| | |
|---|---|
| Prüfer: | Prof. Dr.-Ing. Roland Rühle |
| Betreuerin: | Barbara Burr |
| | |
| Begonnen am: | 10. Februar 1997 |
| Beendet am: | 8. August 1997 |
| | |
| CR-Klassifikation: | D.1.1, D.4.3, D.4.8, E.2, H.2.4, H.3.2, H.4.3 |

Studienarbeit Nr. 1633

# News in Dosen

Tobias Hennerich

# Zusammenfassung

Die in dieser Studienarbeit entwickelten und implementierten Methoden eines neuen Speichersystems für Newsartikel führen dazu, daß ein Newsserver bei entsprechender Modifizierung der Newsserver-Software INN elementare Performancesteigerungen erfahren kann:

- Das Einsortieren und Lesen von Artikeln beschleunigt sich um ca. Faktor 10.
- Der Expire benötigt zum Löschen der Artikel anstatt mehrerer Stunden nur noch wenige Minuten.
- Wegen der Performance-Steigerungen können mehrere Expires am Tag durchgeführt werden, was zu einer gleichmäßigeren Auslastung der Festplatten führt.
- Der Speicherbedarf pro Artikel geht deutlich, die Größe der History-Datei merklich zurück.
- Die Änderungen für die Administration des Newsservers sind relativ einfach.
- Das Verhalten des Newsservers bleibt für die User unverändert.
- Es existiert eine neue Library, mit welcher noch nicht angepaßte „Backends" einfach in das neue Speicherformat integriert werden können.
- Als „Abfallprodukt" entstand eine modifizierte Version des Programmes overchan, welche auch auf herkömmlichen INN-Systemen die Schreibperformance der Overview-Database verdoppelt.

Diese Arbeit ist folgendermaßen aufgebaut:

Im ersten Kapitel wird das Problem umrissen und der Lösungsvorschlag vorgestellt, ohne auf die genaue Implementierung näher einzugehen. Desweiteren werden verschiedene Lösungen gezeigt, die schon heute vorhanden sind.

Im zweiten Kapitel werden die Usenet-News allgemein erklärt, wobei speziell auf Eigenheiten eingegangen wird, die bei der späteren Implementierung berücksichtigt werden mußten.

Das dritte Kapitel stellt den INN in der Version 1.5.1 vor und erläutert die bisherige Funktionsweise und Implementierung verschiedener Features.

Die vorgenommenen Änderungen zur Implementierung des Lösungsvorschlages werden im vierten Kapitel näher erläutert.

Im 5 Kapitel folgt schlußendlich eine Analyse des gewonnenen Zahlenmaterials an Performance-Werten mit entsprechenden Interpretationsversuchen.

**1 EINFÜHRUNG** ..... 5
1.1 Themengebiet ..... 5
1.2 Geschichtliche Plazierung ..... 5
1.3 Problembeschreibung ..... 6
1.3.1 Vorteile ..... 6
1.3.2 Nachteile ..... 6
1.4 Lösungsvorschlag ..... 7
1.4.1 Vorteile ..... 7
1.4.2 Nachteile ..... 8
1.5 Vorhandene Lösungen ..... 9
1.5.1 DNEWS ..... 9
1.5.2 Diabolo ..... 10
1.5.3 Cyclone Newsrouter ..... 10
1.5.4 Zusammenfassung ..... 10

**2 USENET-NEWS** ..... 11
2.1 Namenskonventionen ..... 11
2.2 Aufbau eines News-Artikels ..... 11
2.2.1 Artikel-Header ..... 12
2.2.2 Besondere Artikel-Header ..... 12
2.3 Hierarchien der Newsgruppen ..... 14
2.4 Expire ..... 14
2.5 Crosspostings ..... 15
2.6 Spams ..... 15

**3 AUFBAU DES BISHERIGEN INN 1.5.1** ..... 17
3.1 Übersicht ..... 17
3.2 innd ..... 17
3.2.1 channels ..... 18
3.2.2 Die Datei active ..... 18
3.2.3 renumber ..... 18
3.2.4 Die Datei history ..... 19
3.3 frontends ..... 19
3.4 backends ..... 19
3.4.1 Die Konfigurationsdatei newsfeeds ..... 20
3.5 nnrpd ..... 22

**4 VORGENOMMENE ÄNDERUNGEN** ..... 23
4.1 Neue Speichertechnik der Artikel ..... 23
4.1.1 News in Dosen ..... 23
4.1.2 Lagerung der Dosen ..... 24
4.1.3 Datenformat der Dosen ..... 25
4.2 Einbindung der Dosen in den INN ..... 28
4.2.1 Zugriff auf die Dosen ..... 32
4.2.2 Lesen und Schreiben auf die Dosen ..... 33

4.2.3 Warum keine Channels? ..... 36
4.2.4 Das neue History-Format ..... 36
4.3 Die Backends ..... 37
4.3.1 overchan ..... 37
4.3.2 Batcher ..... 41
4.3.3 innxmit ..... 43
4.3.4 Die QIO-Library ..... 44
4.3.5 Die QCI-Library ..... 44
4.3.6 nnrpd ..... 45
4.4 Administrative Programme ..... 47
4.4.1 Expire ..... 47
4.4.2 Expire.rm bzw. fastrm ..... 47
4.4.3 Expireover ..... 47
4.4.4 news.daily ..... 48
4.4.5 makehistory ..... 48

**5 NEWS IN DOSEN IM PRAXISEINSATZ** ..... 49
5.1 news.bawue.de ..... 49
5.1.1 Die Testumgebung ..... 49
5.1.2 Die Usenet-Situation ..... 49
5.2 Newsaufkommen ..... 50
5.3 Einsortieren von Newsartikeln ..... 51
5.4 Batchen von Newsartikeln ..... 53
5.5 Weitere Statistiken ..... 55

**6 FAZIT** ..... 56

**7 LITERATURVERZEICHNIS** ..... 57

# 1 Einführung

## 1.1 Themengebiet

Usenet-News oder kurz News ist ein weltweites Konferenzsystem mit Diskussionsforen für Themen über Computer, Freizeit, Gesellschaft, Kultur, Kunst, Politik, Religion, Wirtschaft und Wissenschaft.

Es funktioniert ähnlich wie eine Zeitung, die nur aus Leserbriefen besteht. Jeder News-Leser kann selbst Nachrichten verfassen und dadurch Fragen anderer Teilnehmer beantworten, zu Diskussionen Stellung beziehen oder auch selbst bei Problemen die anderen Leser um Hilfe bitten. Die Nachrichten, die in einem hierarchischen, nach Themen sortierten Baum angeordnet sind, werden von mehreren hunderttausend Rechnern (größtenteils über das Internet) ausgetauscht und stehen Millionen von Lesern auf allen sieben Kontinenten (inkl. der Antarktis) zur Verfügung. Heute (Stand 1997) werden täglich hunderttausende Artikel in über 10000 verschiedenen Diskussionsforen geschrieben, ca. alle 15 Monate verdoppelte sich bis dato das Newsaufkommen.

## 1.2 Geschichtliche Plazierung

Das Usenet entstand im Jahr 1979, kurz nach dem Erscheinen von UNIX V7 mit UUCP[1] zwischen zwei Rechnern, welche zwei Studenten an der University of North Carolina verbanden. 1980 kam ein dritter Computer hinzu. Im Lauf der Jahre wurde die Software immer wieder umgeschrieben, um das dauernd steigende Newsaufkommen bewältigen zu können.

Aus reinen Shell-Scripten entwickelten sich Ansammlungen von C-Programmen. Die Programme wurden verbessert, nach und nach wurden zur Übertragung der Newsartikel nicht mehr Modems und das UUCP-Protokoll, sondern das Internet und NNTP[2] verwendet. Ein neues Softwarepaket entstand, welches die große Zahl von kleinen Programmen durch einen Daemon ersetzte, der alle Ein- und Ausgaben übernimmt. Dieses Programmpaket heißt INN und ist quasi auf allen größeren Newssystemen weltweit im Einsatz. Die zum Zeitpunkt der Studienarbeit offizielle Version war 1.5.1 mit diversen Security-Patches.

---

[1] UUCP: UNIX to UNIX CoPy, eine Protokoll zum Übertragen von Dateien zwischen UNIX-Rechnern, in der Regel per Modem über Telefon-Wählverbindungen.

[2] NNTP: Network News Transfer Protocol, ein Protokoll explizit zur Übertragung von Newsartikeln, welches sowohl zwischen Newsservern zum Artikelaustausch als auch zwischen Newsservern und Newsclients eingesetzt wird.

## 1.3 Problembeschreibung

Obwohl die Newsserver-Software von 1997 kaum noch etwas mit den Shellscripten von 1979 zu tun hat, gibt es doch noch eine elementare Gemeinsamkeit zu verzeichnen: *Jeden Newsartikel, den ein Newsserver speichert, wird als einzelne Datei im Dateisystem der Maschine abgelegt.* Der einzige Fortschritt, der in dieser Beziehung jemals gemacht wurde, war die Unterteilung des (zu Beginn einzigen) Artikeldirectories in einen Verzeichnisbaum, der die hierarchischen Newsgruppen abbildet.

### 1.3.1 Vorteile

- Es ist einfach und bis zu einem gewissen Newsaufkommen auch ausreichend schnell. Geoff Collyer schreibt bei der Vorstellung von C-News in seinem Artikel „News Need Not Be Slow", daß das alte B-News zum Expiren alle Newsartikel öffnen mußte, und nennt dabei außerdem die Zahl von „5000 oder noch mehr" Artikel, die damals auf einem Newssystem gleichzeitig verfügbar waren. Heute sind 5000 Artikel eine Zahl, die ein System innerhalb weniger Minuten angeboten werden können.
- Es ist skalierbar und auch bei sehr vielen Artikeln (mehrere Millionen) noch möglich, da relativ gut über mehrere Festplatten verteilbar. Es existiert allerdings das Problem, daß bei sog. Crosspostings (d.h. ein Newsartikel wird in mehreren Gruppen gleichzeitig gepostet) im Falle mehrerer Festplatten mit Symbolic Links gearbeitet werden muß, was zu Schwierigkeiten beim Expire führen kann, da diese Symbolic Links nicht auf schon gelöschte Artikel zeigen dürfen.
- Artikel können sehr schnell einzeln wieder gelöscht werden, auch kurzfristig können die Kriterien, welche Artikel zu löschen sind, problemlos geändert werden oder auch Artikel storniert werden (Cancels).
- Der Durchsatz an Nachrichten durch das Newssystem ist zu groß, als daß mit einer herkömmlichen Datenbank die Artikel verwaltet werden könnten. Datenbanken sind darauf optimiert, Daten sehr schnell abrufen zu können, nicht aber unbedingt innerhalb weniger Tagen den gesamten Datenbestand regelmäßig auszuwechseln.

Höchstwahrscheinlich ist der letzte Punkt wohl auch das Hauptproblem, weshalb sich die Datenhaltung per Datenbank bis jetzt nicht etabliert hat, obwohl das Konzept pro Artikel eine Datei auch eine Reihe von Nachteilen birgt:

### 1.3.2 Nachteile

- Das Anlegen von Dateien ist eine relativ langwierige Angelegenheit, da dafür an mehreren Stellen auf der Platte Zugriffe gemacht werden müssen (Eintrag im Directory, Anlegen eines Inodes, Belegen der eigentlichen Blöcke mit dem Artikelinhalt, markieren der Blöcke als reserviert).

- Da Newsartikel größtenteils sehr kurz sind (in der Regel unter 4 KB), muß relativ viel Verschnitt auf den Festplatten in Kauf genommen werden, da Dateien normalerweise Plattenplatz in ganzen Blöcken verbrauchen.
- Das Löschen von Artikeln ist ebenso eine aufwendige Angelegenheit, für die mehrere Plattenkopfbewegungen nötig sind. Bei großen Newssystemen benötigt der täglich stattfindende Expire nur wenige Minuten für das Auflisten aller zu löschenden Artikel, aber mehrere Stunden, bis tatsächlich alle Dateien von der Platte entfernt sind.

Für alle diese Nachteile wird hier nun folgender Lösungsvorschlag vorgestellt:

## 1.4 Lösungsvorschlag

Das eigentliche Problem sind die großen Datenmengen, die dauernd neu dazukommen und nach unterschiedlichen Zeiträumen wieder gelöscht werden müssen. Im Grunde sind aber alle Informationen, die entscheiden wann ein Artikel wieder gelöscht wird, schon bei seinem Eintreffen auf dem Newssystem bekannt: Newsgruppe, Erstellungsdatum und Ankunftsdatum, evtl. auch eine explizite Expire-Zeile im Header des Artikels. Diese Informationen kann man sich zunutze machen.

Als erstes muß von der klassischen Philosophie pro Artikel eine Datei Abschied genommen werden. Statt dessen werden alle Artikel, die zu einem bestimmten Datum gelöscht werden sollen, in einer einzigen großen Datei kompakt an einem Stück gesammelt. Normalerweise verwenden die News-Administratoren nur wenige verschiedene Zeitintervalle, welche Artikel auf dem System gehalten werden sollen. Nachdem 90% aller Artikel innerhalb weniger Stunden auf jedem beliebigen anderen Newsserver im Internet eintrifft, sind die Artikel fast alle in etwa gleich alt. Dadurch müssen die Artikel zu jedem Zeitpunkt in nur wenige verschiedene Dateien einsortiert werden.

Praktisch bedeutet dies, daß der Newsserver Artikel entgegennimmt und diese in 5-10 verschiedene Dateien ablegt, welche er parallel geöffnet hält. An anderer Stelle wird vermerkt, daß der Artikel 20438 aus der Newsgruppe news.software.b im File 96120618 ab Byte 103.765.289 abgelegt ist, welches am 6. Dezember 1996 um 18:00 Uhr expired, also gelöscht werden soll.

### 1.4.1 Vorteile

- Nachdem nicht mehr dauernd neue Dateien angelegt werden müssen, können Artikel sehr schnell und in großen Bündeln auf den Festplatten abgelegt werden. Ein moderner Rechner kann problemlos 3 MB/sec auf die Festplatte schreiben (was über 1000 Artikel entspräche), hat aber Mühe 100 neue Dateien pro Sekunde anzulegen.

- Da die Artikel alle in einer Datei gesammelt werden, fällt kein Verschnitt auf der Festplatte an, wodurch der zusätzliche Platzbedarf für das Abspeichern der Position des Artikels mehr als ausgeglichen wird.
- Der Expire ist nicht mehr eine Sache von Stunden, sondern erfordert nur das Löschen einiger weniger sehr großer Dateien.
- Da der Expire sehr schnell erledigt ist, können mehrmals am Tag immer wieder Artikel gelöscht werden und dadurch eine gleichmäßigere Auslastung der Festplatten erzielt werden.

### 1.4.2 Nachteile

- Das explizite Löschen einzelner Artikel schon vor der eigentlich vorgesehenen Zeit ist sehr aufwendig. Wenn nicht sehr große Mengen von Artikeln gelöscht werden müssen, wird es einfacher sein nur den entsprechenden Bereich als gelöscht zu markieren, den Speicherplatz aber nicht tatsächlich freizugeben. Für normale Cancels sollte dies ein brauchbarer Kompromiß sein.
- Eine dynamische Änderung des Expires ist nicht möglich bzw. hat erst nach mehreren Tagen eine Auswirkung. Man kann zwar kurzfristig eine noch gar nicht zum Löschen vorgesehene Datei vorzeitig freigeben, aber dadurch verschiebt man das Problem mit dem Plattenplatzmangel nur bis zu dem Zeitpunkt, wenn das vorzeitig gelöschte File dann wirklich gelöscht werden müßte.
- Statistiktools können dem Newsadministrator helfen schon frühzeitig Plattenengpässe zu prognostizieren und rechtzeitig die Expirezeiten abzuändern. Im Notfall ist es allerdings auch möglich, die ganzen Dateiblöcke umzuschichten und dadurch neue Expire-Zeiten einzuführen, dies bedingt aber eine gewisse Downtime des Newssystems. Diese Tools sind allerdings noch nicht entwikkelt worden.

## 1.5 Vorhandene Lösungen

Die hier beschriebene Idee ist schon mehrere Jahre alt und stammt vollständig vom Autor dieser Studienarbeit. Im Winter 1994/1995 existierten keine ernsthaften Newssysteme auf UNIX-Basis außer C-News in verschiedenen modifizierten Versionen sowie INN.

In letzter Zeit hat sich an diesem Szenario einiges geändert. Es wurden mehrere Newssysteme vorgestellt, die einen ähnlichen Ansatz wie der hier umgesetzte beinhalten.

Es soll hier nicht näher auf diese Systeme eingegangen werden, sondern nur kurz die verschiedenen Speichertechniken und ein paar allgemeine Informationen genannt werden.

### 1.5.1 DNEWS

DNEWS ist eine Software der Firma NetWin LTD, Neu Seeland und ist für verschiedene Plattformen (Solaris, HPUX, VMS, Win-NT, Linux, FreeBSD usw.) verfügbar.

Es handelt sich um ein kommerzielles Produkt mit Kosten von ca. $500 für die Vollversion, für akademische Einrichtungen ist die die Registrierung umsonst.

DNEWS ist primär als Sucking Feed konzipiert, d.h. es werden normalerweise nicht alle Newsgruppen per NNTP oder UUCP von anderen Rechnern angeliefert, sondern dynamisch nach lokalem Bedarf erst vom DNEWS-System selbst geholt.

Das DNEWS-System legt auch nicht mehr einzelne Dateien für die Artikel an, sondern packt Artikel zu Blöcken von 100 Stück zusammen. Trotzdem wird an der Speicherung der Artikel in Gruppen festgehalten, was bei einem all.all-Feed zu Performance-Problemen führen kann. Sobald die News nicht mehr gruppenweise geholt werden können sondern zufällig verteilt auf dem lokalen System eintreffen, hat das Schreiben von Artikeln einer Newsgruppe in Blöcke à 100 Stück keinen großen Vorteil mehr gegenüber der alten Methode pro Artikel eine Datei.

Eine Neuerung ist die Expire-Philosophie von DNEWS, welche zusätzlich zu festen Expire-Zeiten auch ein FIFO-Prinzip beinhaltet, d.h. es kann beispielsweise konfiguriert werden, daß bei akutem Festplattenmangel in jeder Gruppe nur noch 100 Artikel verbleiben sollen.

DNEWS hat einen etwas zweifelhaften Ruf. Die Probleme mit der Software sollen heute sehr viel geringer sein als dies noch vor ein paar Jahren war, trotzdem gibt es Empfehlungen dieses System nicht für größere Newsserver einzusetzen.

### 1.5.2 Diabolo

Diabolo ist eine Software von Matt Dillon, http://www.backplane.com/. Dieses System legt ankommende News alle 10 Minuten in einem anderen Verzeichnis ab, dort allerdings weiterhin die Artikel als einzelne Dateien.

Über eine zusätzliche Hash-Tabelle mit History-Funktion ist der Zugriff auf die Artikel ist damit sehr einfach möglich, der Expire aber trotzdem weiterhin ein elementares Problem.

Bis dato besteht für Diabolo kein Enduser-Support, das System kann nur andere Newsserver mit Artikeln versorgen. In einem Newsposting vom 25. Juli 1997 (<5rbmdd$br1$1@flea.best.net>) schreibt Matt Dillon, daß er in ca. 5 Monaten eine Newsreader-fähige Version von Diabolo verfügbar haben wird.

News wird in verschiedene Klassen eingeteilt, innerhalb der Klassen reiner FIFO-Durchsatz, benötigt Memory-Funktionen und wird nur auf einer begrenzten Anzahl von Betriebssystemen unterstützt.

### 1.5.3 Cyclone Newsrouter

Dieses Newssystem stammt von der Firma HighWind Software, Inc. und ist auch ein kommerzielles Produkt (Kosten ca. $6000).

Ankommende News werden auch in großen Blöcken gespeichert und teilweise im Memory gecached, bevor sie an andere Newsserver weiterverteilt werden. Die Artikel werden wieder gelöscht, sobald sie an alle anderen Feeds weiterverteilt wurden.

Cyclone verfügt in seiner derzeitigen Form über keinen Newsreader-Support und ist nur zur Weiterverteilung von News geeignet, so daß man immer noch einen zusätzlichen Newsserver benötigt um selbst Newsartikel lesen zu können. Dadurch sind die Einsatzmöglichkeiten dieses Systems beschränkt, als reiner „News-Router" kann das System aber sehr schnell sein.

### 1.5.4 Zusammenfassung

Bis heute hat sich kein System mit einer anderen Speichertechnik als das etablierte „pro Artikel eine Datei" durchsetzen können. Die Alternativen zu INN unterstützen entweder keinen Reader-Support, sind relativ teuer und/oder nicht für große Fullfeeds geeignet.

Viele Newsadministratoren sind Neuerungen gegenüber reserviert eingestellt. Ein völlig neues Newssystem einzuführen wird schwierig sein, Verbesserungen an einem etablierten System vorzunehmen, haben eher die Chance Anwendung zu finden.

# 2 Usenet-News

## 2.1 Namenskonventionen

Den Rechnerverbund aller Newsserver nennt man allgemein das *Usenet*. Das Usenet besteht zum einen aus *Feeds*, welche die Nachrichten (genannt *Artikel*) verschicken und *Clients*, welche die Artikel empfangen. Wie schon erwähnt, sind viele dieser Rechner per Internet verbunden, das Usenet umfaßt aber alle Usenet-News austauschende Rechner, also auch solche die nur per Modem und UUCP angebunden sind. Die Diskussionsforen nennt man *Newsgroups*, das Absenden von Artikel nennt man *Posten*, das Beantworten eines Artikels einen *Reply*.

## 2.2 Aufbau eines News-Artikels

Zur Veranschaulichung betrachten wir einen Artikel aus der internationalen Newsgruppe sci.eletronics.basics:

```
Path: swabsib!news.bawue.de!news.belwue.de!fu-berlin.de!news.mathworks.com!
uunet!inXS.uu.net!newsfeed.internetmci.com!news.intersurf.net!multiverse!
199.2 18.112.132!sam
From: sam@stdavids.picker.com (Sam Goldwasser)
Newsgroups: sci.electronics.basics
Subject: Re: reducing DC voltage
Date: 01 Jul 1996 23:04:48 GMT
Organization: Multiverse
Lines: 15
Message-ID: <SAM.96Jul1190448@colossus.stdavids.picker.com>
References: <rmanbike-2906961818450001@sb32.znet.com>
NNTP-Posting-Host: 144.54.160.15
In-reply-to: rmanbike@znet.com's message of Sat, 29 Jun 1996 18:18:45 -0700


In article <rmanbike-2906961818450001@sb32.znet.com> rmanbike@znet.com
(Roger Helton) writes:


> What is the simplest method of reducing a 9.6 nicad battery pack to 7.2
> voltage and still maintain the amperage of former. Have a 7.2 krypton bulb
> running off of 9.6 at this time burns out the bulb fairly quickly but
> makes a intense light though.


I assume that you mean drop the voltage and let the current be whatever
is proper for the bulb at that voltage.  As others have suggested, you
can use three 3 A diodes or if you know the current, a suitable resistor.


You could also remove 2 cells from the battery pack if the lamp is its
only load - this would avoid the wasted heat from the diodes or resistor.


    --- sam
```

Dieser Artikel ist ein Reply von einem Herrn Sam Goldwasser auf eine Frage von einem Herrn Helton. Der Artikel ist einteilbar in drei Bereiche: Einen *Header*, einen *Quote* und dem eigentlichen neuen Text, den Sam geschrieben hat.

### 2.2.1 Artikel-Header

Der Header umfaßt (in diesem Fall) das obere Drittel des Artikels und wird größtenteils automatisch vom Newssystem erzeugt. Jede Zeile des Newsheaders beginnt mit einem Keyword inkl. Doppelpunkt (hier fett dargestellt) und einem Wert. Eine Leerzeile beendet den Header. Die meisten Newsreader-Programme geben dem normalen Leser nur einen Teil des Headers aus, da viele Zeilen nur von technischem Interesse sind.

Die Zeile `From:` sagt z.B. aus, von wem der Artikel geschrieben wurde (sowohl Emailadresse als auch der richtige Name der Person). Die Zeile `Organization:` gibt den Betreiber des Newssystems an, von welchem der Artikel ursprünglich gepostet wurde. Dies kann ein Firmenname sein, bei vielen privaten Newssystemen aber auch „Private Site" oder andere Texte.

`Date:`, `Newsgroups:`, `Subject:` und `Lines:` geben an, wann in welche Newsgruppe wie viele Zeilen Text zu welchem Thema gepostet wurden. Die Zeile `NNTP-Posting-Host:` nennt den Rechner, von welchem der Artikel ursprünglich mittels NNTP an das erste Newssystem gesendet wurde.

Nur die Subject-Zeile muß selbst eingegeben werden und sollte in aller Regel mit Bedacht gewählt werden, damit andere Leser sofort das Thema des Artikels erkennen können. Nachdem der Artikel ein Reply auf einen anderen Artikel ist, fängt das Subjekt mit den Buchstaben „Re:" (für Reply) an und hat ansonsten dasselbe Subjekt wie der ursprüngliche Artikel.

Die `Reference:`-Zeile nimmt Bezug auf den ursprünglichen Artikel, auf den hier geantwortet wird. Dadurch wird es den Newsreader-Programmen ermöglicht die Artikel dem Leser in der richtigen Reihenfolge zu präsentieren.

Die Zeile `Path:` gibt an, über welche Rechner der Artikel bis zu unserem System gelaufen ist. Jedes Newssystem hängt dabei seinen eigenen Namen vorne in die Path-Zeile ein und verschickt dann den Artikel weiter an alle Newssysteme, bei denen der Artikel laut Path-Zeile noch nicht vorbeigekommen ist. Zusätzlich wird jeder Artikel noch mit einer eineindeutigen Message-ID versehen, an der jeder Artikel eindeutig identifiziert werden kann und wodurch verhindert wird, daß ein Artikel mehrmals auf dem System angeliefert wird.

### 2.2.2 Besondere Artikel-Header

Bei manchen Newsartikeln ist es sinnvoll, die Verweildauer des Artikels auf den einzelnen Newsservern beim Posten des Artikels explizit steuern zu können.

So ist ein Posting mit einem Angebot zur Mitfahrgelegenheit nur solange sinnvoll, bis das Datum der Abfahrt erreicht ist. Der Artikel kann spätestens an diesem Datum gelöscht werden.

Andererseits gibt es andere Postings bei denen es Wünschenswert wäre sie möglichst lange auf dem System zu halten, z.B. regelmäßige Postings mit einer Einführung über das Thema einer Newsgruppe (FAQ).

Es ist deswegen möglich einem Artikel eine zusätzliche Headerzeile `Expire:` zuzufügen, mit welcher man das erwünschte Löschdatum für den Artikel angeben kann. Auf den Newssystemen kann man dann einstellen, in welchem Rahmen solchen expliziten Expire-Headers dem Zeitpunkt entsprochen wird.

Eine andere Sonderstellung nehmen Artikel ein, bei denen im Header eine Zeile mit dem Keyword `Control:` vorhanden ist. Solche Artikel nennt man *Control-Messages* und haben den Zweck, mittels eines Newsartikels auf anderen Newsservern bestimmte Aktionen auszulösen.

Über diesen Mechanismus können beispielsweise automatisch neue Newsgruppen auf allen Newsservern eingerichtet oder aber alte auch wieder gelöscht werden. Ein anderer Einsatz ist das nachträgliche Löschen eines verschickten Newsartikels mittels eines *Cancels*, was auch per Control-Message erreicht wird.

Control-Artikel können zwar in allen Newsgruppen geschrieben und verschickt werden, werden aber aus Übersichtlichkeitsgründen auf den Newssystemen in einer speziellen Newsgruppe gespeichert, der *Control-Group*.

Nach dem Header folgt getrennt durch eine Leerzeile der eigentliche Artikel. In diesem Fall hat der Autor den alten Artikel, auf den er Bezug genommen hat gequoted, d.h. der wichtige Teil wurde durch „>"-Zeichen eingerückt nochmals mit verschickt, damit man auch ohne das Original weiß, worum es eigentlich geht. Auch dies machen die Newsreader-Programme automatisch. Es gehört zur Netiquette, nicht mehr als unbedingt notwendig zu quoten, um Bandbreite zu sparen.

## 2.3 Hierarchien der Newsgruppen

Zu Beginn der Usenet-Zeiten wurden den Newslesern einfach alle Artikel seit dem letzten Programmstart des Newsreaders angeboten. Diese Technik war mit dem steigenden Newsaufkommen nicht mehr überschaubar, so daß die Newsgruppen in verschiedene Bereiche aufgeteilt wurden. Die internationalen Top-level Hierarchien lauten heute:

| | |
|---|---|
| alt | Aktuelles und umstrittenes |
| comp | Alles rund um das Thema Computer |
| misc | Thema „Verschiedenes" |
| news | News-Verwaltung und Software |
| rec | Hobby, Sport, Freizeit |
| sci | Wissenschaftliche Themen |
| soc | Soziales und gesellschaftliches |
| talk | Diskussionsorientierte Gruppen |

Jede Top-level-Gruppe enthält mehrere Second-level-Gruppen, die wiederum Third-level-Gruppen usw. enthalten können. Briefmarkensammler treffen sich also in rec.collecting.stamps, über pakistanische Geschichte wird in soc.culture.pakistan.history diskutiert und Probleme mit SCSI-Geräten wird in der Gruppe comp.periphs.scsi behandelt.

Zusätzlich gibt es noch lokale Newshierarchien, die nicht überall erhältlich sind oder in denen die Teilnehmer sich nicht auf Englisch verständigen:

| | |
|---|---|
| aus | australische Newsgruppen |
| de | deutschsprachige Newsgruppen |
| fj | japanische Newsgruppen |
| freiburg | News aus Freiburg |
| maus | News aus dem Maus-Mailbox-Netz |
| fido | News aus dem Fido-Mailbox-Netz |

Das Einrichten neuer Newsgruppen geschieht nach mehr oder weniger ausführlicher Diskussion in dafür vorgesehen Newsgruppen (z.B. news.groups oder für de. in de.admin.news.groups) per Wahl, in Form von Emails an einem bestimmten Wahlleiter. Die Aufrufe zur Diskussion über eine neue Newsgruppe werden RfD (Request for Discussion), die Aufrufe zur Wahl CfV (Call for Vote) genannt. Näheres dazu findet man in den entsprechenden Newsgruppen. Man kann natürlich unabhängig davon lokale Hierarchien anlegen, in denen z.B. firmen- oder organisationsinterne Dinge per News bekanntgegeben oder besprochen werden.

## 2.4 Expire

Nachdem täglich mehrere hundert Megabytes Newsartikel auf einem Newsserver eintreffen, müssen diese nach einer gewissen Zeit der Speicherung auch wieder

gelöscht werden. Dieser Vorgang wird *Expire* genannt und erfolgt gewöhnlichweise einmal jede Nacht. Auf manchen Systemen werden zur gleichmäßigeren Plattenauslastung mehrere Expires am Tag ausgeführt.

Die Newsadministratoren haben dabei die Möglichkeit den Expire auf Gruppenbasis verschieden lange einzustellen. Beispielsweise löscht der Expire dann Artikel in Gruppen mit Bildern und andere große Datenmengen schon nach einem Tag, während Artikel in lokalen Newsgruppen bis zu einem Monat aufbewahrt werden.

Der Expire ist für herkömmliche Newsserver eine stark belastende Aufgabe, welche auf großen Systemen mehrere Stunden dauern kann. Während dieser Zeit werden alle sonstigen Operationen durch den Expire gebremst.

## 2.5 Crosspostings

Falls ein Newsartikel nicht nur für eine Newsgruppe, sondern für mehrere thematisch zutreffend ist, kann man durch Angabe mehrerer Gruppen den Artikel dort gleichzeitig erscheinen lassen. Dies nennt man ein Crossposting und hat den Vorteil, daß die Newssysteme solche Artikel nur einmal (unabhängig von der Anzahl der betroffenen Newsgruppen) verschicken müssen und die Artikel auch nur einmal auf der Festplatte gespeichert werden.

Im Falle der klassischen Artikelverwaltung geschieht dies innerhalb eines Dateisystems per Hardlink, d.h. der Artikel erscheint als mehrere Dateien in allen betroffenen Newsgruppen, belegt aber nur einmal Speicher. Bei verschieden langen Expire-Zeiten kann die Datei des Artikels dann auch individuell in den verschiedenen Verzeichnissen gelöscht werden, ohne daß dadurch die anderen Verzeichnisse betroffen sind.

Wenn das Newssystem aber über mehrere Dateisysteme verteilt ist, sind Hardlinks nicht mehr möglich und es muß mit symbolischen Links gearbeitet werden. In diesem Fall ist das individuelle Löschen problematisch, da hier eine Datei unter Umständen noch von anderer Stelle aus referenziert ist und deswegen erhalten bleiben muß.

## 2.6 Spams

Ein Spam ist die Bezeichnung für das (mutwillige oder versehentliche) Verschikken von Newspostings mit gleichem oder sehr ähnlichem Inhalt in einer Vielzahl von Gruppen. Es kommt dabei nicht darauf an, ob der Artikel per Crossposting oder tatsächlich mehrfach verschickt wurde.

Viele unseriöse Firmen und Privatpersonen versuchen auf diesem Wege mit minimalen Kosten sehr viele Teilnehmer des Usenets zu erreichen, um ihre Dienste und Angebote anzupreisen. Auch bei einer sehr geringen Erfolgsquote von wenigen Promille scheint sich der Aufwand noch zu lohnen.

So vorteilhaft und günstig dieses Vorgehen für den Versender von Spams ist, so lästig ist es für die Allgemeinheit des Usenets: In allen Newsgruppen tauchen Artikel mit völlig gruppenfremden Inhalten auf, durch die sich die Leser der Newsgruppe hindurchquälen müssen und die auf Kosten der Allgemeinheit Netzwerkbandbreite und Festplattenkapazität verbrauchen.

Es wurden deswegen in der Zwischenzeit Cancel-Robots programmiert, die alle Newsartikel scannen und dabei nach Spams Ausschau halten. Wenn ein Spam erkannt wurde, wird für alle Artikel dieses Spams entsprechende Cancel-Messages verschickt. Daraufhin werden die Spam-Artikel auf allen Systemen gelöscht oder aber (falls die Cancel-Message vor dem Spam auf einem System eintrifft) gar nicht mehr angenommen.

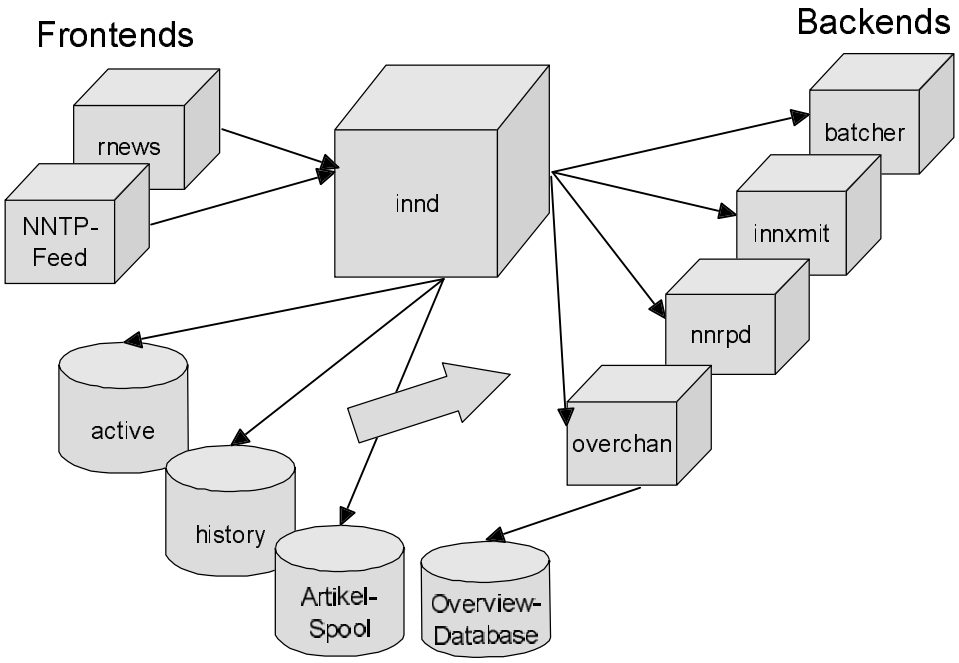
Dies führt allerdings dazu, daß auf den Newsservern heute im Verzeichnis „control" täglich mehrere zigtausend Dateien angelegt und (wegen des Expires) auch wieder gelöscht werden müssen. Dateisysteme haben aber große Probleme in einem einzigen Verzeichnis viele Tausend Dateien zu verwalten.

# 3 Aufbau des bisherigen INN 1.5.1

## 3.1 Übersicht

Das Programmpaket InterNetNews basiert zu großen Teilen auf der Entwicklung von Rich Salz Anfang der 90er Jahre. In der Zwischenzeit wurde die Entwicklung vom Internet Software Consortium[3] übernommen.

Das System besteht aus vielen einzelnen Programmen mit einem Sourcodeumfang von ca. 50000 LOC. Der Aufbau des INN läßt sich wie folgt einteilen:




## 3.2 innd

Das eigentliche Herzstück des INN-Systems ist ein Daemon, der alle Artikel entgegennimmt und deren Header überprüft (und im Zweifelsfall entsprechend ergänzt), sie danach auf die Platte ablegt und für alle hinter ihm liegenden Sites entsprechend vermerkt.

[3] Siehe http://www.isc.org/

### 3.2.1 channels

Der **innd** hält dazu eine große Anzahl von Deskriptoren offen (sog. Channels), über die er von anderen Feeds Artikel per NNTP-Protokoll geschickt bekommt. Auf der anderen Seite werden über andere Channels entweder direkt andere Programme per stdin/stdout mit Daten versorgt, oder aber Verweise auf bestimmte Artikel in Dateien vermerkt. Falls der **innd** mehr Channels versorgen muß als er Dateideskriptoren vom Betriebssystem zur Verfügung gestellt bekommt, puffert er die Daten im Speicher und öffnet die Files zyklisch.

Der **innd** verwaltet auch mehrere Systemdateien, in denen Informationen über Artikel gespeichert werden:

### 3.2.2 Die Datei active

Zum einen handelt es sich dabei um das `active`-File, in welchem alle Newsgruppen des Systems vermerkt sind.

```
de.talk.misc 0000048218 14697 y
de.talk.sex 0000094854 75906 y
de.test 0000049674 38887 y
junk 0000067861 57385 y
news.announce.important 0000000025 00026 m
news.announce.newusers 0000001103 01092 m
news.lists 0000002463 02464 m
news.lists.ps-maps 0000001002 01003 m
soc.human-nets 0000000000 00001 m
stgt.amiga 0000000259 00208 y
stgt.atari 0000000299 00267 x
```

Die Datei besteht aus einer Tabelle mit 4 Spalten, als erstes der Gruppenname, danach die größte und kleinste vorhandene Artikelnummer in dieser Gruppe und zuletzt ein Flag welches aussagt, ob jeder in die Gruppe posten darf (y), ob die Gruppe moderiert ist (m) oder aber, ob die Gruppe auf dem System unerwünscht ist (x).

### 3.2.3 renumber

Jedes Mal, wenn ein neuer Artikel auf dem System angeliefert wird, inkrementiert der **innd** die entsprechende höchste Artikelnummer um Eins. Die niedrigste Artikelnummer bleibt immer auf dem selben Wert, da der **innd** selbst keine Artikel löscht.

Nach einem Expire (siehe unten) müssen deswegen die niedrigsten Artikelnummern neu gesetzt werden. Dies geschieht durch einen sog. *renumber*, bei welchem der **innd** sich durch alle Verzeichnisse arbeitet und dort nach Dateien sucht, die nur aus Zahlen bestehen. Diese Dateien werden als Artikel interpretiert und danach dann das active-File entsprechend angepaßt.

### 3.2.4 Die Datei history

Die andere Systemdatei ist die `history`, in der von allen Artikeln die Message-ID, der Zeitpunkt des Postings des Artikels, der Zeitpunkt des Eintreffens des Artikels und die Newsgruppen, in denen der Artikel auf dem System abgelegt wurde, gespeichert werden.

```
<33CA4518.4E49@lowellsun.com>   868901120~-~868893972   rec.games.computer.quake.playing/308
<19970714132301.JAA20911@ladder02.news.aol.com> 868901120~-~
<5qdck4$c3f@eyrie.graphics.cornell.edu> 868901120~-~868890052   rec.audio.high-end/60
<nate-1407971032310001@bos-ip-3-127.ziplink.net>   868901120~-~868890750   rec.music.folk/195
<5qdc0t$8lm$1@scream.auckland.ac.nz>    868901120~-~868889437   rec.food.cooking/6467
<5qddtd$5g8@gazette.engr.sgi.com>       868901121~-~868891373   rec.scuba/285
<5qd9fv$2b0@mtinsc04.worldnet.att.net>  868901121~-~868886854   rec.arts.sf.starwars.misc/682
<33CA51E3.1729@bcpl.lib.md.us>  868901121~-~868897251   rec.radio.amateur.misc/207
<33CA1C68.606D@nortel.ca>       868901122~-~868883560   rec.models.scale/401
<5qddnt$asp@squire.cen.brad.ac.uk>      868901122~-~868891197   rec.sport.rugby.league/88
```

Die Message-ID wird benötigt, damit der **innd** entscheiden kann, ob er einen Artikel schon erhalten hat. Der dazugehörige Name gibt an, wo der Artikel zu finden ist. Die verschiedenen Uhrzeiten werden vom **expire** benötigt, damit schnell entschieden werden kann, wann Artikel wieder gelöscht werden sollen.

Wenn ein Artikel expired, werden die Newsgruppeneinträge gelöscht, die restlichen Informationen über den Artikel aber noch für mehrere Tage gehalten, um ein erneutes Empfangen desselben Artikels (sog. Dupes) ausschließen zu können. Als Beispiel ist hier der Artikel mit der Message-ID <19970714132301.JAA20911@ladder02.news.aol.com> gegeben.

## 3.3 frontends

Obwohl der **innd** Artikel per NNTP direkt entgegen nehmen kann, gibt es noch andere Möglichkeiten dem **innd** Newsartikel zu übergeben. So werden z.B. Newsbatches, die per UUCP geholt wurden, von einem Programm namens **rnews** entpackt und dann dem **innd** zur Verfügung gestellt.

Die Frontends machen nichts anderes als Newsartikel zu laden, die in einem bestimmten Format an dem System angeliefert wurden (per UUCP-Batch, per Mail oder wie auch immer). Dem **innd** werden diese Artikel dann vom Frontend per lokaler NNTP-Verbindung angeboten. Aus der Sicht des **innd** besteht dabei keinerlei Unterschied zu einer sonstigen NNTP-Verbindung zu einem anderen Newsserver irgendwo im Internet.

## 3.4 backends

Der **innd** kommuniziert mit seinen Nachbarsites nur bezüglich dem Empfangen von Artikeln selber, das Verschicken wird von anderen Programmen vorgenommen, den sog. backends.

### 3.4.1 Die Konfigurationsdatei newsfeeds

Die Informationen, welcher Site was für Newsartikel zu schicken sind und welches Backend-Programm dafür einzusetzen ist, erfährt der **innd** aus der Konfigurationsdatei `newsfeeds`.

Diese Datei ist ziemlich komplex und wird deswegen hier nicht vollständig beschrieben. Für uns von Interesse ist, daß der **innd** je nach Eintrag in der Datei newsfeeds Informationen für die Backends in verschiedenen Formaten ablegen und übermitteln kann.

```
bawue/news.bawue.de:\
      *\
      :Tf,Wnm:
```

Dieser Eintrag hier bewirkt beispielsweise, daß für den Rechner news.bawue.de, der News per NNTP zugestellt bekommt, eine Datei angelegt wird (Parameter „f" beim „Tf") in der alle Newsartikel hinterlegt werden die auf dem lokalen System ankommen (oder dort gepostet werden)[4].

Das Format dieser Datei wird durch den Eintrag „Wnm" definiert, in diesem Fall „n" für Name der Newsartikel und „m" für Message-ID. Die Datei sieht dementsprechend dann so aus:

```
de/comp/os/ms-windows/misc/17913 <393367@news.koeln.netsurf.de>
de/etc/finanz/boerse/27376 <33D878@wsrz2.wiso.uni-erlangen.de>
de/comp/os/ms-windows/misc/17914 <33df38@news.koeln.netsurf.de>
de/comm/mobil/27059 <5ralbf$qa2@guide.muc.de>
de/comp/sys/ibm-pc/18959 <33e33e99@news.koeln.netsurf.de>
de/alt/flame/9388 <33dbddc8.708096@news.swol.de>
de/comm/infosystems/www/authoring/11789 <01bc9@tibet.ltts.com>
```

Das backend-Programm **innxmit** (siehe Kapitel 4.3.3, Seite 32) kümmert sich dann per cronjob um die Weiterverarbeitung der aufgelisteten Newsartikel.

Dem System phranjep werden nur Newsartikel von bestimmten Newsgruppen per UUCP zugestellt, weswegen anstatt des Wildcards '*' eine Liste von Newsgruppen angegeben wird. Auch in diesem Fall wird wieder eine Datei angelegt,

[4] Es sei denn, sie waren schon vorher beim System news.bawue.de, was unser **innd** anhand der „Path:"-Zeile des Artikelheaders erkennen kann - siehe Seite 8

dieses Mal aber ein anderes Format gewählt, welches für das Backend-Programm **batcher** (siehe Kapitel 4.3.2, Seite 30) besser geeignet ist.

```
phranjep/phranjep.lb.bawue.de:\
      comp.infosystems.www*,\
      *next*,\
      de.comp.os.unix,\
      stgt\
      :Tf,Wfb:
```

Die Parameter „Wfb" erzeugen hiermit **F**ilenamen und die Länge der einzelnen Artikel in **B**ytes.

```
      de/comp/os/linux/x/8705 5132
      de/comp/os/linux/x/8706 4869
      de/comm/isdn/technik/5317 15007
      de/comp/misc/16644 279
      de/etc/finanz/boerse/27377 9731
      de/org/politik/spd/8425 345
      de/rec/motorrad/102098 9485
      de/talk/romance/66298 2256
      de/comp/os/ms-windows/misc/17916 1208
```

Auch zur Klasse der Backends (wenn auch nicht direkt für die Kommunikation mit anderen Sites zuständig) gehören Programme, welche Newsartikel anderweitig weiterverarbeiten. Es werden dann z.B. aus den vorbeikommenden News Statistiken erstellt, die Artikel nach bestimmten Schlüsselwörtern durchsucht oder Newsartikel bestimmter Gruppen archiviert.

Zu dieser Gruppe von Backends gehört auch das Programm **overchan**. Dieses verwaltet über alle Newsgruppen die sog. overview-Database, eine kleine Datenbank in der von allen auf dem System vorhandenen Artikeln in komprimierter Form die wichtigsten Daten (Name des Autors und des Subjects, Größe, Datum, References usw.) gespeichert werden, damit die Newsreader schnell Übersichten über die Newsgruppen darstellen können. Technisch wird dies dadurch realisiert, daß die Overview-Database pro Newsgruppe eine Datei anlegt, in welche die Daten geschrieben werden.

```
overview:\
      *\
      :Tc,WO:/usr/local/news/bin/overchan -D /var/spool/news
```

```
39019 Re: Hochwasser in Polen & Eisenbahnverkehr  rushertz@zsdjh.rus.uni-stuttgart.de (Joerg
Hertzer)     18 Jul 1997 18:17:09 GMT        <5qobv5$6qe$1@news.belwue.de>
<5qliec$spe$1@news.belwue.de> <19970718134601.JAA15256@ladder01.news.aol.com>
<5qntcu$5hr$1@news.belwue.de> <33CFA77A.446B@mail.uni-mainz.de>   1289    17
39020   Re: Bahnhofsbezeichnungen in Berlin     Thomes_Bond@tmb.in-berlin.de (Thomas Wendt)
 Fri, 18 Jul 97 15:48:00 CET     <6a6myfmycOB@thomes.tmb.IN-Berlin.de>
<869174801.4392@dejanews.com>   1814    28
39021   Re: [offtopic] Re: Loveparade: Raver legen S-Bahnbetrieb lahm   olaf@bigred.inka.de
(Olaf Titz) 18 Jul 1997 16:20:20 +0200      <ediqi1.az1@bigred.inka.de>
```

```
<01bc8ed7$00d4f340$LocalHost@default>  <slrn5slb2f.4u.dirk@walhalla.nicolai-stift.hannover.de>
 <edh1qz.mbj@bigred.inka.de> <33CF1124.38FE@hrz1.uni-oldenburg.de>       1911    20
```

## 3.5 nnrpd

Die Newsreader kommunizieren auch per NNTP mit dem Newsserver. Der **innd** macht dies allerdings nicht selbst (er kümmert sich nur um die Kommunikation mit anderen Newsservern), sondern startet dafür ein weiteres Programm, den **nnrpd**. Dieser bietet den News-Clients eine Liste von Newsgruppen an, zeigt die vorhandenen oder seit dem letzten Zugriff neuen News einer Gruppe und verschickt schlußendlich auch Artikel.

Der **nnrpd** kann auf Anforderung der Newsreader auch Overview-Daten über Newsgruppen verschicken. Falls dabei die notwendige Overview-Database nicht vorliegt, werden die Daten „on the fly" aus den einzelnen Artikeln der Newsgruppe generiert

# 4 Vorgenommene Änderungen

## 4.1 Neue Speichertechnik der Artikel

### 4.1.1 News in Dosen

Die neue Organisation und Abspeichertechnik der Newsartikel ist der Kern der Arbeit. Für jeden Newsartikel wird ab sofort beim Eintreffen auf dem Newssystem das Datum des geplanten Expires berechnet und der Artikel dementsprechend an eine große Datei angehängt.

Diese Dateien bezeichne ich als *Dosen* oder Englisch *cans*, weil sie wie Konservendosen über ein Verfallsdatum verfügen, nach deren Überschreitung sie weggeworfen werden sollten. Das Verfallsdatum ist bei den Artikel-Cans durch den Dateinamen festgelegt, der aus Jahreszahl, Monat, Tag, Stunde und einer zweistelligen aufsteigenden Nummer besteht.

Im Gegensatz zu echten Dosen verfügen Artikeldosen (technisch gesehen) über ein uninteressant großes Füllvolumen, von in der Regel mindestens 2 GBytes, was ca. 0,5-1 Million Artikel entspräche. Nachdem aber Dateien dieser Größe sehr unhandlich sind, verfügt das System über eine einstellbare Maximalgröße für Artikeldosen, welche im Moment bei 100 MBytes liegt und vom Dateiformat bis auf 9.9 Gigabytes erweiterbar ist. Falls mehr Artikel zu einem bestimmten Expire-Zeitpunkt gespeichert werden müssen als in eine Artikeldose passen, wird eine neue Dose aufgemacht.

Die Zeiteinteilung, in welcher den Dosen ein Verfallsdatum aufgestempelt wird, muß zur Compile-Zeit angegeben werden. Derzeit ist als Zeiteinheit für den Expire 4 Stunden gewählt, was 6 Dosen am Tagen ergibt. Somit kann ein halbes Jahr in 1080 Dosen abgefüllt werden, was als guter Kompromiß zwischen handlicher Anzahl von Dosen und handlichem Zeitabstand für den Expire erschien.

Ein Verzeichnis mit Dosen sieht hiermit also wie folgt aus:

```
bash# ls -l
total 123134
-rw-rw-r--  1 news       21444 Jul 14 03:40 199707141401
-rw-rw-r--  1 news       87658 Jul 14 04:10 199707141801
-rw-rw-r--  1 news      126039 Jul 14 08:40 199707142201
-rw-rw-r--  1 news       89247 Jul 14 12:10 199707150201
-rw-rw-r--  1 news      121645 Jul 14 12:10 199707150601
-rw-rw-r--  1 news      128160 Jul 14 11:23 199707151401
-rw-rw-r--  1 news       15695 Jul 14 05:20 199707151801
-rw-rw-r--  1 news       70585 Jul 13 22:00 199707172201
-rw-rw-r--  1 news        5846 Jul 14 03:03 199707180201
[...]
-rw-rw-r--  1 news        2503 Jul 13 04:09 199707270201
-rw-rw-r--  1 news    20409793 Jul 14 11:30 199707272201
-rw-rw-r--  1 news     9323452 Jul 14 12:17 199707280201
-rw-rw-r--  1 news    22538099 Jul 14 11:42 199707280601
-rw-rw-r--  1 news    23483383 Jul 14 12:28 199707281001
-rw-rw-r--  1 news    10478460 Jul 14 12:23 199707281401
-rw-rw-r--  1 news    15296832 Jul 14 06:00 199707281801
-rw-rw-r--  1 news    14189184 Jul 14 09:12 199707282201
-rw-rw-r--  1 news     3963084 Jul 14 06:51 199707290201
```

### 4.1.2 Lagerung der Dosen

Die einzelnen Dosen werden im Dateisystem in beliebig vielen Verzeichnissen abgelegt. Dosen lagert man normalerweise auf einem Regal, dementsprechend befinden sich die Artikeldosen in Verzeichnissen, deren Namen mit „shelf." beginnen müssen. Der weitere Name des Verzeichnis ist frei wählbar, wodurch der Administrator die verschiedenen Regale den lokalen Gegebenheiten anpassen kann, also z.B. `shelf.quantum4Gig`, `shelf.seagate9Gig`, oder auch schlicht `shelf.01`, `shelf.02`, usw. Als Verzeichnisse sind dabei entweder „richtige" Directories, gemountete Dateisysteme oder Symbolic Links auf andere Festplatten möglich.

Die Dosen werden bis dato durch einen sehr einfachen Algorithmus auf die verschiedenen Regale verteilt: Bei jedem Öffnen einer neuen Dose wird der vorhandene Platz auf den verschiedenen Regalen berechnet und dann das Regal mit dem größten Platz genommen. Wenn mehrere Regale genau gleich viel Platz bieten, wird das erste von ihnen genommen. Es nützt also (noch) nichts, mehrere Verzeichnisse auf einem Filesystem anzulegen um dadurch die Anzahl an Dateien pro Verzeichnis niedrig zu halten.

Dieser Algorithmus sollte in Zukunft auf alle Fälle nochmals überarbeitet werden. Neben dem schon genannten Problem, daß mehrere Verzeichnisse auf einem Filesystem keinen größeren Nutzen haben, hat sich in der Praxis gezeigt, daß der Algorithmus bei Einsatz von deutlich unterschiedlich großen Regalen Probleme aufwirft:

Bei Neuinstallation des Systems werden die Dosen zu Beginn nur auf dem größten Regal plaziert. Dies geschieht solange, bis der noch vorhandene Platz auf dem größten Regal kleiner wird als die Gesamtgröße des zweiten Regals. Erst

dann beginnt das System einige neue Dosen auf dem zweiten Regal aufzustellen. Gleichzeitig werden aber Artikel weiterhin in mehrere schon angefangene Dosen auf dem ersten Regal gespeichert, es dauert einige Zeit, bis wirklich größere Mengen von Artikeln auf das zweite Regal gelangen. In der Zwischenzeit können aber noch mehrere Hundert MBytes Artikel auf dem ersten Regal untergebracht werden.

Kurz gesagt, sind zwischen dem Moment, wann eine Dose neu aufgemacht wird und dem, wann eine Dose stark benützt wird, so große Zeitunterschiede, daß die Regale sehr starken Belegungsschwankungen unterworfen sind, falls nicht ab Beginn die Dosen gleichmäßig auf sie verteilt werden.

Eine ungleichmäßige Verteilung der News auf die einzelnen Platten bewirkt allerdings Performance-Einbußen, wie im Laufe der Entwicklung des Systems festgestellt werden konnte.

### 4.1.3 Datenformat der Dosen

Das interne Datenformat der Dosen ist bewußt einfach gehalten. Jede Datei beginnt mit einer Zeile Magic, gefolgt von einer Zeile mit einer Versionsnummer dieser Dose. Dadurch ist die Möglichkeit offen gehalten, daß spätere Versionen des Systems auch andere Dosenformate (z.B. mit komprimierten Artikeln) verarbeiten können. Die derzeitige Implementierung sieht auch Mischbetrieb verschiedener Versionen von Dosen vor, so daß eine Umstellung problemlos möglich wäre.

Die dritte Zeile einer Dose besteht aus dem '+'-Zeichen und einer 10stelligen Zahl mit führenden Nullen, welche die Gesamtlänge der Dosendatei angibt. Sollte diese Zahl von der tatsächlichen Länge der Datei abweichen, so hat der **innd** die Datei gerade noch zum Schreiben geöffnet und die Längenzeile noch nicht auf den neusten Stand gebracht. Sollte in diesem Zustand das System abstürzen, ist klar, welche Dosen auf eventuell verstümmelte Artikel überprüft werden müssen.

Nach diesem Header folgen nun nacheinander die eigentlichen Artikel, getrennt jeweils durch 12 Bytes, bestehend aus dem ASCII-Zeichen '+', die 10stellige Länge des Artikels und dem Newline-Zeichen.

Der Inhalt einer Dose sieht exemplarisch wie das folgende Beispiel aus (an den Stellen mit „[...]" wurden die eigentlichen Artikel entsprechend gekürzt):

```
Hennerich's articleCan
Version 1.0
+0000798218
+0000019882
Path:
news.bawue.de!news.belwue.de!fu-berlin.de!newsfeed.nacamar.de!rz.uni-
karlsruhe.de\!news.urz.uni-heidelberg.de!german
From: dantefaq@dante.de (DE-TeX-FAQ-Maintainer)
Newsgroups: de.comp.text.tex,de.answers,news.answers
Subject: TeX, LaTeX, DANTE e.V.: FAQ - Fragen und Antworten (Part 1
of 11)
Supersedes: <de-tex-faq-1-867918916@dante.de>
Followup-To: de.comp.text.tex
Date: 3 Aug 1997 05:00:31 GMT
Organization: DANTE, Deutschsprachige Anwendervereinigung TeX e.V.

[...]

Anregungen, Ergänzungen, Korrekturen, Verbesserungsvorschläge, neue
Frage und Antworten per E-Mail bitte an

        dantefaq@dante.de

Danke schon jetzt im voraus!

                                        Bernd Raichle
                                        Thomas Hafner
                                E-Mail: dantefaq@dante.de

--- DE-TeX-FAQ Ende Teil 1 ---
+0000020142
Path:
news.bawue.de!news.belwue.de!fu-berlin.de!newsfeed.nacamar.de!blackbu
sh.xl
ink.net!rz.uni-karlsruhe.de\!news.urz.uni-heidelberg.de!german
From: dantefaq@dante.de (DE-TeX-FAQ-Maintainer)
Newsgroups: de.comp.text.tex,de.answers,news.answers
Subject: TeX, LaTeX, DANTE e.V.: FAQ - Fragen und Antworten (Part 2
of 11)
Supersedes: <de-tex-faq-2-867918916@dante.de>
Followup-To: de.comp.text.tex
Date: 3 Aug 1997 05:00:34 GMT
[...]
```

Für die Formatwahl der Längenkennzeichnung waren folgende Gründe ausschlaggebend:

- Das ASCII-Format macht die Datei mit üblichen UNIX-Tools leicht zu bearbeiten. Shell-Scripts können mit ASCII-Zahlen einfacher umgehen als mit einem 4 Byte long Integer.
- Eine 4 Byte-Zahl müßte u.U. zwischen Little- und Big-Endian-Systemen konvertiert werden, falls das gesamte Newssystem inkl. Artikelbestand auf eine andere Maschine umziehen müßte.
- Die 10stellige ASCII-Zahl kostet mit Begrenzern nur 8 Bytes pro Artikel mehr als eine 4 Byte-Zahl (was im Verhältnis zur Länge des Artikels zu vernachlässigen ist), bringt aber auch eine gewisse Sicherheit falsche Zeiger auf

Artikel in einer Dose zu erkennen, die nicht auf den Anfang eines Artikels zeigen.

- Durch Ersetzen des vorstehenden '+'-Zeichens ist es einfach möglich einen Artikel speziell zu kennzeichnen. So werden in der derzeitigen Implementierung z.B. gecancelte (also frühzeitig gelöschte) Artikel nicht wirklich aus der Dose entfernt, sondern einfach nur durch Ersetzung des '+'-Zeichens durch ein '-' als gelöscht markiert. Spätere Erweiterungen können durch andere Ersetzungen des '+' bzw. '-' - Zeichens noch andere Kennzeichnungen für bestimmte Artikel durchführen.

Einzelne Artikel aus einer Dose wieder zu entfernen ist allgemein nicht ohne weiteres möglich, da dadurch alle Zeiger auf Artikel in dieser Dose neu berechnet werden müßten.

Bei genügend vorhandenem Plattenplatz ist es allerdings denkbar, daß man im Rahmen des Expire-Prozeßes die einzelnen Dosen durchscannt und sich die Eigenschaft von UNIX-Filesystemen zunutze macht, daß Dateien Löcher beinhalten dürfen, d.h. man kann beim Beschreiben von Dateien bestimmte Speicherbereiche überspringen und dann weiter schreiben, ohne daß dabei durch den übersprungenen Bereich Plattenplatz belegt wird.

Dadurch kann der Speicherbedarf von expireten Artikeln auf die 12 Bytes der Artikellänge reduziert werden, allerdings benötigt man dazu zum Reorganisationszeitpunkt noch zusätzlichen Plattenplatz in der Größe der gerade zu bereinigenden Dose.

## 4.2 Einbindung der Dosen in den INN

Der **innd** verwaltet über jeden Artikel eine Datenstruktur[5], in welcher alle wichtigen Informationen über den Artikel gespeichert sind. Des weiteren verfügt der **innd** im Hauptspeicher über eine Liste aller ihm bekannten Newsgruppen, welche er beim Startup aus dem active-File generiert.

Die Datenstrukturen der einzelnen Newsgruppen wurden so modifiziert und erweitert, daß der **innd** von allen Newsgruppen nun den Expire-Zeitpunkt kennt. Dazu wurden Teile des Programmes **expire** im **innd** integriert, welche das File `expire.ctl` auswerten. Bei jedem Auslesen des active-Files wird deswegen gleichzeitig auch das `expire.ctl` analysiert und zu allen Newsgruppen der expire-Zeitpunkt festgelegt. Ein Ändern des Konfigurationsfiles `expire.ctl` muß also dem **innd** per **ctlinnd**-Kommando „reload active" bekannt gemacht werden.

```
/*
**  A newsgroup has a name in different formats, and a high-water count,
**  also kept in different formats.  It also has a list of sites that
**  get this group.
*/
typedef struct _NEWSGROUP {
  long                  Start;        /* Offset into the active file      */
  char                  *Name;
  char                  *Dir;         /* The name, as a directory         */
  int                   NameLength;
  ARTNUM                Last;
  ARTNUM                Filenum;      /* File name to use                 */
  int                   Lastwidth;
  int                   PostCount;    /* Have we already put it here?     */
  char                  *LastString;
  char                  *Rest;
  SITEIDX               nSites;
  int                   *Sites;
  SITEIDX               nPoison;
  int                   *Poison;
  struct _NEWSGROUP     *Alias;
  time_t                Keep;         /* TH - from expire.c               */
  time_t                Default;
  time_t                Purge;
} NEWSGROUP;
```

Die Datenstrukturen des gerade zu bearbeitenden Artikels wurden so modifiziert, daß beim Eintreffen des Artikels berechnet wird, wann der Artikel wieder gelöscht werden soll und dieser Zeitpunkt dann bei den Eigenschaften des Artikels gespeichert wird. Wenn es sich bei einem Artikel um ein Crossposting

[5] Die Datenstruktur besteht vom Zeitpunkt des Eintreffens des Artikels bis zum Abspeichern auf der Festplatte und dem Weiterverteilen zu anderen Newssystemen.

handelt, wird der Artikel entsprechend der am längsten auf dem System gehaltenen Newsgruppe behandelt. Eventuelle Expire-Zeilen in den Artikeln selbst werden entsprechend der eingestellten Grenzen von `expire.ctl` dabei berücksichtigt.

```
/*
**  Miscellaneous data we want to keep on an article.  All the
**  fields are not always valid.
*/
typedef struct _ARTDATA {
   STRING      Poster;
   STRING      Replyto;
   char                *Body;
   time_t              Posted;
   time_t              Arrived;
   time_t              Expires;
   time_t      Delete;          /* Tobi */
   int                 Groupcount;
[...]
} ARTDATA;

STATIC void
ARTlookforexpire(Data)
    ARTDATA    *Data;
{
    register NEWSGROUP **ngptr;
    time_t time;

    Data->Delete=0;

    for( ngptr=GroupPointers; *ngptr; ngptr++ )
    { if( Data->Expires )
      {   if( Data->Expires < Now.time+(*ngptr)->Keep )
              time=Now.time+(*ngptr)->Keep;
          else if( Data->Expires > Now.time+(*ngptr)->Purge )
              time=Now.time+(*ngptr)->Purge;
          else
              time=Data->Expires;
      }
      else
          time=Now.time+(*ngptr)->Default;

      if( time>Data->Delete )
          Data->Delete=time;
    }

 /*syslog(L_TRACE, "article %s delete at %d", Data->MessageID,
Data->Delete);*/
}
```

Das Abspeichern der Artikel geschieht im Sourcecode des **innd** im File `innd/art.c`. Dort mußte also der Zugriff auf das Filesystem durch den Zugriff auf die Artikeldosen ersetzt werden.

```
/*
**  Write an article using writev.  The article is split into pieces,
**  shown below separated by pipe signs.  The items in square brackets are
**  "inserted" by this routine.
**     |headers...
**     Path: |[Path.Data]|rest of path...
**     headers...
**     |[Lines header, if needed]|
**     |[Xref header]|
**
**     Article body.
**  Also, the Data->Size field is filled in.
*/
STATIC int
ARTwrite(Delete, Article, Data)
    time_t                  Delete;
    BUFFER                  *Article;
    ARTDATA                 *Data;
{
    static char             WHEN[] = "article";
    static char             NL[] = "\n";
    static BUFFER  Headers;
 /*    register int           fd;*/
    register IOVEC *vp;
    register long  size;
    register char  *p;
    IOVEC                   iov[7];
    IOVEC                   *end;
    char                    bytesbuff[SMBUF];
    int                     i;
    ARTCAN                  *can;
    char                    buffer[1024];

    if ((p = HeaderFind(Article->Data, "Path", 4)) == NULL
     || p == Article->Data) {
        /* This should not happen. */
        syslog(L_TRACE, "%s internal %s no Path header",
            Data->MessageID, LogName);
        return -1;
    }

    /* Set up the scatter/gather vectors. */
    vp = iov;
    size = 0;
    vp->iov_base = Article->Data;
    vp->iov_len  = p - Article->Data;
    size += (vp++)->iov_len;
    vp->iov_base = Path.Data;
    vp->iov_len  = Path.Used;
    size += (vp++)->iov_len;
    vp->iov_base = p;
    vp->iov_len  = Data->Body - p;
    size += (vp++)->iov_len;
    if (ARTheaders[_lines].Found == 0) {
        (void)sprintf(Data->Lines, "Lines: %d\n", Data->LinesValue);
        i = strlen(Data->Lines);
        vp->iov_base = Data->Lines;
        (vp++)->iov_len  = i;
        size += i;
        /* Install in header table; STRLEN("Lines: ") == 7. */
        (void)strcpy(ARTheaders[_lines].Value, Data->Lines + 7);
        ARTheaders[_lines].Length = i - 7;
        ARTheaders[_lines].Found = 1;
    }

    /* Install in header table; STRLEN("Xref: ") == 6. */
    HDR(_xref) = Xref.Data + 6;
    ARTheaders[_xref].Length = Xref.Used - 6;
    ARTheaders[_xref].Found = 1;

    vp->iov_base = Xref.Data;
    vp->iov_len  = Xref.Used;
    size += (vp++)->iov_len;

    /* Open the file. Already here, because we need the canpos... */
    can=CANopenForWrite((((time_t)(Delete/CANTIME))+1)*CANTIME, size);
```

```
sprintf(buffer, "Xcanpos: %s!%010ld\n", can->Name, can->Bytepos[WRITE]);
Data->Canpos=COPY(buffer);

HDR(_xcanpos) = Data->Canpos+9;
ARTheaders[_xcanpos].Length = strlen(Data->Canpos+9);
ARTheaders[_xcanpos].Found = 1;

vp->iov_base = Data->Canpos;
vp->iov_len  = strlen(Data->Canpos);
size += (vp++)->iov_len;

end = vp;
vp->iov_base = NL;
vp->iov_len  = 1;
size += (vp++)->iov_len;

vp->iov_base = Data->Body;
vp->iov_len  = &Article->Data[Article->Used] - Data->Body;
size += (vp++)->iov_len;
Data->SizeValue = size;
(void)sprintf(Data->Size, "%ld", Data->SizeValue);
Data->SizeLength = strlen(Data->Size);
HDR(_bytes) = Data->Size;
ARTheaders[_bytes].Length = Data->SizeLength;
ARTheaders[_bytes].Found = 1;

sprintf(buffer, "+%010ld\n", size);
if( write(can->fd[WRITE], buffer, 12)<0
|| xwritev(can->fd[WRITE], iov, vp - iov) < 0) {

   if( errno!=ENOSPC )
   {   i = errno;
       syslog(L_ERROR, "%s cant write %s %m", LogName, can->Name);
       IOError(WHEN, i);

       return -1;
   }
   else
       syslog(L_TRACE, "cant write %s, creating new can", can->Name);

   ftruncate(can->fd[WRITE], can->Bytepos[WRITE]);
   CANcreate(can,(((time_t)(Delete/CANTIME))+1)*CANTIME);/* !!! */

   sprintf(buffer, "%s!%010ld", can->Name, can->Bytepos[WRITE]);
   Data->Canpos=COPY(buffer);

   if( write(can->fd[WRITE], buffer, 12)<0
   ||  xwritev(can->fd[WRITE], iov, vp - iov) < 0) {
       i = errno;
       syslog(L_ERROR, "%s cant write %s %m", LogName, can->Name);
       IOError(WHEN, i);

       return -1;
   }
}

can->Bytepos[WRITE]+=size+12;

/* Need the header data? */
if (!NeedHeaders)
   return 0;

/* Figure out how much space we'll need and get it. */
(void)sprintf(bytesbuff, "Bytes: %ld\n", size);
for (i = strlen(bytesbuff), vp = iov; vp < end; vp++)
  i += vp->iov_len;

if (Headers.Data == NULL) {
   Headers.Size = i;
   Headers.Data = NEW(char, Headers.Size + 1);
}
else if (Headers.Size <= i) {
   Headers.Size = i;
```

```
            RENEW(Headers.Data, char, Headers.Size + 1);
        }

        /* Add the data. */
        BUFFset(&Headers, bytesbuff, strlen(bytesbuff));
        for (vp = iov; vp < end; vp++)
            BUFFappend(&Headers, vp->iov_base, vp->iov_len);
        Data->Headers = &Headers;

        return 0;
    }
```

### 4.2.1 Zugriff auf die Dosen

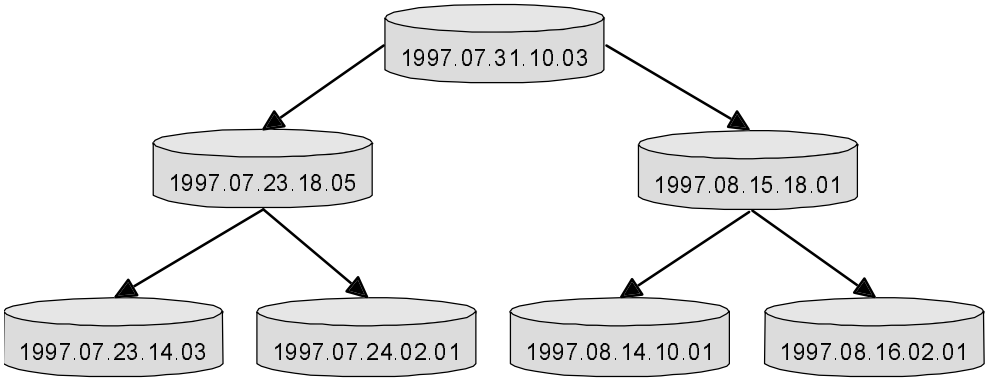
Der **innd** durchsucht beim Start das Verzeichnis `/var/spool/news/articles` nach Directory-Einträgen die mit „shelf." beginnen und überprüft alle darin befindliche Dateien. Files deren Namen in das Schema "YYYYMMDDHHXX" passen, werden als Dosen identifiziert und genauer überprüft, indem die ersten Zeilen gelesen werden. Wenn die erste Zeile mit dem 'Magic' übereinstimmt und die zweite Zeile eine bekannte Versionsnummer ergibt (derzeit 1.0) wird die Datei als Dose akzeptiert und vom INN in einem AVL-Baum[6] gespeichert.

Unter der Annahme, daß 1000 Dosen eine praktische Obergrenze an möglichen Dateien darstellt (siehe oben), bedeutet dies eine maximale Verzweigungstiefe von 11 um auf eine beliebige Dose zuzugreifen. Nachdem der AVL-Baum auch nur einmal pro Start des **innd** erstellt werden muß und maximal nur alle 100 MB News bzw. alle 4 Stunden eine Umbalancierung notwendig ist, wurden AVL-Bäume einer Hashtable vorgezogen. Im Zweifelsfall besteht die Abhängigkeit auf den AVL-Bäumen nur aus wenigen Zeilen Code, die sehr einfach durch entsprechende Hash-Aufrufe ersetzt werden könnten.

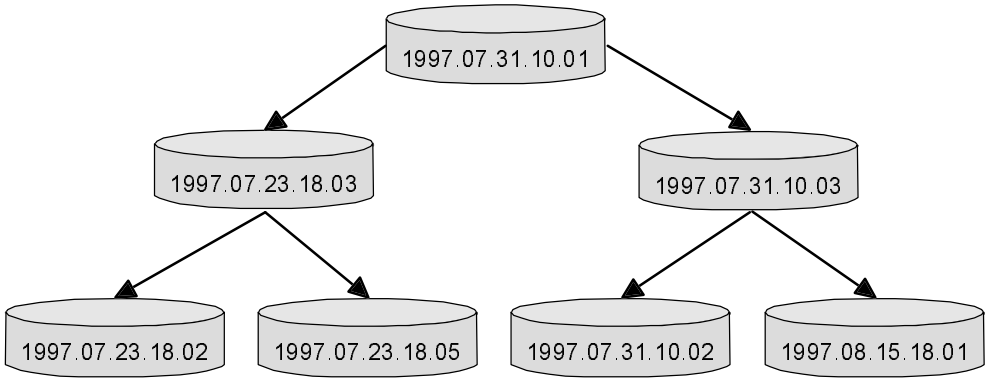
Der **innd** verwaltet über die Dosen zwei AVL-Bäume: Einen zum Beschreiben der Dosen und einen zweiten zum Lesen.

Auf die Dosen des Schreib-Baumes wird per `time_t` zugegriffen (also die übliche UNIX-Zeit in Sekunden seit 1970) und liefert immer die aktuelle Dose des entsprechenden Zeitraumes zurück. Falls beispielsweise ein Artikel in zwei Tagen gelöscht werden soll, schaut der **innd** im Schreib-Baum nach der Dose 'Now'+86400 und bekommt dann z.B. die Dose `shelf.02/199707231605` geliefert, also die 5. Dose von jenen, die am 23. Juli 1997 um 16:00 Uhr gelöscht werden sollen.

[6] Selbstbalancierende Binärbäume. Diese Methode der Datenspeicherung hat den Vorteil, daß auf die Daten sehr schnell beim Suchen zugegriffen werden kann, hat dagegen aber den Nachteil, daß das Einfügen und Löschen von Daten relativ langsam ist. Die AVL-Bäume wurden nicht selbst implementiert, es wurde auf eine Library von Paul Vixie zurückgegriffen, die wiederum auf Sourcecode von Niklaus Wirth beruht.




Auch der lesende AVL-Baum wird über das Delete-Datum indiziert, allerdings werden in diesem Fall auch noch die Dosennummern mit berücksichtigt, da Cancels und andere Lesezugriffe in allen Dosen und nicht nur in der zuletzt beschriebenen auftreten können.




Einen direkten Zugriff über den Dosendateinamen wurde deswegen nicht realisiert, damit sich Dosennamen prinzipiell während der Laufzeit des INN aus Reorganisationsgründen der Platten ändern könnten, auch wenn dies vom restlichen System derzeit nicht unterstützt wird. Außerdem war es so sehr einfach möglich, bestimmte Dosen schon frühzeitig zu löschen und damit trotz nun ungültigem Namen immer noch die entsprechende Dose im Speicher und AVL-Baum verwalten zu können.

Nachdem der **innd** nun auf alle Dosen innerhalb kürzester Zeit lesend und schreibend zugreifen kann, ist es nun einfach die ankommenden Artikel entsprechend der eingestellten Expirezeiten in die verschiedenen Dosen einzusortieren und bei Bedarf auch wieder einzulesen.

### 4.2.2 Lesen und Schreiben auf die Dosen

Die einfachste Vorgehensweise wäre bei jedem Artikel nach der entsprechenden Datei zu suchen, sie zu öffnen, den Artikel am Ende der Datei anzuhängen, den Header der Dose zu aktualisieren und die Datei wieder zu schließen. Nachdem

Öffnen und Schließen relativ teure Systemoperationen sind, würde diese Vorgehensweise kaum Geschwindigkeitsvorteile gegenüber dem ursprünglichen Verfahren „pro Artikel eine Datei" bringen.

Eine andere einfache Vorgehensweise wäre, alle Dateien gleichzeitig offen zu halten und erst beim Shutdown des **innd** die Dateien wieder zu schließen. Dies ist nicht möglich, da das Betriebssystem nur eine bestimmte Anzahl von gleichzeitig offenen Dateien pro Prozeß verwalten kann.

Zur Lösung dieses Problems wurde ein Algorithmus eingesetzt, der als „Second Chance-Variante von First In - First Out"-Algorithmus bekannt ist und aus der Betriebssystemlehre im Bereich virtuelles Speichermanagement stammt: Der **innd** verwaltet eine (kurze) Liste von gleichzeitig geöffneten Dateien. Jedes Mal wenn eine neue Datei geöffnet werden muß, wird die am längsten schon offene Datei geschlossen (reiner FIFO). Nachdem die am längsten offene Datei aber nicht unbedingt die am wenigsten benötigte Datei darstellt, wird der Algorithmus etwas modifiziert:

Bei jedem Zugriff auf eine Datei wird ein Flag gesetzt, daß die Datei verwendet wurde. Beim Öffnen einer neuen Datei wird bei der ältesten schon offenen Datei untersucht, ob ihr Schreibflag gesetzt ist. Wenn ja, wurde sie in letzter Zeit verwendet und ist deswegen u.U. nicht unbedingt der beste Kandidat um geschlossen zu werden. Deswegen wird ihr Schreibflag gelöscht und die Datei als jüngste erneut einsortiert (Second Chance). Danach wird die nächste Datei untersucht, solange bis eine Datei gefunden wird, die seit dem letzten Dateiwechsel nicht mehr zugegriffen wurde. Falls auf alle Dateien zugegriffen wurde, degeneriert der Algorithmus zum reinen FIFO, ansonsten macht der Algorithmus nichts, als die älteste seit dem letzten Wechsel nicht mehr benutze Datei zu schließen und die nun benötigte Datei zu öffnen.

In der aktuellen Implementierung kann für Schreib- und Lesezugriffe getrennt die Anzahl an offen gehaltenen Dateien eingestellt werden.

Die Datenstruktur für die Dosen sieht dementsprechend wie folgt aus:

```
typedef struct _ARTCAN {
    char     Version;        /* up to now only 1 */
    time_t   Delete;         /* expire */
    char     *Name;          /* .../shelf_xxx/199704232205 */
    int      Number;         /* in this example 5 */
    int      fd[2];          /* filedescriptor */
    long     Bytepos[2];     /* position of filedescriptor */
    char     AccessFlag[2];  /* used in last time? */
    char     dirty;          /* not flushed */
} ARTCAN;
```

Die nächste Seite zeigt die für das Öffnen und Schließen der Dosen zuständige Routine `CANdescriptorCycle()`, die regen Gebrauch von den Datenstruktur der Dosen macht:

```
int
CANdescriptorCycle(rwFlag, can)
    int         rwFlag;
    ARTCAN      *can;
{
    struct      stat statBuf;
    int         canCheck;
    int         ocp;

    do
    {   if( ++openCansPos[rwFlag]==maxOpenCans[rwFlag] )
            openCansPos[rwFlag]=0;

        ocp=openCansPos[rwFlag];
        if( openCans[rwFlag][ocp]==NULL )
            openCans[rwFlag][ocp]=can;

        if( openCans[rwFlag][ocp]->AccessFlag[rwFlag] )
            openCans[rwFlag][ocp]->AccessFlag[rwFlag]=0;
        else
        {   if( openCans[rwFlag][ocp]->fd[rwFlag] )
                CANcloseCan(rwFlag, openCans[rwFlag][ocp]);

            openCans[rwFlag][ocp]=can;

            if( rwFlag==READ )
            {   if( (can->fd[rwFlag]=open(can->Name, O_RDWR, 0666))<0 )
                {   if( CANdelete(can->Name)>time(NULL) )
                        syslog(L_ERROR, "cant open %s %m", can->Name);

                    can->AccessFlag[READ]=FALSE;
                    can->fd[rwFlag]=0;
                    return -1;
                }
            }
            else
            {   if( (can->fd[rwFlag]=open(can->Name, O_RDWR, 0666))<0 )
                {   syslog(L_ERROR, "cant open %s %m", can->Name);
                    exit(-1);
                }
            }
            if( (canCheck=CANcheck(can->fd[rwFlag]))!=0 )
            {   if( rwFlag!=READ || canCheck!=-3 )
                {   switch( canCheck )
                    {   case -1: syslog(L_ERROR, "can %s corrupt, wrong magic",
                                                 can->Name);
                                 break;
                        case -2: syslog(L_ERROR, "can %s corrupt, unknown version",
                                                 can->Name);
                                 break;
                        case -3: syslog(L_ERROR, "can %s corrupt, wrong size,
                                                 unclean shutdown?", can->Name);
                    }
                    return canCheck;
                }
            }
            if( rwFlag==READ )
                can->Bytepos[READ]=47;
            else
            {   fstat(can->fd[rwFlag], &statBuf);
                lseek(can->fd[rwFlag], statBuf.st_size, L_SET);
                can->Bytepos[rwFlag]=statBuf.st_size;
            }

            can->AccessFlag[rwFlag]=TRUE;

            if( rwFlag==READ )
                syslog(L_TRACE, "opening can %s (read)", can->Name);
            else
                syslog(L_TRACE, "opening can %s, starting at %ld",
                                can->Name, can->Bytepos[rwFlag]);
        }
    } while( openCans[rwFlag][ocp]->AccessFlag[rwFlag]==0 );
    return 0;
}
```

Bei 8 gleichzeitig zum Schreiben geöffneten Dateihandels zum Schreiben wurde im praktischen Betrieb keine nennenswerten Öffnungs- und Schließaktivitäten festgestellt. Das Lesen findet über eine größere Anzahl von Dateien statt, deswegen ist auch bei 12 Handels immer noch regelmäßig eine Open-Close-Aktivität zu verzeichnen, diese bleibt aber bei den Testsystemen noch im vertretbaren Rahmen (im Schnitt ein Wechsel pro Minute).

### 4.2.3 Warum keine Channels?

Mit Entwurf und Implementierung des gerade genannten Zugriffsystems auf die Dosen wurde ein sehr ähnliches Verfahren des INNs auf seine Out.going-Files per Channels zuzugreifen nochmals neu implementiert. Die Gründe die Artikel nicht auch in Channels zu schreiben waren folgende:

- Channels existieren nur innerhalb des **innd** selbst, alle Backends wie **batcher**, **innxmit** und **nnrpd** greifen auf die Artikel über das Filesystem direkt zu, also mußte für sie ein entsprechendes System sowieso realisiert werden:
- Channels sind zwar prinzipiell so programmiert, daß ein Channel gleichzeitig lesen und schreiben kann, verfügen aber über keinen Zugriff inmitten von Dateien. Somit sind sie für den lesenden Zugriff auf Dosen ungeeignet.
- Es wurde davon ausgegangen, daß der parallele Betrieb zweier Descriptorschleifen, eine für geringere Datenmengen (Channels) und eine für größere Datenmengen (Artikel) im Endeffekt größere Durchsatzzahlen ermöglichen als eine Schleife für beide zusammen

### 4.2.4 Das neue History-Format

Zu Beginn wurde aus Gründen der Kompatibilität zu Programmen, die Statistiken aus den Daten der Datei `history` erstellen, so wenig wie möglich Änderungen am Datenformat der Datei vorgenommen.

Es zeigte sich allerdings, daß die zusätzliche Länge von mindestens 30 Bytes pro Artikel zu einem problematischen Größenzuwachs der Datei führte. Dies ist insbesondere deswegen ein Problem, weil die `dbz`-Library nichtverwendete Bits der Hash-Pointer für andere Zwecke verwendet und dadurch einen elementaren Geschwindigkeitszuwachs erlangt.

Das neue Format der history-Datei besteht deswegen nur noch aus der Message-ID, dem Zeitpunkt des Eintreffens der Artikel und der Canpos, an der der Artikel zu finden ist. Somit ist die durchschnittliche Länge einer Zeile beim neuen history-Format sogar ca. 10% kürzer als dies bisher der Fall war.

Ein Beispiel einer aktuellen history sieht wie folgt aus:

```
<33E869CA.64E8@bluewave.co.uk>  870871177       shelf.01/199708081801!0007697054
<33e86ae2.2425268@news.erols.com>       870871177       shelf.01/199708081801!0007698523
<rod-060897074020001@ix.netcom.com>     870871177       shelf.01/199708081801!0007699542
<01bca302$46031340$2d654cd1@afcanny.nz> 870871177       shelf.01/199708081801!0007704445
<33e86a24.2235612@news.erols.com>       870871177       shelf.01/199708081801!0007705221
<cancel.33E764D6.3AAC@usedtel.com>      870871177       shelf.01/199708070603!0002970693
<33E8915E.5042@NOSPAM.marine.usf.edu>   870871177       shelf.01/199708081801!0007706248
<5s9opv$r801@raptor.site.se>    870871177       shelf.01/199708081801!0007708039
<33E8C6EF.3E20@netvigator.com>  870871177       shelf.01/199708081801!0007709929
<33E86813.41C6@cdsp.neu.edu>    870871177       shelf.01/199708081801!0007711009
```

## 4.3 Die Backends

Nach Implementierung der oben genannten Speichermethoden und Algorithmen stellte sich beim **innd** die erhoffte Geschwindigkeitssteigerung ein. Der etwas erhöhte Verwaltungsaufwand wurde durch sehr viel weniger Plattenzugriffe mehr als nur aufgehoben.

Dies änderte sich leider dramatisch, als die Overview-Database eingeschaltet wurde und die wichtigsten Informationen über alle Artikel getrennt gespeichert wurden.

### 4.3.1 overchan

Eine Analyse des (kleinen) Programmes **overchan**, welches vom **innd** die dazu notwendigen Daten erhält und diese dann abspeichert, konnte den Grund für den Performance-Einbruch schnell erklären:

**Overchan** erhält von jedem einsortierten Artikel die wichtigsten Informationen (in welchen Newsgruppen der Artikel erscheint, wer, wann, welche Länge und mit welcher Message-ID den Artikel geschrieben hat und auf welche anderen Artikel sich dieser Artikel bezieht) und speichert diese Informationen dann ab.

Dazu hat die Overview-Database pro Newsgruppe eine Datei, in der alle Overview-Daten von gerade vorhandenen Artikeln am Stück gespeichert sind. Praktisch bedeutet dies, daß bei einem Crossposting eines Artikels in die drei Gruppen „comp.sys.next.misc, de.comp.sys.next, maus.sys.next" drei Dateien geöffnet werden, Overview-Daten angefügt und die Dateien wieder geschlossen werden. Wenn der nächste Artikel ein Reply auf den vorherigen Artikel ist, werden wieder die drei Dateien geöffnet und entsprechende Overview-Daten angefügt.

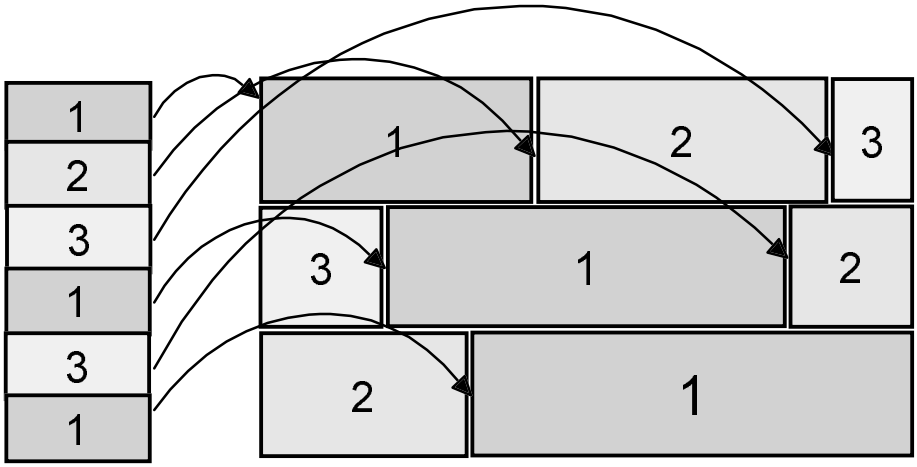
Die oben genannten Methoden, um möglichst selten Dateien öffnen und schließen zu müssen, wurden also durch die Overview-Database zunichte gemacht, da hier für jeden ankommenden Artikel weiterhin eine (oder bei Crosspostings sogar mehrere) Dateien geöffnet, beschrieben und wieder geschlossen werden mußten. Ein einfaches Abschalten der Datenbank war leider keine Lösung des

Problems, da sie elementar für das Auffinden der einzelnen Artikel einer Newsgruppe in den Artikeldosen vorgesehen war.

Auf der anderen Seite war es aber auch nicht möglich einfach viele Overview-Dateien offen zu halten, da dies erst bei mehreren hundert parallel offenen Dateien Wirkung gezeigt hätte und so viele Filehandles nicht zur Verfügung gestanden hätten.

Als Lösung wurde folgende Vorgehensweise entwickelt: Das Programm **overchan** wird mit einem Puffer[7] ausgestattet, in welchem Overchan-Daten im RAM zwischengespeichert werden.

Dabei zeigt ein Array auf die einzelnen Einträge, so daß beim späteren Auslesen des Puffers schnell auf die overview-Zeilen zugegriffen werden kann.



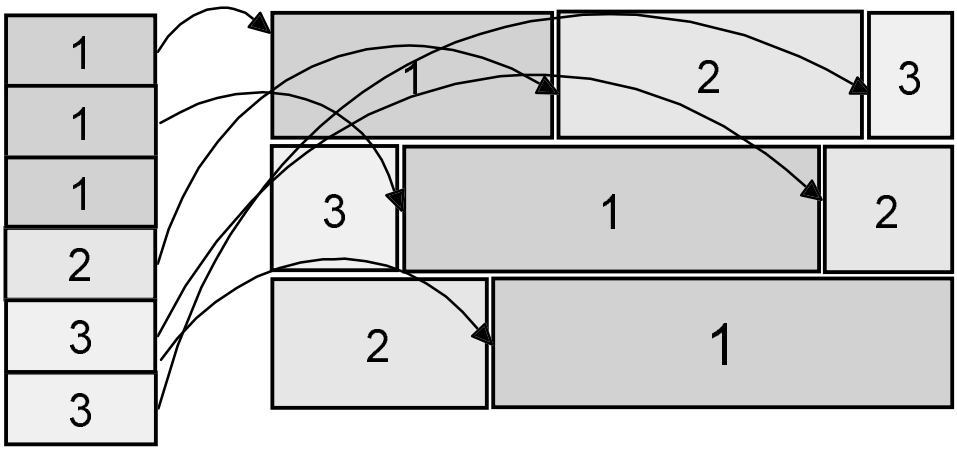

Jedes Mal wenn

- der Puffer voll ist, oder
- Daten von 5000 Artikeln gesammelt wurden, oder
- wenn eine Minute nicht mehr gespeichert wurde

werden die gepufferten Daten nach Newsgruppen sortiert und dann alle Overchan-Daten einer Newsgruppe mit einem open()/close()-Zugriff auf einmal abgespeichert, woraufhin erneut Artikeldaten zwischengepuffert werden.

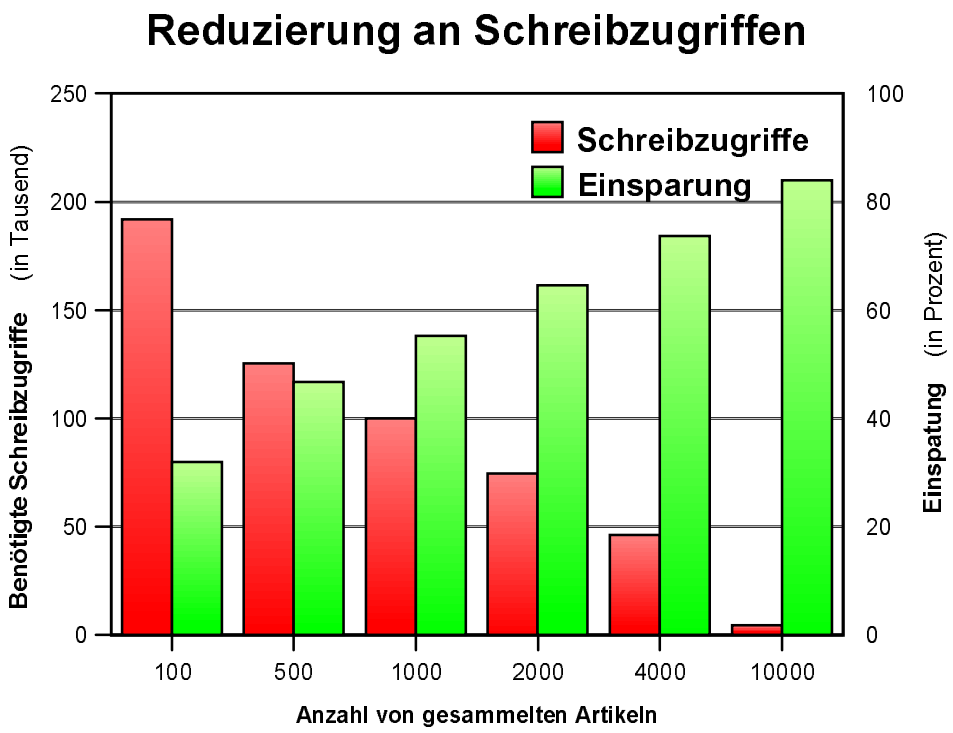
[7] In der aktuellen Implementierung handelt es sich dabei defaultmäßig um 5 MBytes.

Der Puffer bleibt dabei unverändert und nur die Pointer des Arrays müssen verschoben werden.




Bei einer statistischen Gleichverteilung der Artikel über alle Newsgruppen würde dieses Verfahren erst bei einer Puffergröße Erfolg zeigen, die größer als die Gesamtzahl an Newsgruppen auf dem System ist. Die Praxis zeigt aber, daß durch diese Technik auch bei Newssystemen mit mehr als 10000 Newsgruppen und einer Puffergröße von 5 MBytes bzw. maximal 5000 Artikel noch im Schnitt deutlich über 50% der Zugriffe eingespart werden.

Das folgende Diagramm zeigt eine nicht repräsentative Statistik auf news.bawue.de über die Verteilung von 280000 Artikeln in der Reihenfolge ihres Eintreffens auf die verschiedenen Newsgruppen:




Die 280000 Artikel wurden in Blöcke verschiedener Größe eingeteilt (100, 500, 1000, ...) und dann untersucht, auf wieviel verschiedene Newsgruppen die Artikel innerhalb eines Blocks zusammenfallen.

So verteilen sich z.B. 100 nacheinander auf dem System eingetroffene Artikel im Schnitt auf 70 verschiedene Newsgruppen, d.h. der optimierte **overchan** kann 30 Dateizugriffe einsparen. Bei 1000 zusammengesammelten Artikeln befinden sich diese im Schnitt in 447 verschiedenen Newsgruppen, womit der **overchan** schon ca. 55% der Zugriffe einsparen kann.

Dies liegt daran, daß gerade nicht von einer gleichmäßigen Verteilung der News auf alle Newsgruppen ausgegangen werden kann, sondern viele Gruppen so gut wie gar keinen Traffic erfahren, wohingegen auf andere Newsgruppen (maßgeblich die Gruppe Control[8]) große Anteile der Newsartikel entfallen.

[8] Dies liegt an den Cancel-Robots, siehe Kapitel 2.6 auf Seite 15.

**Anmerkung:** Probleme bei der Implementierung von overchan

Das Programm **overchan** macht beim Schreiben der sortierten Overview-Zeilen regen Gebrauch der INN-Library-Funktion `xwritev()`, eine Funktion welche intern auf die Betriebssystemfunktion `writev()` aufsetzt.

Dieser Funktion `writev()` ermöglicht es mit einem einzigen Aufruf eine ganze Reihe von verschiedenen Speicherbereichen gleichzeitig abzuspeichern. Leider herrscht in der UNIX-Welt keine genaue Regelung dafür, wie viele Speicherbereiche mit einem Aufruf maximal abgespeichert werden können.

Die `xwritev()`-Funktion des INN ruft deswegen die interne Betriebssystemfunktion auf, wertet aus der Antwort des Systems aus, wie viele Bereiche tatsächlich gleichzeitig geschrieben werden konnten und wiederholt dann mit den verbliebenen Speicherbereichen den Systemaufruf so oft, bis alles geschrieben wurde oder aber der `xwritev()` wegen einer unbekannten Fehlermeldung abbrechen muß.

Im Falle der modifizierten Implementierung des **overchan** trat diese Meldung leider regelmäßig auf, da hier im ungünstigen (bzw. in Bezug auf Performancegewinn sehr günstigen) Fall bis über 1000 verschiedene Overchan-Zeilen in eine Datei zusammen geschrieben werden konnten und demnach dem `xwritev()` über 1000 Speicherbereiche gleichzeitig übergeben wurden. Dies führte regelmäßig zum Verlust von Overchan-Zeilen und dadurch auch zu nicht mehr zugreifbare Artikel im System.

In verschiedenen #include-Files auf diversen Maschinen wurde ein nicht standardisiertes und immer wieder anders benanntes `#define` gefunden, welches als Obergrenze für den `writev()`-Aufruf eine Zahl von 16 Speicherbereichen nennt. Diese Zahl wird nun berücksichtigt und dem `xwritev()` nie mehr als 16 Overchan-Zeilen gleichzeitig übergeben.

### 4.3.2 Batcher

Ursprünglich wurden die News von Rechner zu Rechner per UUCP in sog. *Batches* übertragen, d.h. viele Artikel wurden zu einer großen Datei (Batch) zusammengefaßt, die Datei dann verschickt und auf der Zielmaschine wieder entpackt. Obwohl die Newsbatches heute nur noch selten eingesetzt werden, wurde das Programm **batcher** (welches die Batches erstellt) an das neue Newssystem angepaßt.

Der Batcher bekommt als Parameter bei Programmstart eine Datei genannt (out.going-Datei), in der alle Newsartikel aufgelistet sind, die zu einem (oder bei entsprechender Anzahl auch mehreren) Batch(es) zusammengefaßt werden sollen. Das bisherige Format ist auf Seite 18 beschrieben.

Bei der klassischen Methode Newsartikel zu speichern, war es für den Batcher einfach jeden Artikel über den Newsgruppennamen und die Artikelnummer als

File zu öffnen, zu lesen und die Datei wieder zu schließen. Bei der neuen Methode ist dies nur noch über Umwege möglich.

Nachdem die out.going-Datei vom **innd** erstellt wird, war es einfach dem **innd** in Zukunft die Dose und Byteposition innerhalb der Dose anstatt der Artikelnummer angeben zu lassen:

```
shelf.01/199708081801!0007821305 1117
shelf.01/199708081801!0007822434 866
shelf.02/199708161801!0000635424 1510
shelf.01/199708081801!0007827714 973
shelf.01/199708081801!0007832800 955
shelf.01/199708081801!0007833767 1266
shelf.02/199708161801!0000636946 5070
shelf.02/199708161801!0000642028 1582
shelf.02/199708161801!0000643622 944
shelf.01/199708071801!0033560077 1524
shelf.01/199708081801!0007837054 2661
shelf.01/199708081801!0007839727 1437
shelf.02/199708161801!0000644578 1895
```

Erreicht wird dies dadurch, daß die Konfigurationsdatei newsfeeds mit einem neuen Flag ausgestattet wurde, welches statt eines relativen („p") oder absoluten Pfades („P") die Position von Artikeln relativ zu den Dosen ausgibt. Dementsprechend sehen die (schon oben vorgestellten) Einträge für das Newsfeeds-File nun so aus:

```
phranjep/phranjep.lb.bawue.de:\
      comp.infosystems.www*,\
      *next*,\
      de.comp.os.unix,\
      stgt\
      :Tf,Wcb:phranjep
```

Der Batcher öffnet in Zukunft jeweils die entsprechende Dose, liest 12 Bytes ein, um zu überprüfen ob er wirklich am Anfang eines Artikels ist (Format "+0000000000\n"), und ob dieser nicht schon wieder gecancelt wurde (anstatt des „+" ein „-" vor der Artikellänge) liest nachfolgend die Anzahl an Bytes ein, die der Artikel lang ist und schreibt ihn dann komplett wieder in den gerade bearbeiteten Batch.

Diesen Zugriff kann man durch einen einfachen Trick noch deutlich beschleunigen: Der Batcher schließt nach dem Lesen eines Artikels die Dose nicht sofort wieder, sondern wartet ab, was für ein Artikel als nächstes zu batchen ist. Falls dieser Artikel sich zufälligerweise in der gleichen Dose befindet, kann er sich das Öffnen sparen und weitermachen, falls der Artikel innerhalb der Dose unmittelbar auf den letzten Artikel folgt, muß nicht einmal der Filehandel verschoben werden.

Diesen „Zufall" kann man sehr einfach zu einem Regelfall machen, indem das out.going-File aufsteigend sortiert wird, bevor man es dem Batcher übergibt. Damit es der Befehl **sort** von UNIX einfach hat, werden deswegen die Artikelpositionen innerhalb der Dose auch mit führenden Nullen angegeben, wodurch

keine numerischen Operationen durchgeführt werden müssen, sondern einfach Strings verglichen werden können.

Im praktischen Einsatz ist der **batcher** durch dieses Vorgehen so schnell, daß die Festplatten-Performance unrelevant ist und nur noch die CPU-Geschwindigkeit für den Komprimierer (gzip) der limitierende Faktor ist. Ohne gzip kann der **batcher** auf durchschnittlicher PC-Hardware deutlich über 700 Artikel pro Sekunde lesen und nach /dev/null pipen.

### 4.3.3 innxmit

Heute werden nur noch selten News per Batches verteilt. Der größte Teil an Usenet-News wird zwischen den Rechnern im Internet per NNTP ausgetauscht. Dies ist ein Protokoll, welches (u.a.) vor dem Versenden eines Artikels erst anfragt, ob der andere Rechner an diesem Artikel interessiert ist. Dazu werden zuerst nur die Message-IDs von Artikeln verschickt und das andere System kann dann entscheiden, ob es diesen Artikel schon (von einem anderen System) bekommen hat, oder ob es ihn noch benötigt. Bei größeren Newsservern ist üblich, sich von mindestens zwei anderen Systemen mit News versorgen zu lassen, um dadurch über eine gewisse Redundanz zu verfügen.

Die Änderungen sind äquivalent zu denen beim **batcher**. Neuer `newsfeeds`-Eintrag

```
bawue/news.bawue.de:\
      *\
      :Tf,Wcm:
```

und neues Ausgabeformat:

```
shelf.01/199708152201!0004553730 <5s7inm@sparcserver.lrz-muenchen.de>
shelf.01/199708152201!0004558890 <33E74C7@photo.verm.tu-muenchen.de>
shelf.01/199708152201!0004560678 <33e74db8.0@news.rz.uni-passau.de>
shelf.01/199708152201!0004563078 <1997Aug5.@amylnd.s.bawue.de>
shelf.01/199708202201!0000625854 <33E6363A.7217A931@hotmail.com>
shelf.02/199708071402!0008157153 <33e7d519.2442495@news.bbs-la.com>
shelf.02/199708071402!0008330752 <33E6C032.38B4A499@pacbell.net>
shelf.02/199708071402!0008411076 <MPG.e4d4af1522bb@127.36.9.81>
shelf.02/199708071402!0008605774 <5s6lca@lightning.cise.npl.co.uk>
```

Nachdem der Austausch von News per NNTP sehr viel komplizierter ist als das einfache Zusammenfassen von Artikeln beim batchen, ist das Programm zum Verschicken von News per NNTP **innxmit** sehr viel komplexer aufgebaut als das schon vorgestellte Programm **batcher**:

So hat man z.B. das Problem, daß große Newsartikel relativ viel Zeit benötigen bis sie vollständig über das Netzwerk übertragen sind, das vollständige Laden solcher großen Artikel in den Hauptspeicher würde unnötig für lange Zeit Speicherressourcen belegen. Modernere Versionen von **innxmit** übertragen zudem mehrere Artikel gleichzeitig (streaming NNTP), um dadurch die Leitungen besser auszulasten. Diese innxmit-Versionen müßten dann dementsprechend mehrere

Artikel gleichzeitig im Speicher halten, was bei vielen gleichzeitigen NNTP-Verbindungen zu unterschiedlichen Rechnern auch große Mengen an Hauptspeicher benötigen würde.

### 4.3.4 Die QIO-Library

Es gibt deswegen beim INN eine Library namens QIO[9], welche das Problem wie folgt löst: Ein Programm öffnet eine Datei mittels der neuen Funktion `QIOopen(filename)`. Danach kann auf die Datei per `QIOREAD()` zugegriffen werden, wobei bei jedem Aufruf die Library eine Zeile der Datei zurück gibt. Intern liest die Library zu Beginn einen Block von 8 KBytes in einen Puffer ein und gibt dann pro `QIOREAD()`-Aufruf immer eine Zeile aus diesem Puffer zurück. Sobald der Puffer abgearbeitet wurde, ließt QIO die nächsten 8 KBytes in den Puffer, so lange, bis die Datei vollständig geladen wurde.

Der Vorteil dieses Verfahrens ist, daß 8 KBytes eine Datenmenge ist, die man auch von vielen verschiedenen Dateien gleichzeitig im Speicher halten kann, ohne deswegen unnötig viel Hauptspeicher zu belegen, auf der anderen Seite aber 8 KBytes genügend viel Speicher ist, um die Festplatte nicht unnötig mit vielen kleinen Leseanforderungen zu belasten.

Programme wie **innxmit** greifen nun auf die Artikel über die QIO-Library zu. Dies ist bei der modifizierten Version des INN mit Artikeldosen nicht mehr ohne weiteres möglich, da mehr als nur ein Artikel in einer Datei gespeichert ist, die QIO-Library allerdings weder in der Mitte einer Datei mit dem Lesen anfangen kann, noch das Ende eines Artikel bemerkt, sondern bis zum tatsächlichen Ende der Datei weiterlesen würde.

### 4.3.5 Die QCI-Library

Es wurde deswegen eine neue Bibliothek namens QCI[10] entwickelt, welche vom Aufruf her mit der QIO-Library identisch ist, intern aber die Artikeldosen verwaltet.

Ähnlich wie der **innd** verfügt die Library über einen AVL-Baum von schon einmal geöffneten und über eine Ringliste von gerade geöffneten Dosen, so daß auch per Second-Chance-FIFO so selten wie möglich Dateien geöffnet und geschlossen werden müssen.

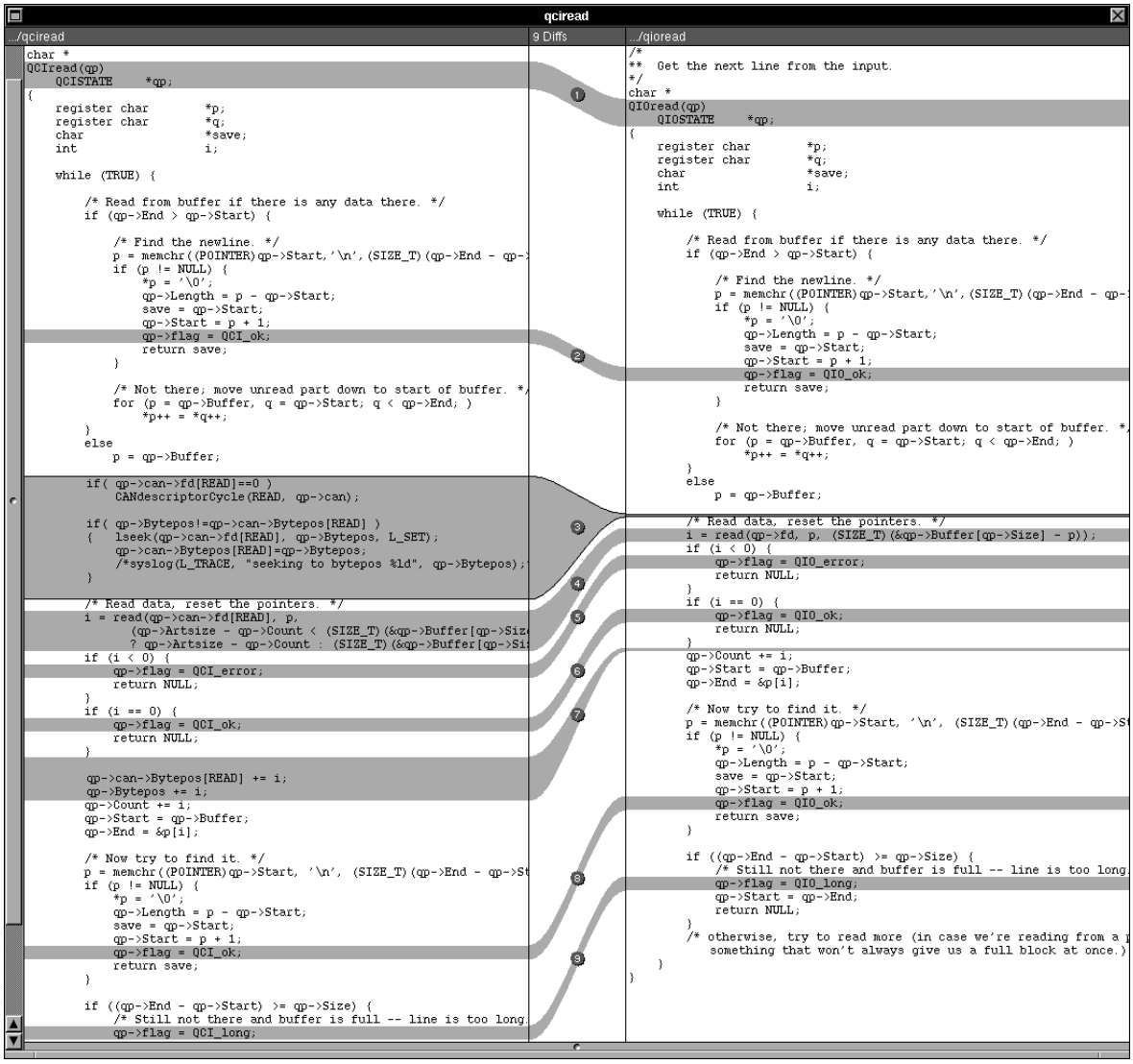
Auf diesem Wege können in Zukunft auch andere Backends des INN sehr einfach an das neue Speichersystem angepaßt werden, falls diese auch auf die Artikel per QIO-Library zugreifen. Im günstigsten Fall müssen dazu nur die entsprechenden Library-Aufrufe durch QCI-Aufrufe ersetzt werden.

---

[9] Quick Input Output - dies ist in soweit komisch, als daß die Library nur lesen und gar nicht schreiben kann

[10] Quick Can Input - schreiben kann diese Library auch nicht

Ein kurzer Ausschnitt aus dem Sourcecode der QCI-Library im direkten Vergleich zur ursprünglichen QIO-Libaray zeigt den Umfang und Größe der notwendigen Veränderungen:




### 4.3.6 nnrpd

Auch die eigentlichen Leser von Usenet-News holen sich die Artikel mit ihren Newsreadern über das NNTP-Protokoll. Der **innd** startet dazu bei jedem Connect von einem Rechner, der nicht als Newsfeed eingetragen ist, ein externes Programm, welches den Newsreader mit Artikeln versorgt, genannt **nnrpd**.

Moderne Newsreader öffnen oft mehrere NNTP-Verbindungen gleichzeitig, um mehrere Artikel parallel zu übertragen und um den Delay des Netzwerkes auf ein Minimum zu reduzieren. Auch werden die NNTP-Verbindungen oft schon nach

der Übertragung eines einzigen Artikels wieder abgebaut und dadurch der entsprechende **nnrpd** beendet.

Beim Protokoll NNTP besteht die Möglichkeit sich von einer beliebigen Newsgruppe alle vorhandenen Artikel anzeigen zu lassen, d.h. eine Liste von Nummern, unter denen die Artikel auf dem Newssystem abgespeichert sind. Dazu greift der **nnrpd** einfach auf das entsprechende Directory im hierarchischen Dateibaum zu und läßt sich alle Dateien auflisten. Dateien, deren Namen nur aus Nummern bestehen, werden als Artikel erkannt und dem Newsreader zum Lesen angeboten.

Im Falle der Artikeldosen ist der einfache Zugriff auf das Directory nicht mehr möglich. Die einzelnen Artikel sind weit über verschiedenen Dosen auf verschiedenen Regalen verstreut, deren Positionen nur von den Ankunftszeiten der Artikel und den (damals) aktuellen Expireeinstellungen abhängig sind.

Glücklicherweise existiert aber noch die oben beschriebene Overview-Database, in welcher pro Newsgruppe eine Datei gehalten wird, in der alle Artikel einer Newsgruppe in Kurzform beschrieben sind. Dies macht sich der modifizierte INN zunutze, indem er beim Einsortieren eines Artikels in eine Dose der Overview-Database zusätzlich zu den üblichen Daten auch noch einen weitere Information, genannt `Xcanpos:` übergibt, nämlich die genaue Position des Artikels bestehend aus Regal, Dose und Byteposition .

Der **nnrpd** liest also beim Aufruf einer Newsgruppe nicht mehr die Verzeichniseinträge, sondern gleich die dazugehörige Overview-Datei. Beim eigentlichen Abruf eines Artikels kann dieser sehr einfach geladen werden, indem über den neuen Eintrag `Xcanpos:` der Overview-Datei die Artikelposition bestimmt wird und der Artikel mit der oben schon beschriebenen QCI-Library geladen wird.

## 4.4 Administrative Programme

### 4.4.1 Expire

Der Expire kann in unserem Fall einfacher behandelt werden als dies beim Original möglich war - große Teile des Codes wurden in den **innd** verschoben und müssen deswegen hier nicht nochmals verwendet werden.

Von jedem Artikel wird in der history gespeichert, in welcher Dose er abgelegt ist. Nachdem am Dosennamen eindeutig zu erkennen ist, wann die Dose (und damit auch der Artikel) zu löschen ist, kann die Funktion des Expires sehr vereinfacht werden:

Der **expire** durchsucht die Datei `history` nach lauter Artikeln, die in Dosen gespeichert sind, deren Verfallsdatum überschritten ist. Entsprechende Einträge werden auf die Message-ID und den Ankunftszeitpunkt gekürzt. Gleichzeitig werden nach schon bisher eingesetzten Regeln und Prozeduren Message-IDs von schon expireten Artikeln gelöscht (siehe Seite 15). Zum Schluß wird ein neuer Hash-Index der `history` erstellt und Artikeldosen mit überschrittenem Verfallsdatum richtig gelöscht.

Es wäre möglich die Performance des **expireover** dadurch zu beschleunigen, indem man vor dem Löschen der Dosen diese nochmals nach den Artikeln durchsucht würde um dadurch die Datei `expire.rm` zu erstellen. Im Moment wird darauf allerdings verzichtet.

### 4.4.2 Expire.rm bzw. fastrm

Die Programme **expire.rm** bzw. **fastrm** werden nicht mehr eingesetzt. Das Löschen der (wenigen) Dosen erledigt das Programm **expire** selbst, so daß zusätzliche Programme dafür unnötig sind. Entsprechende Zeilen im Script **news.daily** wurden deswegen auskommentiert.

### 4.4.3 Expireover

Das Programm **expireover** ist zum Löschen von Artikeln in der Overview-Datenbank bestimmt. Über die Funktionalität des Generierens einer Datenbank aufgrund des tatsächlichen Artikelbestandes verfügt das Programm hiermit nicht mehr, dies macht in Zukunft das neue Programm **makeoverview**. Die Funktionalität des Löschens von Artikeleinträgen anhand einer Datei `expire.rm` bleibt aber weiterhin wie gehabt erhalten, so daß später mittels einer Datei `expire.rm` Artikel schneller gelöscht werden können.

Im Moment findet der Abgleich zum tatsächlichen Datenbestand dadurch statt, daß das Programm **expireover** alle `.overview`-Dateien nach `Xref:`-Einträgen durchsucht, die auf Dosen mit überschrittenen Verfallsdatum

zeigen und diese dann löscht. Die dabei erziehlte Geschwindigkeit ist befriedigend: Das Abarbeiten einer 900 Mbytes großen Overview-Database dauerte auf einem unbelasteten PC ca. 30 Minuten. Im täglichen Einsatz bei gleichzeitigem Einsortieren von Artikeln und Aktuallisieren der Overview-Database benötigt **expireover** ca. 45 Minuten.

### 4.4.4 news.daily

Nachdem der Expire durch den Wegfall des Löschens einzelner Artikel elementar schneller geworden ist, kann er mehrmals am Tag durchgeführt werden, um eine gleichmäßigere Auslastung der Platten zu erreichen.

Das Scannen und Rotieren der Logfiles, das auch durch das Script **news.daily** ausgeführt wird, ist allerdings weiterhin nur einmal täglich sinnvoll. Im cronjob vom User news wurden deswegen zwei Zeilen eingefügt, von denen eine den ursprünglichen täglichen Ablauf von **news.daily** durchführt und eine zweite die zusätzliche Expires abhandelt.

```
5 2             * * * /usr/local/news/bin/news.daily
5 6,10,14,18,22 * * * /usr/local/news/bin/news.daily notdaily nomail
```

### 4.4.5 makehistory

Das Programm **makehistory** ist am täglichen Betrieb von News nicht beteiligt. Es wird aber benötigt, um bei einem Crash des Systems, bei welchem die history-Database beschädigt wird, eine neue erstellen zu können.

Die ursprüngliche Version durchsuchte dazu die Datei `active` nach gültigen Newsgruppen und öffnet pro Newsgruppe das dazu zuständige Directory im Newsspool-Dateibaum, um alle Dateien darin zu öffnen, zu lesen und wieder zu schließen. Ein Durchlauf von **makehistory** dauert dadurch in der Regel mehrere Stunden und wird nur im äußersten Notfall durchgeführt. Außerdem werden Artikel in einer vor kurzem gelöschten Newsgruppe, deren Artikel noch nicht vollständig expiret sind, nicht gelöscht und verbleiben damit auf unbegrenzte Zeit im Spool-Bereich.

Die modifizierte Version von **makehistory** weicht vom bisherigen Vorgehen ab: Es werden nicht mehr einzelne Newsgruppen gescannt, sondern nacheinander alle Artikeldosen. Die Geschwindigkeit des Programms ist dadurch elementar höher: Obwohl das Programm nicht weiter optimiert wurde, können auf einem durchschnittlichen PC innerhalb weniger Minuten ein Gigabyte Newsartikel neu erfaßt werden.

# 5 News in Dosen im Praxiseinsatz

## 5.1 news.bawue.de

Nach längeren Probeläufen auf einem Entwicklungs- und Testsystem wurde die modifizierte Form des INN zuerst auf dem Vereinsrechner des BaWue-Net-Vereins installiert und getestet.

### 5.1.1 Die Testumgebung

Bei der news.bawue.de handelt es sich um einen üblichen INTEL-PC, bestehend aus einem 90 MHz-Pentium, 128 MBytes RAM, zwei SCSI-Adaptern und mehreren Fast-SCSI-Festplatten. Als Betriebssystem wird FreeBSD2.1 eingesetzt.

Der Rechner verwaltet die wichtigsten Dienste des BaWue-Nets, u.a. die Newsversorgung, den WWW-Server, einen Proxy-Cache unter Squid, mehrere Mailinglisten und den DNS-Server des Vereins. Von diesen Diensten belasten die Maschine allerdings nur die Usenet-News nennenswert. Vor der Installation der News in Dosen hatte die Maschine Mühe mit dem täglichen Newsaufkommen zurecht zu kommen.

Leider waren die Bedingungen unter denen die Auswertungen stattgefunden haben alles andere als Ideal: Mehrmals mußten noch Verbesserungen an der Newssoftware vorgenommen werden und die Newsversorgung war zum Ende der Auswertungsphase hin sehr ungleichmäßig. Trotzdem ist es gelungen mit dem vorhandenen Datenmaterial einige interessante Erkenntnisse zu sammeln.

### 5.1.2 Die Usenet-Situation

Der BaWue-Net-Verein erhält seine News fast vollständig vom Newsserver des Landesforschungsnetzes Baden Württemberg , der news.bewue.de.

Nachdem der Verein nur über einen Feed verfügt und die Internetanbindung stark belastet ist, findet die Newsversorgung nicht per NNTP, sondern per komprimierte gzip-UUCP-Batches über TCP/IP statt. Um zu den Spitzenzeiten des interaktiven IP-Verkehrs die Leitung noch weiter zu entlasten, werden im Normalfall zwischen 20:00 abends und 1:00 nachts keine News übertragen. Dies führt ab 1:00 Uhr nachts zu heftigem Newsaufkommen, was erst nach mehreren Stunden wieder nachläßt.

Für die folgende Diagramme bedeutet dies, daß die Zahlen von der Netzwerkanbindung entkoppelt sind und Schwankungen der Leitungsauslastung sich nicht auf die Messungen auswirken.

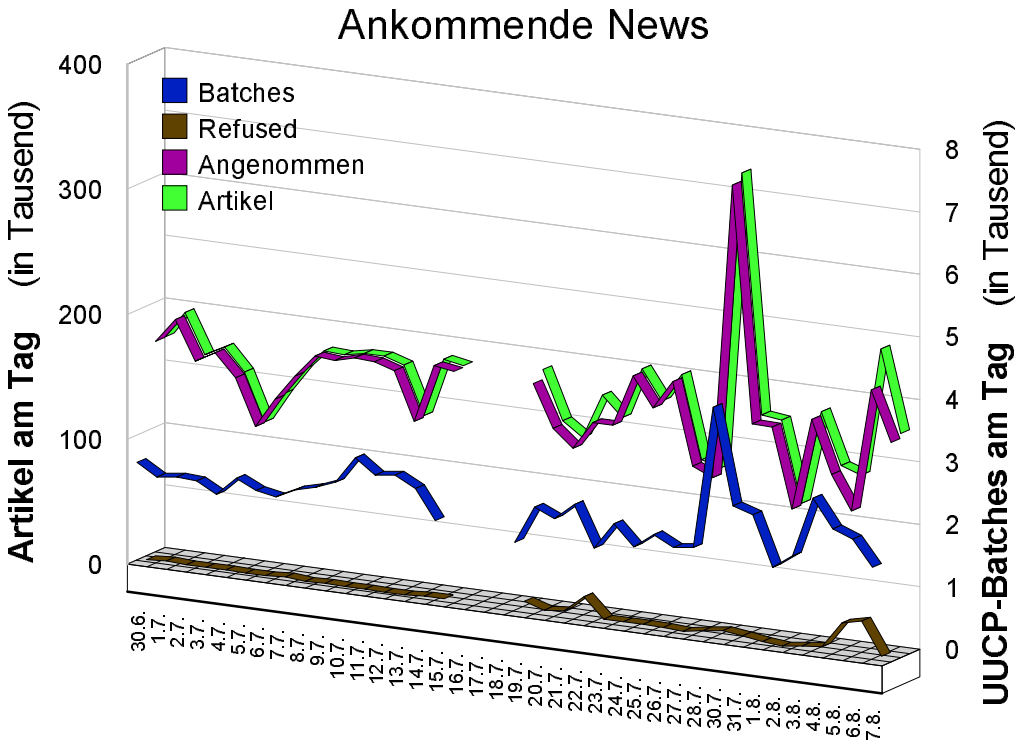
Ankommende News werden vor allem mehrere Systeme weiter gebatcht bei denen dann die eigentlichen Nutzer des BaWue-Nets angeschlossen sind.

Ansonsten existieren noch ein paar Systeme, die mit News per NNTP versorgt werden, im Gesamtaufkommen der News aber ebenso wie interaktive Newsreader nicht weiter ins Gewicht fallen. Dementsprechend konzentrieren sich die folgenden Auswertungen.

## 5.2 Newsaufkommen

Das aktuelle Newsaufkommen besteht entsprechend dem Angebot der news.belwue.de derzeit aus den „Big-8" und den üblichen deutschen Ergänzungen[11], Newsartikel die nur in alt.binaries.* gepostet werden, sind im BaWue-Net aus Kapazitätsgründen auch nicht abonniert, trotzdem sind durch entsprechende Crosspostings auch reichlich Binary-Postings vorhanden.

Das erste Diagramm zeigt das Newsaufkommen auf news.bawue.de im Monat Juli.




Man sieht, daß das durchschnittliche Newsaufkommen kurz vor 200000 Artikel am Tag liegt, wobei an den Wochenenden deutlich weniger News übertragen

[11] Die news.belwue.de transportiert aus diversen Gründen eine Reihe gut bekannter Newsgruppen nicht.

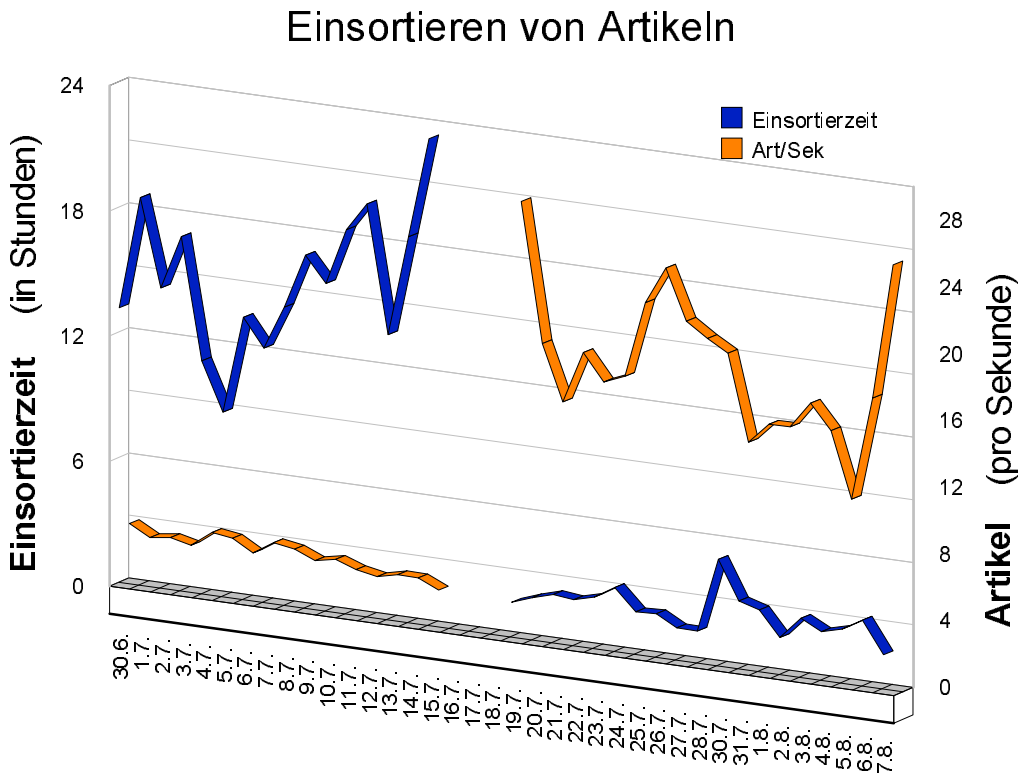
werden und am Ende der Meßperiode ein starker Peek von über 350000 Artikel stattfindet.

Im Zeitraum zwischen dem 16. und 18. Juli liegen keine Daten vor. In dieser Zeit wurde das System umgestellt und auch der alte Artikelbestand teilweise erneut eingespielt, so daß kein brauchbares Zahlenmaterial über diesen Zeitraum vorhanden ist.

Unter Berücksichtigung dieses Newsaufkommens können nun die Veränderungen durch das neue Newssystem betrachtet werden.

## 5.3 Einsortieren von Newsartikeln

Das folgende Diagramm zeigt die Veränderungen in Bezug auf das Einsortieren der ankommenden Newsbatches. Es wurde sowohl die Einsortiergeschwindigkeit der Artikel pro Sekunde gemessen, als auch die insgesamt benötigte Zeit des Systems um die täglich ankommenden News zu verarbeiten.




Man sieht deutlich, daß die Geschwindigkeit des neuen Systems elementar zugenommen hat.

Anstatt im Schnitt 15 Stunden am Tag Newsbatches zu verarbeiten, ist die Maschine in der Regel 3 Stunden am Tag mit Entpacken beschäftigt. Vor der Umstellung wurden unter 4 Artikel pro Sekunde einsortiert, mit der neuen Technik sind bei gleicher Hardware nie weniger als 12 Artikel pro Sekunde verarbeitet worden und der Durchschnitt liegt bei ca. 20 Artikel.

Trotzdem sind große Schwankungen der Geschwindigkeit deutlich sichtbar, die näher untersucht werden mußten. Mehrere Vermutungen für die Gründe der Schwankungen konnten ausgeräumt werden:

- Zu Beginn verursachte noch ein Programmierfehler das Vollaufen der Festplatten, da beim Expire nicht genügend Dosen freigegeben wurden. Nach der Behebung des Fehlers stand zwar wieder genügend Plattenplatz zur Verfügung, die Performance änderte sich allerdings nicht.
- Nachdem die verschiedenen Regale unterschiedliche Größe hatten und die eingestellten Expire-Zeiten nicht die vorhandene Kapazität der Platten ausnützte, verschoben sich im Laufe der Zeit die Newsdosen hauptsächlich auf das größte Regal. Es wurde vermutet, daß durch die ungleichmäßige Belastung der Platten der Performance-Abfall zu begründen sei.

  Versuchsweise wurde deswegen kurzfristig das größte Regal künstlich belegt und dadurch eine Reaktivierung der anderen Festplatte erreicht. Allerdings waren die erzielten Geschwindigkeitssteigerungen enttäuschend.
- Es wurde festgestellt, daß die Overview-Database durch einen anderen Programmierfehler nur zum Teil die expireten Einträge wieder löschte. Bis der Fehler entdeckt wurde, war die Datenbank auf das Fünffache der normalen Größe angewachsen.

  Auch die Behebung dieses Fehlers führte zu keiner Verbesserung der Performance.

Durch weitere Tests und Auswertungen wurde festgestellt, daß die Maschine unter Berücksichtigung der sonst laufenden Dienste mit Speichermangel zu kämpfen hatte. Gleichzeitig war das `history`-File auf über 130 MBytes angewachsen und demnach größer als der vorhandene Hauptspeicher. Es entstand der Verdacht, daß der **innd** beim Überprüfen der `history` in kam.

Diese Vermutung wurde noch verstärkt, als am 1. August ein testweises Verkleinern des `history`-Files auf 80 MBytes (durch Herabsetzen des Parameters „remember") eine Geschwindigkeitssteigerung von mehreren Artikeln pro Sekunde brachte.

Ein anderer Indiz war, daß schon zuvor am 24. Juli beim Testen des Programms **makehistory** eine neue (kurze) history-Datei auf der Maschine angelegt worden war und daraufhin die Performance kurzfristig wieder in die Höhe schnellte.

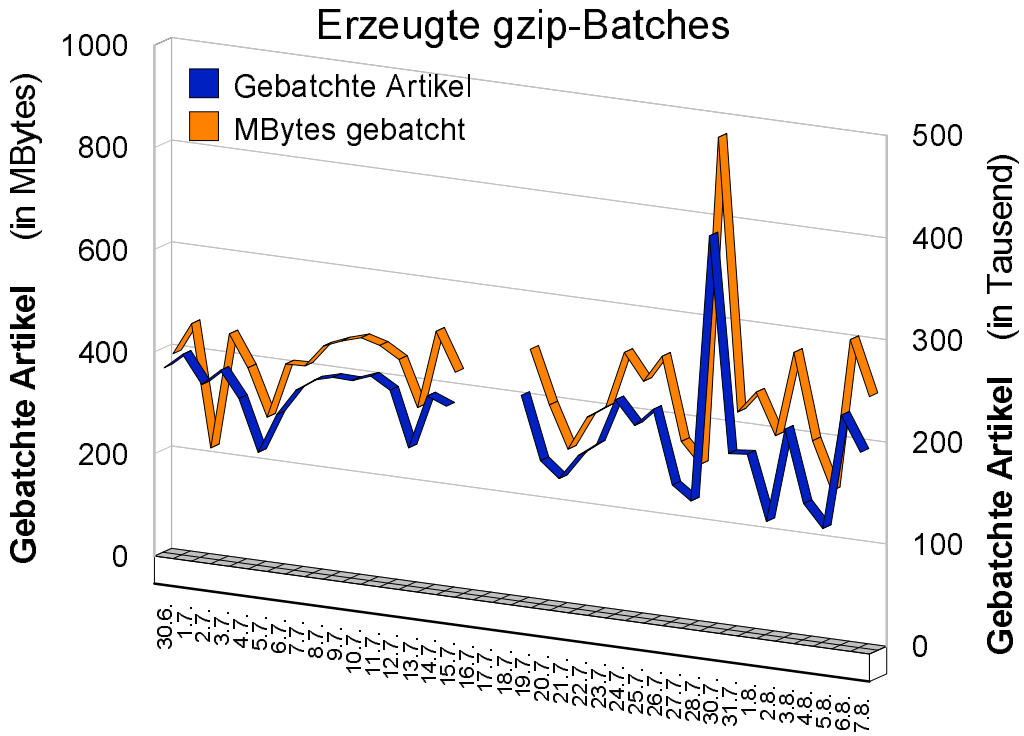
Nach diesen Untersuchungen wurde am 4. August das Datenformat des `history`-Files geändert (siehe Kapitel 4.2.4, Seite 33) und eine neue `history` angelegt. Das Diagramm zeigt sehr deutlich, daß daraufhin die erhofften

Geschwindigkeitssteigerungen eintraten und wieder über 25 Artikel pro Sekunde vom **innd** verarbeitet werden konnten.

Es bleibt abzuwarten, ob mit Anwachsen des neuen history-Files die Geschwindigkeit des Systems wieder abfallen wird. Aufgrund des neuen Formates wird die Länge der Datei bei gleichem Artikelbestand deutlich kürzer ausfallen und somit das Problem nur in abgeschwächter Form auftreten, trotzdem sollte im hier vorliegenden Falle mittelfristig einer Aufstockung der Hardware erfolgen.

## 5.4 Batchen von Newsartikeln

Eine andere Auswertung war die Überprüfung der abgehenden News per UUCP. Die verarbeiteten Datenmengen entsprachen proportional den ankommenden News:

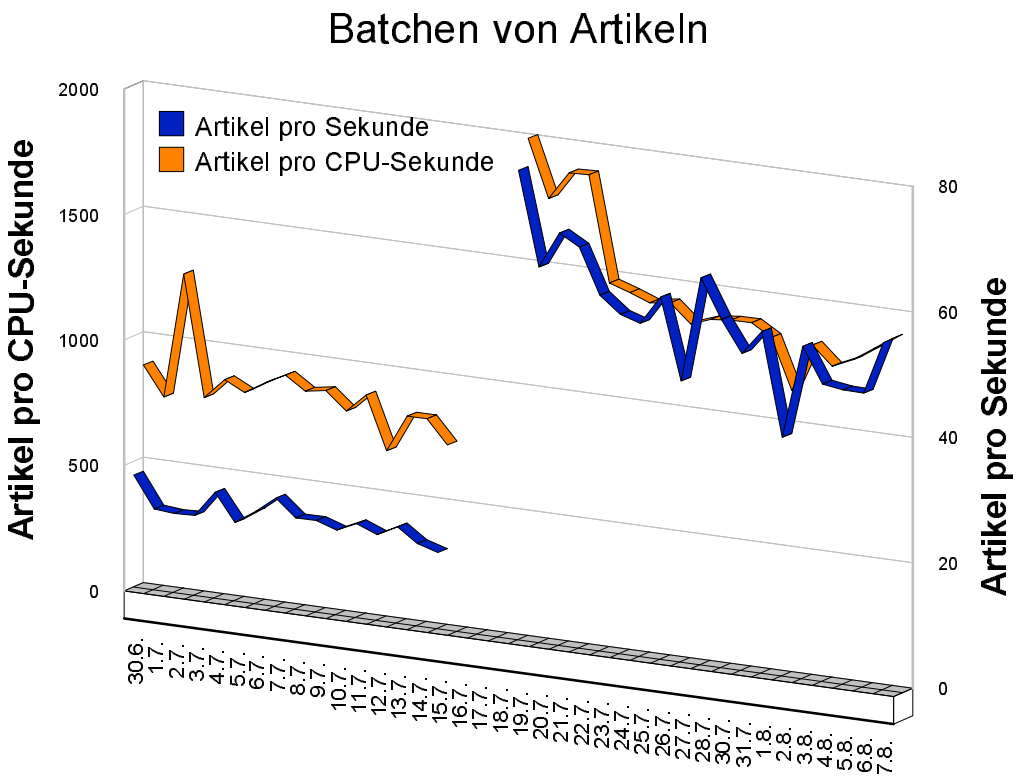



Auch hier sind wieder sowohl die Wochenenden mit weniger Traffic, als auch die Peeks zum Ende der Auswertungszeit zu beobachten. Insgesamt ist die Kurve mit dem neuen System noch unruhiger als dies vorher der Fall war.

Dies liegt vermutlich daran, daß das neue System durch die insgesamt höhere Geschwindigkeit unregelmäßig ankommende Newsmengen ebenso auch wieder weiterschickt und im Gegensatz zum alten System keine dämpfende Wirkung eintritt.

Zudem hatten die anderen Sites Probleme mit dem großen Peek vom 30.7. zurecht zu kommen, wodurch sich News auf news.bawue.de anhäuften und nur unregelmäßig weitergeschickt wurden.

Auch beim Batchen wurde die Geschwindigkeit des Systems gegenüber früher verbessert:



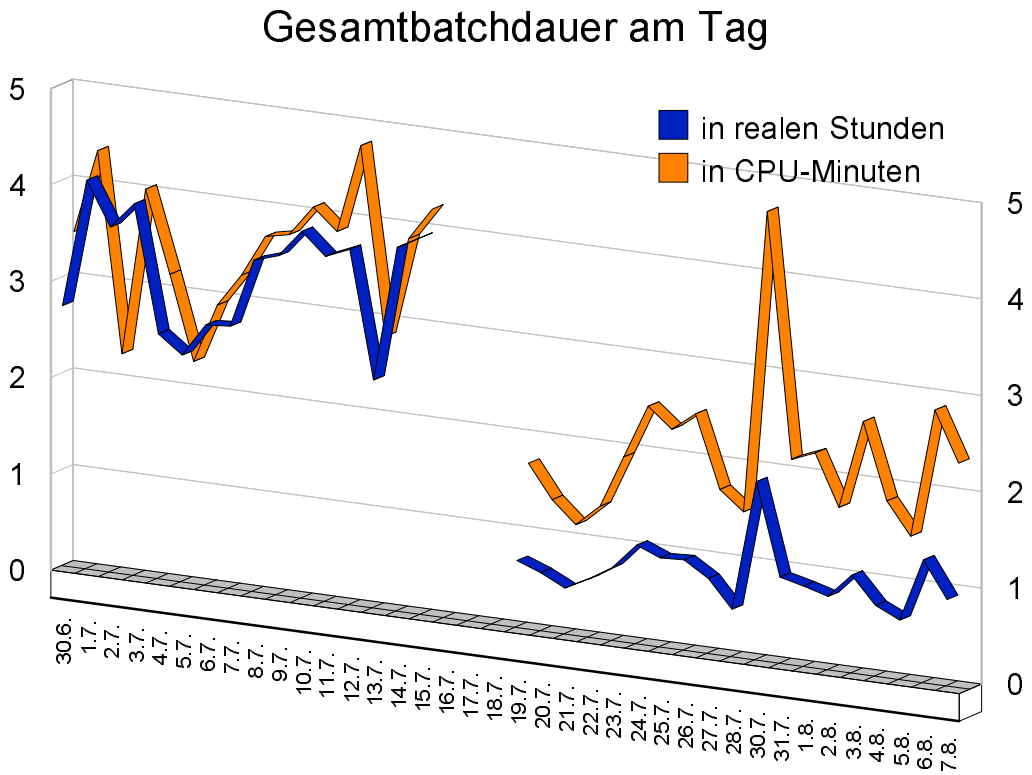

Sowohl die Anzahl der Artikel pro CPU-Sekunde, als auch die Anzahl der Artikel pro realer Sekunde ist deutlich gestiegen.

Der Unterschied zwischen diesen beiden Werten ist die Zeit, in der das Programm **batcher** zwar läuft, aber auf Platten-IO oder den Komprimierer wartet und deswegen keine Rechenzeit verbraucht.

Der Grund für die geringere CPU-Belastung kann man mit der Verringerung an `open()-close()`-Funktionsaufrufen erklären, die allgemeine Beschleunigung durch die höhere IO-Geschwindigkeit des **batchers** im Allgemeinen.

Eine andere interessante Darstellung der **batcher**-Performance ist die Aufschlüsselung in „Stunden am Tag", im Gegensatz zur obigen Aufschlüsselung nach

„Artikel pro Sekunde”:




Auch hier sieht man deutlich, daß die Geschwindigkeit zugenommen hat und der **batcher** mit dem neuen System nur noch ca. 1 Stunde am Tag mit dem Laden und Komprimieren von Artikeln beschäftigt ist, im Gegensatz zu durchschnittlich früher 3 Stunden.

Daß die Beschleunigung nicht noch deutlicher ausfällt, liegt darin begründet, daß beim neuen System der **batcher** hauptsächlich auf den Komprimierer **gzip** warten muß und nicht mehr auf die Platten-IO. Eine schnellere CPU als der derzeit eingesetzte Pentium 90 würde deutliche Steigerungen bringen, was aber bei der derzeitigen Belastung nicht unbedingt erforderlich ist.

## 5.5 Weitere Statistiken

Leider liegen sowohl über **innxmit**, als auch über den **nnrpd** keine aussagekräftigen Auswertungen vor:

Die verarbeiteten Datenmengen im Falle von **innxmit** sind auf dem verwendeten System so gering, daß sich darüber keine Schlüsse ziehen lassen. Beim **nnrpd** wurde bis vor wenigen Tagen noch mit diversen Programmproblemen gekämpft, so daß die Zugriffszahlen vor und nach der Umstellung unbrauchbar stark differieren, um daraus noch Geschwindigkeitsunterschiede beurteilen zu können.

# 6 Fazit

*News in Dosen* kann programmtechnisch als Erfolg gewertet werden.

Die Performanceunterschiede zu einem herkömmlichen INN sind so deutlich, daß man mit vorhandener Hardware mindestens zwei Jahre länger auskommen kann oder aber bei Neukauf von Hardware mit halbierten Kosten zu rechnen ist.

Nachdem die Platten-IO beim nun vorliegenden System nicht mehr der limitierende Faktor ist, muß in anderen Bereichen des INN Optimierungen vorgenommen werden:

- Der benötigte Hauptspeicher ist ein ernstzunehmendes Problem. `history`-Files mit einer Gesamtlänge von mehreren hundert Mbytes sind keine Seltenheit mehr und Belasten die Maschine deutlich.
- Pro Newsreaderanfrage einen eigenen **nnrpd** zu starten ist sowohl speichertechnisch, als auch im Bezug auf den Betriebssystemoverhead problematisch und mittelfristig ohne Zukunft. Ein multi-threaded **nnrpd** sollte hier Abhilfe schaffen können.
- Der ganze Sourcecode sollte überarbeitet werden. An vielen Stellen kommen aus Robustheits- bzw. Kompatibilitätsgründen bei Konfigurationsfehlern des Systems Fallbackroutinen zum Einsatz, ohne daß dies dem Administrator durch eine entsprechende Fehlermeldung gemeldet wird. Dadurch wird unnötig Rechenzeit vergeudet und die Fehlersuche ist sehr schwierig.

Insgesamt muß die Frage erlaubt sein, wie lange das Usenet in seiner bisherigen Form noch weiter funktionieren kann. Bis heute konnten die Entwicklungen im Hardwarebereich und der verbesserten Software mit dem stetig wachsenden Newsaufkommen mithalten. Ob dies mittelfristig so bleiben wird, ist mehr als fraglich.

# 7 Literaturverzeichnis


[1] Geoff Collyer, *News Need Not Be Slow*, Departmen of Statistics, University of Toronto

[2] Rich $alz, *Installing InterNetNews*, Open Software Foundation

[3] Andrew S. Tanenbaum, *Computer Networks (second edition)*, Prentice-Hall International, ISBN 0-13-166836-6

[4] Andrew S. Tanenbaum, *Betriessysteme, Entwurf und Realisierung Teil 1*, Prentice-Hall International, ISBN 3-446-15268-7

[5] Kernighan/Ritchie, *Programmieren in C*, Prentice Hall International, ISBN 3-446-15497-3

[6] David A. Curry, *UNIX Systems Programming for SVR4*, O'Reilly & Associates, Inc. ISBN 1-56592-163-1

[7] DNEWS: URL http://netwinsite.com/

[8] Cyclone: URL http://www.highwind.com/

[9] Diabolo: URL http://www.backplane.com/

[10] ISC: URL http://www.isc.org/

[11] Sonstiges: URL news://news.software.nntp/

## Abschließende Erklärung

Hiermit versichere ich, daß ich diese Studienarbeit selbstständig verfaßt und nur die angegebenen Hilfsmittel verwendet habe.

_____

Tobias Hennerich