsberg schreibt in seinem Nachwort, dass er sogar einige technische Details weggelassen habe, um nicht möglichen Nachahmern zuviel Informationen zu geben.

Vielleicht lässt die Lektüre ja den einen oder anderen IT-Vorstand etwas differenzierter darüber nachdenken, ob er die Business-Continuity-Komponenten im letzten Angebot seines IT-Lieferanten aus Kostengründen herausstreicht. Und vielleicht fällt dem Verantwortlichen für das Notstromaggregat ja auf, dass der zum Betrieb notwendige Treibstoff nur für zwei Tage reicht, und im Katastrophenfall Nachschub sehr schwer zu besorgen sein wird. Ich jedenfalls habe neuerdings 16 Liter Wasser im Vorratskeller neben dem Weinregal.

Über Hartmut

Hartmut Streppel ist Spezialist für die Themen Hochverfügbarkeit und Business Continuity. Er arbeitete zuletzt für die Firma Oracle, zu der er durch die Akquisition von Sun Microsystems gekommen war. In den 1990-er Jahren war er zwei Jahre lang Mitglied des Vorstands der GUUG.

Hilfreiches für alle Beteiligten
Autorenrichtlinien

Selbst etwas für die UpTimes schreiben? Aber ja! Als Thema ist willkommen, was ein GUUG-Mitglied interessiert und im Themenbereich der GUUG liegt. Was sonst noch zu beachten ist, steht in diesen Autorenrichtlinien.

Der Schriftsteller ragt zu den Sternen empor,
Mit ausgefranstem T-Shirt.
Er raunt seiner Zeit ihre Wonnen ins Ohr,
Mit ausgefranstem T-Shirt.

Frei nach Frank Wedekind, Die Schriftstellerhymne

Wir sind an Beiträgen interessiert. Wir, das ist diejenige Gruppe innerhalb der GUUG, die dafür sorgt, dass die UpTimes entsteht. Dieser Prozess steht jedem GUUG-Mitglied offen. Der Ort dafür ist die Mailingliste <redaktion@uptimes.de>.

Welche Themen und Beitragsarten kann ich einsenden?

Die UpTimes richtet sich als Vereinszeitschrift der GUUG an Leser, die sich meistens beruflich mit Computernetzwerken, IT-Sicherheit, Unix-Systemadministration und artverwandten Themen auseinandersetzen. Technologische Diskussionen, Methodenbeschreibungen und Einführungen in neue Themen sind für dieses Zielpublikum interessant, Basiswissen im Stil von *Einführung in die Bourne Shell* hingegen eher nicht. Wer sich nicht sicher ist, ob sein Thema für die UpTimes von Interesse ist, kann uns gern eine E-Mail an <redaktion@uptimes.de> schicken.

Neben Fachbeiträgen sind Berichte aus dem Vereinsleben, Buchrezensionen, Konferenzberichte, humoristische Formen und natürlich Leserbriefe interessant. Wer nicht gleich mehrseitige Artikel schreiben möchte, beginnt also mit einem kleineren Beitrag.

Fachbeiträge sind sachbezogen, verwenden fachsprachliches Vokabular und anspruchsvolle Erläuterungen, besitzen technische Tiefe und ggf. auch Exkurse. Berichte aus dem Vereinsleben greifen aktuelle Themen auf oder legen Gedankengänge rund um die GUUG und ihre Community dar. Konferenzberichte zeigen, welche Veranstaltungen jemand besucht hat, was er/sie dort erfahren hat und ob die Veranstaltung nach Meinung des Autoren beachtenswert oder verzichtbar war. Unterhaltsame Formen können ein Essay oder eine Glosse sein, aber auch Mischformen mit Fachartikeln (Beispiel: der „Winter-Krimi“ in Ausgabe 2013-3). Auch unterhaltsame Formen besitzen jedoch inhaltlichen Anspruch. Denn die UpTimes ist und bleibt die Mitgliederzeitschrift eines Fachvereines.

In der UpTimes legen wir daher auch Wert auf professionelle publizistische Gepflogenheiten und einheitliche Schreibweisen. Dafür sorgt zum Beispiel ein einheitliches Layout der Artikel, oder etwa die grundsätzliche Vermeidung von Worten in Großbuchstaben (entspricht typografisches Schreien) oder von Worten in Anführungsstrichen zum Zeichen der Uneigentlichkeit (entspricht Distanzierung von den eigenen Worten). Wichtig sind außerdem beispielsweise Quellenangaben bei Zitaten, Kenntlichmachung fremder Gedanken, Nachvollziehbarkeit der Argumentation sowie Informationen zum Autor nach dem Artikel.

In welchem Format soll ich meinen Artikel einsenden?

ASCII: Am liebsten blanke UTF8-Texte. Gern mit beschreibenden Anmerkungen oder Hinweisen (zum Beispiel mit Prozentzeichen zum Kenntlichmachen der Meta-Ebene).

LaTeX: Wir setzen die UpTimes mit LaTeX. Weil wir – wie es sich beim Publizieren gehört – mehrspaltig setzen und ein homogenes Erscheinungsbild anstreben, verwenden wir für die UpTimes bestimmte Formatierungen. Es ist nicht erwünscht, eigene Layoutanweisungen einzusenden. Wir behalten uns vor, Texte für die Veröffentlichung in der UpTimes umzuformatieren. Eine Vorlage mit den von uns verwendeten Auszeichnungen für Tabellen, Kästen und Abbildungen gibt es unter http://www.guug.de/uptimes/artikelvorlage.tex.

Listings: Der mehrspaltige Druck erlaubt maximal 45 Zeichen Breite für Code-Beispiele, inklusive 1 Leerzeichen und einem Zeichen für den Zeilenumbruch innerhalb einer Code-Zeile (Backslash). Breitere Listings formatieren wir um, verkleinern die Schriftgröße oder setzen sie als separate Abbildung.

Bilder: Wir verarbeiten gängige Bildformate, soweit ImageMagick sie verdaut und sie hochauflösend sind. Am besten eignen sich PNG- oder PDF-Bilddateien. Plant bei längeren Artikeln mit 1 Abbildung pro 3000 Zeichen. Das müssen nicht Bilder sein, sondern auch Tabellen, Listings oder ein Exkurskasten sind möglich. Verseht Eure Bilder nicht mit Rahmen oder Verzierungen, weil die Redaktion diese im UpTimes-Stil selbst vornimmt.

Wie lang kann mein Artikel sein?

Ein einseitiger Artikel hat mit zwei Zwischentiteln um die 2.700 Anschläge. Mit etwa 15.000 Anschlägen – inklusive 3 Abbildungen – landet man auf rund vier Seiten. Wir nehmen gern auch achtseitige Artikel, achten dabei aber darauf, dass der Zusammenhang erhalten bleibt und dass es genug Bilder gibt, damit keine Textwüsten enstehen.

Wer Interesse hat, für die UpTimes zu schreiben, macht sich am besten um die Zeichenzahl nicht so viele Gedanken – auch für kurze oder lange Formate finden wir einen Platz. Die Redaktion ist bei der konkreten Ideenentwicklung gern behilflich. Für eine Artikelidee an <redaktion@uptimes.de> reicht es, wenn Ihr ein bestimmtes Thema behandeln wollt.

Wohin mit meinem Manuskript?

Am einfachsten per E-Mail an <redaktion@uptimes.de> schicken. Das ist jederzeit möglich, spätestens jedoch vier Wochen vor dem Erscheinen der nächsten UpTimes. Zum Manuskript ist ein kleiner Infotext zum Autor wichtig, ein Bild wünschenswert.

Nützlich ist, wenn der Text vor Einsendung durch eine Rechtschreibkorrektur gelaufen ist. `aspell`, `ispell` oder `flyspell` für Textdateien sowie die von LibreOffice bieten sich an. Wenn Ihr Euren Text an die Redaktion schickt, solltet Ihr also weitestmöglich bereits auf die Rechtschreibung geachtet haben: Nach der Einschickung ist Rechtschreibung und Typo-Korrektur Aufgabe der Redaktion. Die Texte in der UpTimes folgen der neuen deutschen Rechtschreibung.

Wie verlaufen Redaktion und Satz?

Wir behalten uns vor, Texte für die Veröffentlichung in der UpTimes zu kürzen und zu redigieren. Das bedeutet, dafür zu sorgen, dass der Artikel nicht ausufert, versehentliche Leeraussagen wegfallen, Syntax und Satzanschlüsse geglättet werden, dass Passiva und Substantivierungen verringert und Unklarheiten beseitigt werden (die zum Beispiel Fragen offen lassen oder aus Passivkonstruktionen resultieren, ohne dass der Schreibende das merkt). Manchmal ist dieser Prozess mit Nachfragen an den Autoren verbunden.

Die endgültige Textversion geht jedem Autoren am Ende zur Kontrolle zu. Dabei geht es um die inhaltliche Kontrolle, ob sich durch den Redaktionsprozess Missverständnisse oder Falschaussagen entstanden sind. Danach setzt die Redaktion die Artikel. Wenn der Satz weitgehend gediehen ist – also ein *Release Candidate* als PDF vorliegt – erhalten die Autoren als erste diesen RC. Danach wird die UpTimes dann veröffentlicht.

Gibt es Rechtliches zu beachten?

Die Inhalte der UpTimes stehen ab Veröffentlichung unter der CC-BY-SA-Lizenz, damit jeder Leser die Artikel und Bilder bei Nennung der Quelle weiterverbreiten und auch weiterverarbeiten darf. Bei allen eingereichten Manuskripten gehen wir davon aus, dass der Autor sie selbst geschrieben hat und der UpTimes ein nichtexklusives, aber zeitlich und räumlich unbegrenztes Nutzungs- und Bearbeitungsrecht unter der CC-BY-SA einräumt.

Bei Fotos oder Abbildungen Dritter ist es rechtlich unabdingbar, dass der Autor sich bei dem Urheber die Erlaubnis zu dieser Nutzung einholt, und fragt, wie die Quelle genannt zu werden wünscht. Die Frage nach der CC-BY-SA ist hierbei besonders wichtig.

An Exklusivrechten, wie sie bei kommerziellen Fachzeitschriften üblich sind, hat die UpTimes kein Interesse. Es ist den Autoren freigestellt, ihre Artikel noch anderweitig nach Belieben zu veröffentlichen.

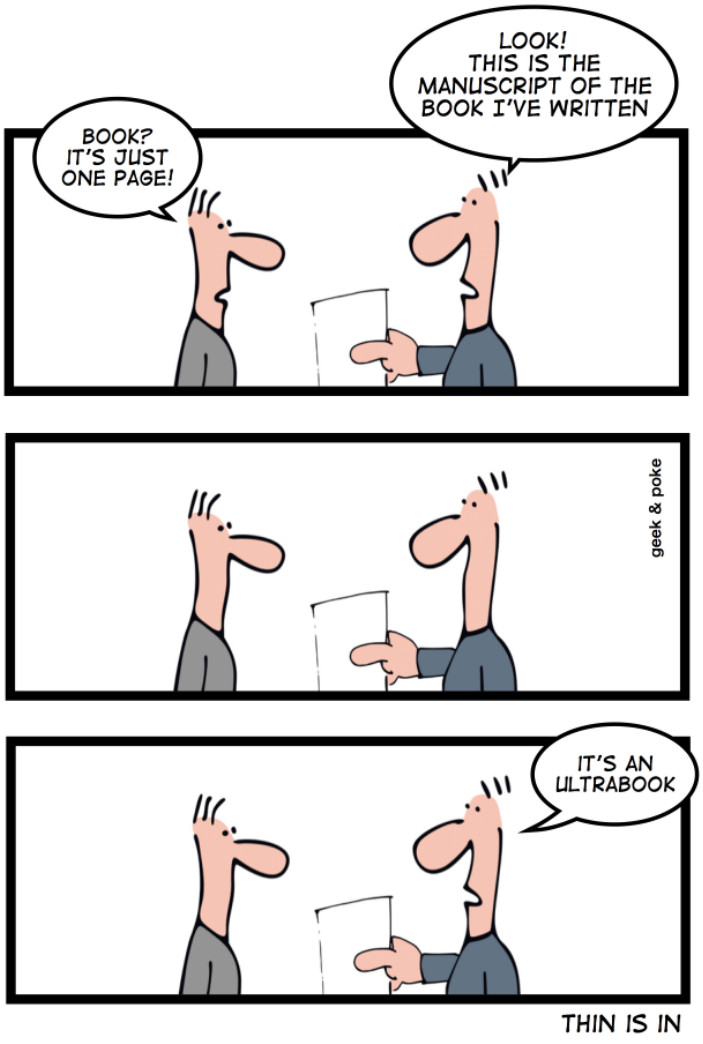
Bekomme ich ein Autorenhonorar?

Für Fach- und literarische Beiträge zahlt die GUUG dem Autor nach Aufforderung durch die Redaktion und Rechnungstellung durch den Autor pro Seite 50 € zuzüglich eventuell anfallender USt. Beiträge für die Rubrik „Vereinsleben“,

Buchrezensionen und Artikel bezahlter Redakteure sind davon ausgenommen. Gleiches gilt für Paper, wenn die UpTimes die Proceedings der Konferenz enthält.

Nächste redaktionelle Ausgabe:
UpTimes 2015-2, Winter-Ausgabe

- Redaktionsschluss: Montag, 2. November 2015.
- Erscheinung: Samstag, 5. Dezember 2015.
- Gesuchte Inhalte: Fachbeiträge über Unix und verwandte Themen, Veranstaltungsberichte, Rezensionen, Beiträge aus dem Vereinsleben.
- Manuskript-Template: http://www.guug.de/uptimes/artikel-vorlage.tex
- Fragen, Artikelideen und Manuskripte an: <kehrer@guug.de> oder an die Redaktionsmailingliste <redaktion@uptimes.de>



Über die GUUG
German Unix User Group e.V.

Vereinigung deutscher Unix-Benutzer

Die Vereinigung Deutscher Unix-Benutzer hat gegenwärtig rund 700 Mitglieder, davon etwa 90 Firmen und Institutionen.

Im Mittelpunkt der Aktivitäten der GUUG stehen Konferenzen. Ein großes viertägiges Event der GUUG hat eine besondere Tradition und fachliche Bedeutung: In der ersten Jahreshälfte treffen sich diejenigen, die ihren beruflichen Schwerpunkt im Bereich der IT-Sicherheit, der System- oder Netzwerkadministration haben, beim *GUUG-Frühjahrsfachgespräch* (FFG).

Seit Oktober 2002 erscheint mit der *Uptimes* – die Sie gerade lesen – eine Vereinszeitung. Seit 2012 erscheint die Uptimes einerseits zu jedem FFG in Form einer gedruckten Proceedings-Ausgabe (ISBN), und andererseits im Rest des Jahres als digitale Redaktionsausgabe (ISSN). Daneben erhalten GUUG-Mitglieder zur Zeit die Zeitschrift *LANline* aus dem Konradin-Verlag kostenlos im Rahmen ihrer Mitgliedschaft.

Schließlich gibt es noch eine Reihe regionaler Treffen (http://www.guug.de/lokal): im Rhein-Ruhr- und im Rhein-Main-Gebiet sowie in Berlin, Hamburg, Karlsruhe und München.

Warum GUUG-Mitglied werden?

Die GUUG setzt sich für eine lebendige und professionelle Weiterentwicklung im Open Source-Bereich und für alle Belange der System-, Netzwerkadministration und IT-Sicherheit ein. Wir freuen uns besonders über diejenigen, die bereit sind, sich aktiv in der GUUG zu engagieren. Da die Mitgliedschaft mit jährlichen Kosten

Fördermitglied	350 €
persönliches Mitglied	90 €
in der Ausbildung	30 €

verbunden ist, stellt sich die Frage, welche Vorteile damit verbunden sind?

Neben der Unterstützung der erwähnten Ziele der GUUG profitieren Mitglieder auch finanziell davon, insbesondere durch die ermäßigten Gebühren bei den Konferenzen der GUUG und denen anderer europäischer UUGs. Mitglieder bekommen außerdem *c't* und *iX* zum reduzierten Abopreis.

Wie GUUG-Mitglied werden?

Füllen Sie einfach das umseitige Anmeldeformular aus und schicken Sie es per Fax oder Post an die unten auf dem Formular angegebene Adresse. Falls Sie die Seite nicht herausreißen wollen: Sie können den Mitgliedsantrag als PDF herunterladen, siehe URL auf dem Mitgliedsantrag.

Impressum

Uptimes – Mitgliederzeitschrift der
German Unix User Group (GUUG) e.V.
Herausgeber: GUUG e.V.
Grube-Nassau-Straße 3
D-56462 Höhn
E-Mail: <redaktion@uptimes.de>
Internet: http://www.guug.de/uptimes/

Autoren dieser Ausgabe: Anika Kehrer, Kristian Köhntopp, Jens Link, Henri Wahl, Jürgen Plate, Hartmut Streppel
V.i.S.d.P: Martin Schulte, Vorstandsvorsitzender, Anschrift siehe Herausgeber
Chefredaktion: Anika Kehrer
Redaktion: Wolfgang Stief
LaTeX-Layout (PDF): Robin Schröder, Mathias Weidner
XHTML-Layout (ePub): Mathias Weidner
Titelbild und -gestaltung: Hella Breitkopf
Bild: Comicreihe *geek & poke* CC-BY-SA, mit freundlicher Genehmigung von Oliver Widder. Andere Quellennachweise am jeweiligen Bild.
Titelbild: Detail aus einem Lauf von *Memtest*, diese Version stellte allerdings Fehler fest, die nicht da waren (https://bugzilla.redhat.com/show_bug.cgi?id=805813). Der hier zu unrecht beschimpfte Speicher tut nun schon seit Jahren unbescholten seinen Dienst, unter anderem beim Erstellen von UpTimes-Titelbildern ;-).
Verlag: Lehmanns Media GmbH, Hardenbergstraße 5, 10623 Berlin
ISSN: 2195-0016

Wenn Sie Interesse an Anzeigen in der UpTimes haben,
wenden Sie sich bitte an <werbung@guug.de>.