UNIVERSITY OF OSLO
Department of Informatics

Handling Information Overload on Usenet

Advanced Caching Methods for News

Jan Ingvoldstad

Cand Scient Thesis

4th August 2001

Abstract

Usenet is the name of a world wide network of servers for group communication between people. From 1979 and onwards, it has seen a near exponential growth in the amount of data transported, which has been a strain on bandwidth and storage. There has been a wide range of academic research with focus on the WWW, but Usenet has been neglected. Instead, Usenet's evolution has been dominated by practical solutions.

This thesis describes the history of Usenet in a growth perspective, and introduces methods for collection and analysis of statistical data for testing the usefulness of various caching strategies. A set of different caching strategies are proposed and examined in light of bandwidth and storage demands as well as user perceived performance.

I have shown that advanced caching methods for news offers relief for reading servers' storage and bandwidth capacity by exploiting usage patterns for fetching or prefetching articles the users may want to read, but it will not solve the problem of near exponential growth nor the problems of Usenet's backbone peers.

Preface

When I first started my studies at the University in Oslo in the autumn of 1991, I thought I was going to be a mathematician, and followed my first class in university level mathematics with vigor. After my service in the Royal Norwegian Navy in 1992 to early 1993, I returned to the university in time for the second semester's start in August, not knowing which course to take next. An advisor said I should take the introductory course in informatics "because everybody needs it". This was interesting enough, but I immediately got hooked on the Internet, on e-mail, on FTP, on MUDs, and of course on Usenet.

I was an on-and-off student for several years, partially working for money, partially studying, and probably mostly writing e-mail, Usenet articles and playing MUD. When the time came to begin work on my degree in communications technology, I wondered, "*what* am I going to write about?" I really did not expect it to be Usenet, but I suspect others saw that coming, especially when I started administering a news server myself.

When I started thinking about Usenet, it did not take long before I saw that there was might be room for improvement. It developed into much more than an idea. I also saw that there was very little written about Usenet in the way I do.

Some of the later ideas have come thanks to the inspiration of other, more experienced people on Usenet, and this thesis would have been impossible without them. I have also had great help with minor programming issues, consistency checking and typo hunting. Thanks to Stig S. Mathisen at Nextra for statistics, and the system administrators at Ifi for letting me play with their reading server. Special thanks go to Knut Erik Borgen, Olav Andree Brevik, Kjetil T. Homme, Lars Syrstad, Magne Syrstad, and my advisor, Gisle Hannemyr.

Oslo, 4th August 2001.

Chapter Overview

1. INTRODUCTION
   Introduction to the thesis, followed by an overview of how Usenet works, its history, how it has developed and grown, challenges with how it is being distributed, ending with an introduction to the advanced caching solutions for handling some of the challenges.

2. METHODS USED
   How I went about writing this thesis.

3. PRIOR ART AND OTHER WORK
   Prior work with Usenet distribution, caching, proxying etc. that I am aware of.

4. DISCUSSION OF CACHING STRATEGIES
   Discussion and preliminary evaluation of the caching strategies introduced at the end of the first chapter.

5. FINDINGS
   Presentation and discussion of data collected at Nextra and Ifi.

6. CONCLUSION
   Results of the findings and outline of future work.

7. APPENDIX A
   Some sources for discussions and findings in the thesis, including material not readily available and software modifications.

8. APPENDIX B
   News articles used in the thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Usenet is the name of a world wide network of servers for group communication between people.

Since Usenet was created in 1979, it has seen an impressive growth from a small academic community to a network used by millions of people from a wide variety of backgrounds all over the world. The total size of the data flowing through Usenet has been more than tripling every year between 1993 and 2001.

This growth has not been without problems, and has raised significant challenges in how to handle the ever increasing volume of Usenet data flow. Very few are able to handle all of Usenet, and as the amount of users and data they produce increase, as do the challenges with having enough network bandwidth and storage capacity. Spending great sums of money on hardware components relieves the situation, but it does not solve it.

My motivation for this thesis was to find a way to reduce the problems we see today. I have introduced the idea of advanced caching methods as a general improvement for parts of the Usenet distribution network, as well as discussed other work that has been done to relieve network bandwidth and storage capacity. I also introduce methods for analyzing and evaluating caching strategies based on statistical data from news servers.

Advanced caching will be an improvement for those news servers with users that do not read every available news article, which goes for most if not all news servers with users. However, caching does not solve the problem of exponential growth. When the available technology no longer can support enough network bandwidth and storage capacity, this will limit itself.

In this chapter, I first provide an introduction to Usenet architecture and technology, followed by Usenet's history from the perspective of growth and the challenges of this growth, as well as a brief mention of some other trends and suggestions for dealing with the volume of Usenet data traffic. Towards the end of the chapter, in section 1.5.2, I present advanced caching strategies that may help handling these challenges. I then pose questions about what

way these methods may improve on Usenet.

To my knowledge, there are no peer reviewed sources for the growth of Usenet in a historical perspective or for caching of news in particular. The development of Usenet technology has been a community effort rather than an academic one, and many of the conventions and standards have been informal at first to be standardized later. The search for prior art is discussed in chapter 2 along with other methods such as data collection and interviews. The prior art that I found relevant is discussed in chapter 3.

In chapter 4, I discuss the strategies mentioned briefly in section 1.5.2 in more detail.

Chapter 5 shows and illustrates my findings from the data collection. The conclusion in chapter 6 discusses and concludes from these findings, and identifies several areas of future work.

I have attempted to structure and word the thesis for an audience that is not familiar with Usenet, its historical background, how it used to work, what the protocols are, or how it works today. It is an advantage to have some familiarity with the Internet, the WWW, e-mail and networks.

Readers familiar with how Usenet works, its history of growth, and the problems arising from it may want to skip the introductory chapter until section 1.5.

Users are presented as if they are male. I have no data that shows whether the typical user is male or female, so this is merely for my own convenience.

Definitions that I introduce are marked clearly, while Usenet specific terminology is explained as it is used with the terms *emphasized*.

1.1 What Is Usenet?

News is a distributed platform for group communication — mainly between humans — based on a network of servers all around the world. “Usenet” is an abbreviation for “Unix User Network”, but is also known under other names, specifically “NetNews”, simply “News” [Hardy, 1993] or “Usenet News”. News is a slightly misleading name for what Usenet is meant for: asynchronous communication between people, as opposed to news items distributed by mass media.

[Salzenberg et al., 1998] defines Usenet the following way:

> Usenet is the set of people who exchange articles tagged with one or more universally-recognized labels, called "newsgroups" (or "groups" for short). There is often confusion about the precise set of newsgroups that constitute Usenet; one commonly accepted definition is that it consists of newsgroups listed in the periodic "List of Active Newsgroups" postings which appear regularly in news.lists.misc and other newsgroups. A broader definition of Usenet would include the newsgroups listed in the ar-

> ticle "Alternative Newsgroup Hierarchies" (frequently posted to news.lists.misc). An even broader definition includes even newsgroups that are restricted to specific geographic regions or organizations. Each Usenet site makes its own decisions about the set of groups available to its users; this set differs from site to site.

This thesis uses the broader definition of Usenet, including the infrastructure behind it.

The communication between users is largely controlled by local administrators of the news service — the *news administrators* — at a news service provider (NSP). An NSP can also be a full Internet Service Provider (ISP).

While Usenet is today mostly a part of the Internet, using the same basic network protocols for communication between servers, it has been common to say that "Usenet is not the Internet". The reason for this is that the transport of news itself is not fundamentally dependent on the Internet; it just is the most used platform today.

There is much more to Usenet than I mention in this chapter, which is intended as an introduction and overview of what I consider relevant for understanding this thesis. Some parts have been simplified in order to avoid too much excruciating detail. I recommend turning to [Hauben and Hauben, 1995] for another historical perspective, [Tanenbaum, 1996] (section 7.5, pp. 669-680) for views based on computer networks, [Udell, 1998] for the groupware perspective, [Spencer and Lawrence, 1998] for administering Usenet systems, and [Spafford and Moraes, 1998] for a software history.

1.2 The Usenet Model

The Usenet News model has the following major aspects to consider:

- Message format
- Message distribution
- Message storage

The main flow of Usenet is commonly through the Internet, using the Network News Transfer Protocol (NNTP) [Kantor and Lapsley, 1986], a TCP[1] based protocol for transmission. Most Internet standards are described in RFCs[2], and the IETF[3] is working on several new standards. Usenet's standards are described in RFCs, but there are de facto Usenet standards not included in the RFCs, although the IETF is working on standardising these enhancements.

[1] Transmission Control Protocol
[2] Request For Comments
[3] Internet Engineering Task Force

Header	Description
Date	Alleged time posted
From	Alleged author and e-mail address
Newsgroups	Comma-separated list of target newsgroups
Message-ID	Unique article identifier
Path	News servers the article passed through
Subject	Subject/topic of discussion
Approved	E-mail address of moderator for moderated newsgroup
Control	Control article, see section 1.2.2 on page 12
Distribution	Hierarchies the article is distributed to
Expires	Suggested deletion date for the article
Followup-To	List of newsgroups followups are directed to
Keywords	Keywords identifying the article
Lines	Number of lines in the article body
Organization	Organization the author or originating host claims to belong to
References	References to former articles, list of Message-IDs
Reply-To	E-mail replies are wanted to this address
Sender	Alleged actual sender of the article
Summary	Brief summary of the article
Xref	List of newsgroups and corresponding article numbers

Table 1.1: News headers defined in RFC 1036
The top 6 headers, marked with **bold** text, are mandatory.

Communication is in the form of *articles*. An article is simply a single text message, authored by one or more entities, usually humans. Some mailing lists are automatically reposted to Usenet. Other automated postings are common.

In the case of human users, the author writes the article in his favorite text editor, and then sends it to a *newsgroup* (also simply called a *group*) on a *news server* (called the *injecting server*) using a specialized program for news, a *newsreader*. This process is called *posting an article*.

The text editor can be internal to the newsreader or vice versa, for instance MS Outlook Express and Gnus respectively. The newsreader program is also referred to as a *user agent* (UA) for consistency with similar e-mail and WWW terminology).

1.2.1 Message Format

The article format (RFC 1036, [Horton and Adams, 1987]) is based on that of Internet mail (e-mail) messages (RFC 822, [Crocker, 1982]).

Articles are logically divided into two separate parts, *head* (also called *headers*) and *body*. The headers contain meta-information about the article, such as who allegedly posted the article, from where, at what time, to which

```
From: Jan Ingvoldstad <jani+news@tsathoggua.rlyeh.net>
Subject: How do newsadmins deal with news traffic today?
Newsgroups: news.software.nntp
Date: 30 Apr 2001 13:03:01 +0000
Message-ID: <ygtlmoiwnei.fsf@tsathoggua.rlyeh.net>
Sender: jani@tsathoggua.rlyeh.net
Path: nntp.uio.no!uio.no!news.tele.dk!148.122.208.681!
        news2.oke.nextra.no!nextra.com!news.klingenberg.no!
        tsathoggua.rlyeh.net!not-for-mail
```

Figure 1.1: Some example news headers

newsgroups, with what subject of discussion, a unique message ID, and the path through which servers the article has been passed to avoid re-relaying to those servers. Other headers may be used, but these are not relevant here; I will discuss some of these when necessary. See table 1.1 on the preceding page and figure 1.1, required headers are in bold.

The article's body contains the actual message, which must be plain text, including quotations of former articles in the same discussion. Usually the author adds a *signature*, which contains information about the author, a quip, a quote from a book or movie, or all of these at the same time. This signature is considered part of the body.

A Note on the Path Header The Path header has a syntax from before the DNS[4] was created, and each news server identifier is separated by a "bang" – '!'. This ID is either a name registered in the UUCP[5] maps or, since the introduction of DNS, the full DNS name of the server. The identifier must only be in place for relaying servers where the article passed as a news article, so if it passes via e.g. an e-mail server, there should be no entry for that. The last entry is not considered part of the path entry, and is in the case of a user agent normally the *local part* (see RFC 822) of an e-mail address, and the "not-for-mail" entry is there in case it is difficult or impractical to supply that local part. With this last exception, it is supposed to be possible to send an e-mail to each entry in the path list, plus the local part after the path list.

1.2.2 Message Distribution and Storage

While the news article format is compliant with the Internet mail message format, news distribution is significantly different from mail distribution.

[4] Domain Name System

[5] Unix-to-Unix CoPy, a data copying protocol

Many mailreaders are also newsreaders[6], which causes some initial confusion for users on this issue.

News articles are commonly spread by a flooding algorithm between news servers, also known as *news feeders/feeding servers* or *peers*. Where each *downstream peer* gets a *newsfeed* of articles from their *upstream peer*. This is called a "pushed" stream, similar to the "push" technology used for WWW

The receiving servers reject articles they already have instead of requesting the ones they do not have. This is called a *pull stream*, like clients pulling documents off the WWW. Note that it is possible for the downstream peer to request articles from their upstream peer, but it is not commonly used.

So far, this is deceptively similar to Internet mail, which is also can be sent from server to server until finally received by the user's mailbox, although the current practice is to send e-mail directly to the server local to the receiving user. However, users do not get this feed directly in their own mailbox, as would be the case with Internet mail and mailing lists. Instead, their newsreader fetches a list of newsgroups and articles from the news server, using NNRP[7]. An illustration is shown in figure 1.3 on page 8. This kind of news server is called a *reader/reading server*. I will refer to this function as *reading server* from now on, in order to avoid confusion between human reader, newsreader program and reader server.

The user then chooses which newsgroups to read articles from from his newsreader's *subscription list*. This list of *subscribed* newsgroups is updated by the user. When the user has chosen a newsgroup, he then can choose which articles to read within that newsgroup.

Note on Built-in Filtering in Newsreaders Many newsreaders offer filtering methods based on patterns in article headers and body in the form of a so-called *kill file*. If an article matches this pattern, the newsreader will not download the article, and if it is already downloaded, it will not display it. Some newsreaders offer additional functionality in form of a *score file*. This is also a kind of filter, but unlike a kill file, the choice is not black or white. *Scoring* is more flexible, and allows the user to set positive or negative values for various patterns. These values are cumulative. In addition to setting values for patterns, the user specifies a score threshold for which articles should be displayed. This way, it is possible to e.g. ignore certain authors, unless they post with a subject the user finds more interesting (high score for the subject pattern) than he finds the author uninteresting or annoying (low score for the author pattern).

Articles are stored on these central news servers, making them shared as opposed to mailing lists, where each user effectively stores his own message copy in his mailbox. This does not prevent the user from downloading and

[6] MS Outlook Express and Gnus are such combined news- and mailreaders.

[7] Network News Reader Protocol, basically NNTP with a few changes

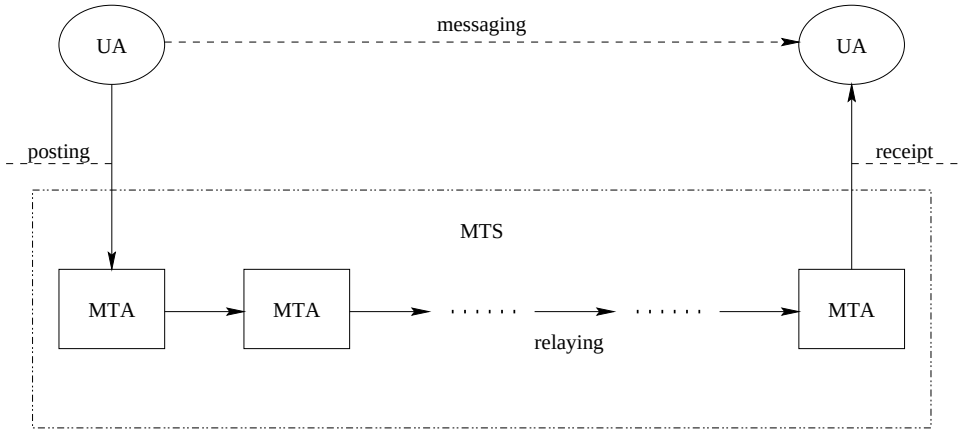

Figure 1.2: A model for e-mail [Rose, 1993]

storing his own copy.

Another way to explain the distribution of news from peer to peer, is to compare it to the message transfer system (MTS) of relaying mail transfer agents (MTA) in OSI's message handling system (MHS, from the X.400 recommendations [Rose, 1993]). This is close enough to Internet e-mail in how it works that the comparison makes sense; Internet e-mail only issues receipts upon failed delivery, and then to the sender of the message. Messages are stored and forwarded for each node on the path from the sending UA to the receiving UA in both models. The important difference is that for Usenet news, messages are not distributed directly to the end users; they have to request them from their local reading server. Figure 1.2 shows the e-mail model, and figure 1.3 on the next page shows the similar Usenet model.

As opposed to e-mail, is that news is not a reliable medium of transport for messages. News was not designed for reliability, and there are control mechanisms that allow people to remove their own articles after they were posted. It is possible for one reading server to offer articles within one same newsgroup that another news server does not, yet responses to these articles may show up on both servers. This will typically happen if an article is attempted sent from one of the servers to the other, and the other does not respond or accept it before a predefined timeout at the first server. Where e-mail via SMTP usually will generate a response to the message sender if the message could not be delivered, news offers no such service. This is good for the users, whose mailboxes would be overflowing with such responses if there should be one generate for each of the news servers that could not receive it, considering that there are tens of thousands of news servers the article may have been attempted distributed to.

While figure 1.3 on the following page shows the general idea behind how an article is distributed, it hides the fact that different peers have a vary-

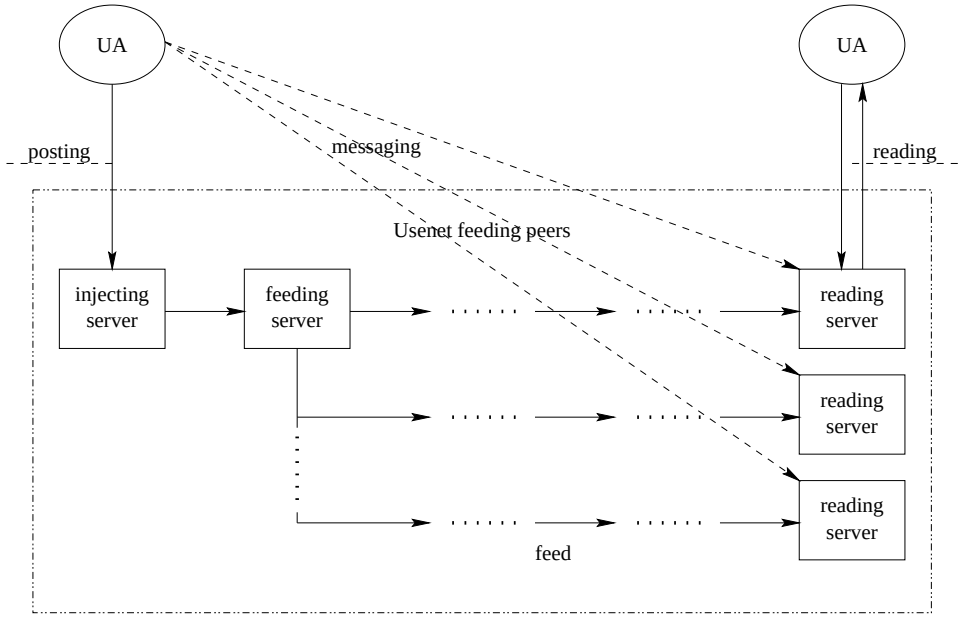

Figure 1.3: The Usenet distribution model simplified

ing number of newsfeeds. Even worse, it does not show that peers do not transport exactly the same amount of articles everywhere to everyone. What really happens is that each of the news administrators has made agreements with one or several news administrators about which newsgroups or hierarchies they will distribute between themselves. Some of these transport as much data as they can get — a full newsfeed — and can be considered part of a Usenet "backbone". Others transport other amounts of data. In addition to these differences in newsfeed size, these peers do not necessarily connect with the closest other peer. These issues are attempted visualized in the fairly complex figure 1.4 on the next page. The real Usenet distribution network is far more complex.

Information Structure

Articles are organized in *newsgroups* (discussion groups), similar to mailing lists in that they each have a name and a particular topic of discussion. In difference from mailing lists, newsgroups are organized in named hierarchies.

It is possible for an article to be posted to several newsgroups simultaneously; this is called *crossposting*.

The newsgroup names are on the form:

```
hierarchy_name[.subhierarchy_name]*.group_name
```

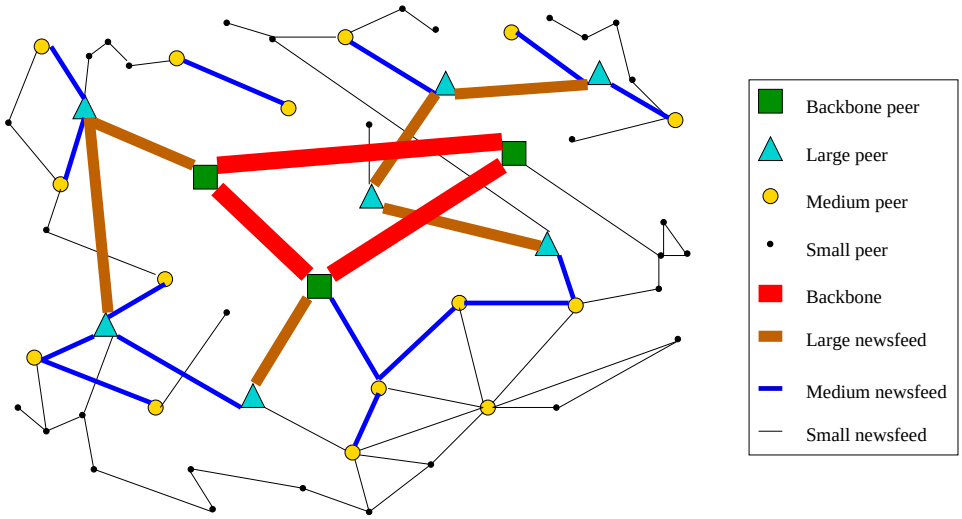

Figure 1.4: Distribution network among Usenet peers simplified

Hierarchy	**What**
comp	Computers
humanities	Arts and humanities
misc	Miscellaneous
news	Usenet
rec	Recreational
sci	Science
soc	Social/Sociology
talk	General discussions

Table 1.2: The Big 8 hierarchies

The names bear some significance to what the topic of discussion on that particular group is, both in that it has influence on what is to be discussed there, and in that it shows what actually is discussed. In addition to its name, most groups have a brief description stored at the reading server.

The core of Usenet's organization is eight top level topically organized hierarchies (see table 1.2) frequently referred to as the *Big 8*. The alternate hierarchy *alt*, which is more "free" in how groups are created and organized, is also considered part of the core by many users and news administrators.

There are also national and local hierarchies that not necessarily follow this organization scheme for choice of top level names, but use similar schemes for their own subhierarchies. See also table 1.3 on the next page for a list of example hierarchies. Below, I attempt to show the hierarchical structure through a couple of example newsgroups:

Hierarchy	What	Function	Organization
swnet	Swedish	Generic	Topical
no	Norwegian	Generic	Topical
norge	Norwegian	Geographic	Geographical
ntnu	University	Generic	Topical
uio	University	Generic, classes	Topical
3dfx	3dfx corp.	Products	Topical
microsoft	Microsoft corp.	Products, technologies	Topical
demon	ISP Demon, UK	Generic, support	Topical
online	ISP Nextra, Norway	Generic, support	Topical

Table 1.3: Example hierarchies

- **comp.lang.c**
  The top hierarchy *comp* is a short for "computer(s)". The subhierarchy *lang* is a short for "language" within the general topic of computers, so we understand it to mean computer language. The final part of the name tells us it deals with C, a programming language. This group has the description *"Discussion about C."*.

- **rec.arts.sf.movies**
  The top hierarchy *rec* is a short for "recreation(al)". The subhierarchy *arts* is self-explanatory, while the name of its subhierarchy *sf* contain the initials for "science fiction". This group has the description *"Discussing SF motion pictures."*.

The group description stored at the reading server helps people to select the correct group, in case the group name does not reflect the topic well enough. It is also common for popular groups to have *charters*, describing desired content, usage, and user behaviour, plus *FAQs*[8].

Within groups, articles (even the crossposted ones) belong to an inner structure, called a *thread*. If a new article is posted in response — a *followup* — to an older article, the newsreader is supposed to create a header — **References** — with the Message-ID of the older article. If the References header already exists, it should keep the old content of it and add the Message-ID of the older article to the end of that content.

A Note on the Subject Header If the user does not choose to change the subject of the article he is posting a followup to, the UA is supposed to append "`Re: `"[9] (Latin for "with regard to") at the beginning of the subject. Some UAs have the misconception that "Re" means "reply", and translates

[8] Frequently Asked/Answered Questions, usually in a list form

[9] The space following the colon is mandatory.

Command	Description
ARTICLE	Get the complete article with number artno or Message-ID mid
AUTHINFO	User authentication
BODY	Get the body of article with number artno or Message-ID mid
GROUP	Change group to the specified newsgroup, get available article numbers
HEAD	Get the headers of article with number artno or Message-ID mid
HELP	Lists summary of available commands
IHAVE	We have article with Message-ID mid, and are prepared to send it
LAST	Jump to the previous message in the current newsgroup if possible
LIST	List various data, default is available newsgroups and articles
MODE reader	Change connection mode from server to reader
NEWGROUPS	List new newsgroups since a date and time
NEWNEWS	List new articles in the specified newsgroup since a date and time
NEXT	Jump to the next message in the current newsgroup if possible
POST	Post an article in accordance with the news article format
QUIT	Exit and close connection
SLAVE	Tell the server that we are a slave server, not a user
STAT	Get the Message-ID of article number artno, mark it as the current article

Table 1.4: NNTP commands, RFC 977

"Re" into abbreviations of other languages' word for "reply". This causes problems when standard compliant UAs parse the same subject line, and append "`Re:  `" again, because the subject did not start with it already.

NNTP Commands

Table 1.4 shows NNTP commands as defined in RFC 977, and table 1.5 shows those additional commands supported by INN[10] 2.2.

[10]InterNetNews, a news transport system created in 1991 by Rich Saltz, [Salz, 1991]

Command	Description
LISTGROUP	List article numbers in a newsgroup
XGTITLE	Get the title/description for a newsgroup
XHDR	Get a specific header for a range of articles, current newsgroup
XOVER	Get NOV data for a range of articles, current newsgroup
XPAT	Get headers matching a pattern by range or Message-ID, current newsgroup
XPATH	Get the file name of a stored article by Message-ID

Table 1.5: Additional NNTP commands, from INN 2.2

Control Mechanisms

To control the creation, modification, and removal of newsgroups, and to give users a chance of withdrawing articles after they have been posted, there is a special kind of articles known as *control* messages.

Using a very basic authentication scheme through the **Control** header, one can send *newgroup*, *checkgroups*, *rmgroup*, *cancel* and *supersedes* messages, which the news servers of the world may choose to honor or ignore, depending on the administrator's policy.

Article Level Control The *cancel* and *supersedes* control message types are there to control actual articles. Sending a *cancel* message should — if it is honored[11] — result in the removal of the article you ask to cancel out. A *supersedes* message has its own header, **Supersedes**, but is effectually just a combined cancellation of an old article, with a new article attached; effectively, it replaces the old one. The header can also be used for repositioning the new article within the thread it belonged to.

Newsgroup Level Control The *newgroup*, *checkgroups* and *rmgroup* control message types are for creating newsgroups, updating a list of newsgroups (including descriptions) and removing newsgroups.

Modes of Participation When a newsgroup is created or modified, the group essentially has three different modes of participation by the users. These are:

1. Reading rights only
2. Reading and posting rights
3. Reading and limited posting rights (moderated newsgroup)

A *moderated newsgroup* is no different from an ordinary newsgroup, with a few exceptions. All articles posted to such a newsgroup are automatically forwarded via e-mail to a *moderator*. That moderator can either be a program or a person, who checks the articles according to some criteria defined for that particular newsgroup. Articles that fulfill those criteria are posted to the newsgroup through the use of the **Approved** header. Other articles are discarded, and sometimes returned to their originating author.

[11] Not all news servers honor cancel and supersedes messages, because it is very easy to forge these message types, making it easy to get rid of other people's articles.

```
From: moderator@dana.de
Subject: cmsg checkgroups
Newsgroups: de.admin.news.announce,de.admin.news.groups,
        de.alt.admin
Followup-To: de.admin.news.misc
Date: 06 Jul 2001 15:53:54 -0000
Sender: moderator@dana.de
Control: checkgroups
Approved: moderator@dana.de
Message-ID: <checkgroups_2001-07@dana.de>
X-PGP-Key: 0xD3033C99
X-PGP-Sig: 2.6.3ia From,Sender,Newsgroups,Followup-To,Subject,
        Control,Approved,Message-ID,Date
        iQCVAwUBO0XfEnU26rXTAzyZAQFq5gP9HK0//6qBiPjChcARIZoCZvmZh
        EHFGWpzkXLemOVzPD32MDUQ6oMgqUohhVxDIK24AHNxm74QtokTKuVv0u
        OnBaubYHCrm9CGOQMD6k7ztfib10+2iKSisB6hQaPgudH/fg3VlVrSIYr
        6p33IFGzD1xMKTnPdEHso3KIsnHMKM9U=
        =Q4mo
```

Figure 1.5: Example of headers for PGP-verification

Verification of Identity Currently, there is no built-in verification of identity in NNTP. News administrators choose whether to honor the different control message types, and to which degree they do so. A common solution is to require that creating, updating or removing newsgroups requires an additional set of headers for PGP[12] authentication, **X-PGP-Key** and **X-PGP-Sig**, which are used for verifying the named headers (see figure 1.5 for example PGP headers and headers meant for verification).

Storage Methods

Since articles are basic text files, the basic storage method is a simple Unix filesystem hierarchy, following the group hierarchies, and storing articles in numbered files as they were accepted by the news server. The storage area for news is called a *news spool*. A typical structure as seen from a user of a Unix system is shown in figure 1.6 on the next page. "`ls`" is a Unix command for listing files and directories, and the "`-F`" option displays the ending '/' for directories. "`/var/spool/news`" is the directory where news articles traditionally are stored under Unix.

Some of the headers are stored in *news overview* (*NOV*) databases, to make it easier on the reading server when user agents want to get an overview of certain header information for several articles. The NNTP command for

[12] Pretty Good Privacy, a commonly used software for authentication by a choice of several cryptography standards.

```
$ ls -F /var/spool/news/comp/arch/
7804 7805 7806 7807 7808 7809 7810 7811 7812 7813 7814 7815
7816 7817 7818 7819 7820 7821 7822 7823 7824 arithmetic/
bus/ embedded/ fpga/ hobbyist/ storage/
$ ls -F /var/spool/news/comp/arch/arithmetic
651 652 653 655 656 657 658 659
```

Figure 1.6: Directory structure for a news spool under Unix

Article number (or filename)
Subject header
From header
Date header
Message-ID header
References header
Byte count
Line count (Lines header)
Optional additional headers (must include header name)

Table 1.6: Information stored in NOV databases

doing that is XOVER, which works on a per group basis.

Table 1.6 shows the information that will be stored in a NOV database, in the order user agents will receive them ([Collyer, 1992], included in A.2 on page 98). User agents typically request overview data when they are about to display a summary of the available articles in a group. This means that the user is either subscribing to the group or that he is looking it over.

Headers stored in NOV are quick to fetch, because there is less data to manage for the news server. If other header data is needed, the news server must look in the article store for these headers. This is very resource demanding for the news server, and normally includes the same work as looking up an individual article; only the amount of data received by the user agent differs.

DEFINITION 1 (READ EVENT)
When a newsreader fetches data from the news server that does not exist in the news overview data, we call it a read event*. The NNTP commands that always cause a read event are ARTICLE and BODY. With INN 2.2, the commands HEAD, XHDR and XPAT also do so.*

A read event is not necessarily the result of a direct request for an article by a user. Advanced user agents offer filtering and scoring of articles that may cause many more read requests than there are articles being read.

Other methods have been used, including concatenation of several articles into single files, *cyclic news spools* (ring buffers), or more traditional databases.

Common for all these methods is that articles usually are stored for a limited time, determined by those administering the news server. Cyclic news spools are self-limiting on space; when the spool is full, the news server writes new articles over old articles. Traditional spools have *expire* times set on a group or hierarchy level, so that articles older than the specified time (e.g. 2, 7 or 31 days) are removed from the news spool. In both cases, expired news articles usually remain unavailable to the user from that reading server. They can remain available through proxying methods, such as one reading server acting as a *slave* towards a *master* with longer news *retention* times. If the slave does not have a particular article, it forwards the request to its master.

1.3 The History and Development of Usenet

In 1999, Usenet News turned 20 years. In those 20 years, many things have changed, but some underlying principles have remained.

When BBSes (Bulletin Board Systems) were very popular, many people expressed that Usenet was just another BBS. Where BBSes (with few exceptions) were limited to single computers and people connected with their modems (or whatever means they had) to post their messages and discuss with others of like or different mind, Usenet was from the beginning a distributed system, where messages were transmitted between different computers to be available from more servers. Usenet was probably best compared with a network of BBSes, each carrying the same discussions.

1.3.1 The Beginning of Usenet

The birth of Usenet is linked to a single event: An operating system upgrade rendered existing bulletin board software non-functional, which caused two graduate students at Duke University in North Carolina, Tom Truscott and Jim Ellis, to develop the idea of a distributed news system. This was in the fall of 1979 [Hauben and Hauben, 1995].

At first, Usenet was a substitute for a broken bulletin board system, an experiment with UUCP, based on a 3-page Unix shell script. The script allowed people to subscribe to different groups, post and read notes in sequence, and also post to different groups at the same time (crossposting) [Hauben and Hauben, 1995].

Steve Bellovin, one of the people who Truscott and Ellis presented their design to, wrote the shell script using Unix V7 to test the design concept. The first Usenet was a two-server setup, but it evolved quickly. Bellovin notes:

> "We estimated a maximum size of 100 sites, and 1-2 articles a day,

net-wide...you couldn't read things out of order. The goal there (and in many other spots) was to have software free of databases. Instead, we chose to let the file system do the work."

[Hauben and Hauben, 1995]

1.3.2 Big Changes

Usenet became a hierarchical structure of discussion groups, each group a community, with its own expectations of conduct, the so-called netiquette. Using UUCP for distribution and the old structure worked well for a few years, but it became obvious to the news administrators that the distribution model would not hold, and that the hierarchical structure needed a revolution. The result was a new communication protocol, NNTP, a new article format (RFC 1036) a process of renaming hierarchies ("The Great Renaming" [Bumgarner, 1995]) during 1987 and 1988.

Both the distribution protocol and the article format were changed. In February 1986, the current protocol — NNTP — was described by Brian Kantor (U.C. San Diego) and Phil Lapsley (U.C. Berkeley) in RFC 977
[Kantor and Lapsley, 1986].

The current article format was described by M. Horton (AT&T Bell Labs) and R. Adams (Center for Seismic Studies) in RFC 1036 in December 1987. Since then, the de facto article format has changed somewhat ("Son-of-RFC 1036", [Spencer, 1994]), and modifications have been made to the distribution protocol (RFC 2980, [Barber, 2000]), through efforts from the INN project and other news software programmers. The IETF is working on a new set of Usenet standards to reflect current and future practice and needs (Article format: [Lindsey, 2001], NNTP: [Barber, 2001]).

After "The Great Renaming", Usenet consisted of the top level *comp, misc, news, rec, sci, soc* and *talk* hierarchies. The *humanities* hierarchy followed later. These are most commonly referred to as the "Big 8". Today, "Usenet", "News" or both refer to a seemingly arbitrary selection of the many available hierarchies in the world, not just the "Big 8".

1.3.3 Traffic Growth in a Historical Perspective

Usenet traffic — meaning the number and size of the daily accepted flow of articles for a site that attempts to get "all" that is posted to Usenet — has never increased slowly, at least not volume measured in bytes. According to Hardy ([Hardy, 1993], citing Gene Spafford), the rate of growth was fairly constant from 1979 to 1988 (see table 1.7 on the facing page, based on a similar table in [Spafford, 1990]).

About 1993, the use of services available via the Internet (such as Usenet) began increasing dramatically with the introduction of the World Wide Web (WWW), and the following success stories of the commercial ISPs.

Year	Sites (Growth)		Art/Day (Growth)		Size/Day (Growth)	
1979	3	(-)	2	(-)	?	(-)
1980	15	(400%)	10	(400%)	?	(?)
1981	150	(900%)	20	(100%)	?	(?)
1982	400	(167%)	50	(150%)	?	(?)
1983	600	(50%)	120	(140%)	?	(?)
1984	900	(50%)	225	(88%)	?	(?)
1985	1 300	(44%)	375	(67%)	1 MB	(?)
1986	2 500	(92%)	500	(33%)	2 MB	(100%)
1987	5 000	(100%)	1 000	(100%)	2.5 MB	(25%)
1988	11 000	(120%)	1 800	(80%)	4 MB	(60%)

Table 1.7: Usenet growth 1979 to 1988

From then to now, we have seen a change from a mostly academic environment to a common public service, available to millions of users across the world. Usage has also changed from being mostly text with a few pictures and sound clips, to many more pictures and even movies. This non-text content is also known as *binaries*.

Even full-blown CD ISO images and DVD ISO images have been posted to newsgroups meant for binaries, while home bandwidth has increased from 300 baud modems to "broad-band" technologies like DSL[13] and cable modem. This was foreseen in [Hardy, 1993]:

> The increasing use of Usenet to send binary data such as pictures and sounds, and in the future, animations and video, will push these requirements significantly higher in the near future.

As a result of these changes and the increase in users, a full newsfeed can contain around 100 000 newsgroups, delivered at a rate of more than 1.5 million articles for a total of more than 300 gigabytes a day, between tens of thousands of servers[14]. The next section will explain these numbers further.

1.4 Challenges and Problems With Today's Distribution Model

With a volume of around 300 gigabytes in 1.5 million articles each day for one huge server [Kondou, 2001], there are obviously problems with storage space and bandwidth, if you wish to store everything you can receive of articles for more than a day.

[13] Digital Subscriber Line, subtypes SDSL (symmetric), ADSL (asymmetric), VDSL (very high speed DSL)

[14] Freenix' "Top 1000 Usenet sites" [Freenix, 2001] lists the most-distributing servers

Many people think that articles should be stored for at least a month, and say that "disk space is cheap". Perhaps that is true if you count in EIDE based hard disks, which cost from around 2 USD/GB. [15]

If you want to run a news server with a newsfeed of the size mentioned earlier, and you want to store all articles for a month, you will need only 9000 GB of that "cheap" storage space. That is around 18 000 dollars, if we assume that the above cost increases linearly with storage space and ignore the problem of having hardware, space and cooling to support that many (more than 100) drives. But it is not really that simple. News administrators tell me that they do not want to rely on cheap disks, because they need a high degree of reliability, and also a high degree of performance, so they depend on SCSI disks, which are far more expensive, at least three times more so than EIDE. [16]

The large feed news server must each second be able to receive and write more than 17 articles sized 210 kilobytes, **if** the 1.5 million articles (300 GB) are spread evenly over the day. That may be easy enough for the news server and the local network, but this is also traffic intended for the Internet. This is approximately 30 Mbps before any IP packet overhead, which means a minimum of a dedicated T3 line [17] would be needed.

It should be evident that having a news server with a large feed is not a small investment, and that saving some of the storage space and bandwidth by not taking a large feed will save money.

In the following section, we will have a closer look at the continued growth of Usenet.

1.4.1 Continued Traffic Growth

Hardy writes in The Usenet System [Hardy, 1993] that

> Usenet has continued to grow exponentially since its creation in 1979 by two graduate students at Duke University, with traffic volume increasing recently by as much as 10-15% per month. A full newsfeed at a typical site might average more than 30 megabytes per day (about 10 times the size of the King James Bible, orf 100 paperback novels, or [10 200 A4 pages].).

Let us examine how the growth of Usenet has continued, from 1988 to 2001.

At the time I started working with this thesis, I only had the graphs from [Nixon, 2000] (figures 1.7 on the next page and 1.8 on the facing page) to

[15] Typical EIDE disk, e.g. an 80 GB Maxtor, according to www.pricewatch.com, 2001-07-16

[16] A typical SCSI disk, e.g. a 73 GB Hitachi, costs about USD 650, according to www.pricewatch.com, 2001-07-16.

[17] T1 = 1.5 Mbps, T2 = 4.5 Mbps, T3 = 45 Mbps

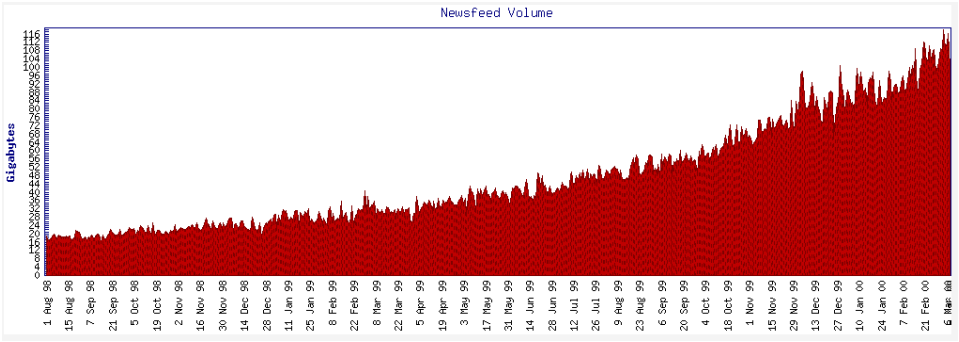

Figure 1.7: Size of a full Usenet feed August 1998 – February 2000 [Nixon, 2000]

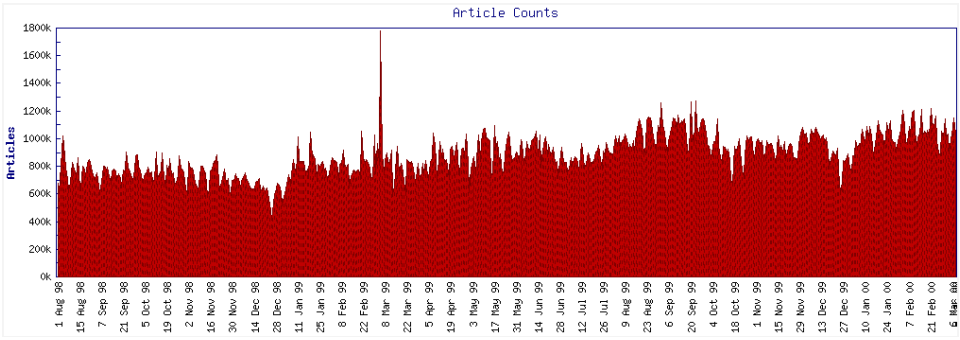

Figure 1.8: Count of articles in a full Usenet feed August 1998 – February 2000 [Nixon, 2000]

Source	Date	Size/Day	Growth	
			Yearly	Monthly
[Spafford, 1990]	1988 Oct	4 MB	60%	4%
[Van Hees, 1993]	1993 Oct	56.8 MB	70%	4.5%
[Nixon, 2000]	1998 Aug	18 GB	231%	10.5%
[Nixon, 2000]	2000 Feb	120 GB	254%	11%

Accuracy is 1% yearly, 0.5% monthly

Table 1.8: Usenet feed size 1988 to 2000 (based on [Nixon, 2000])

Source	Date	Arts/Day	Growth	
			Yearly	Monthly
[Spafford, 1990]	1988 Oct	1 800	80%	5%
[Van Hees, 1993]	1993 Oct	16 800	56%	4%
[Nixon, 2000]	1998 Aug	700 000	116%	6.5%
[Nixon, 2000]	2000 Feb	1 100 000	35%	2.5%

Accuracy is 1% yearly, 0.5% monthly

Table 1.9: Usenet articles 1988 to 2000 (based on [Nixon, 2000])

work with. It seemed like Usenet had a stable near exponential growth in feed size, and that this trend would continue indefinitely. Another apparent trend was that feed size would continue growing stronger than the number of articles, see tables 1.8 and 1.9.

My impression then was that Usenet feeds would continue to grow until technical limitations made it extremely difficult or impossible to sustain the enormous data flow, especially considering the enormous growth rate between 1993 and 1998 — the greatest growth in Usenet's history so far.

However, in March 2000, [Nixon, 2000] went off the web, and updates on the feed sizes were no longer available. I was forced to look for another source of information, and got a pointer to [Kondou, 2001], which has a different starting point in both feed size and number of articles registered, and reveals that there is no different trend from 1998 to 2001. I have asked for and looked at other sources for data, but the general agreement among experienced news administrators has been that [Kondou, 2001] is the most reprecentative for a full newsfeed's size. The other sources of data seemed in general to be less stable in terms of feed size and number of articles per day, and they also showed lower numbers for both volume and number of unique articles. In other words, [Kondou, 2001] appeared more complete.

In a news article in **news.software.nntp** in June 2001 (copy in B.3 on page 117), I was made aware that Katsuhiro Kondou was about to leave his job administering newsfeed.mesh.ad.jp, the host the graphs in [Kondou, 2001]

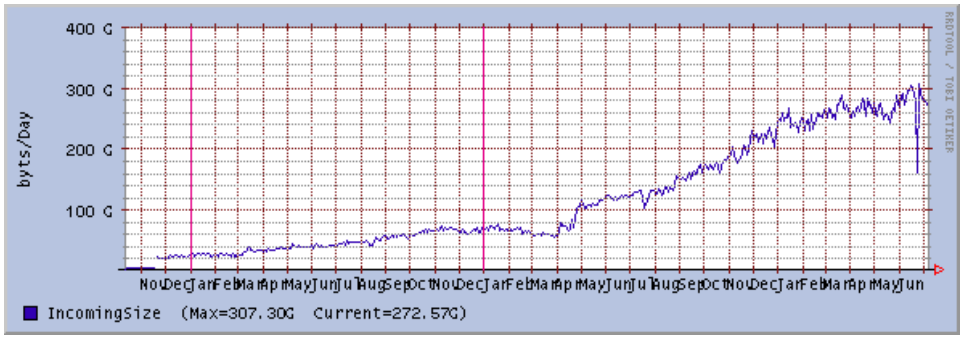

Figure 1.9: Size of a full Usenet feed November 1998 – June 2001 [Kondou, 2001]

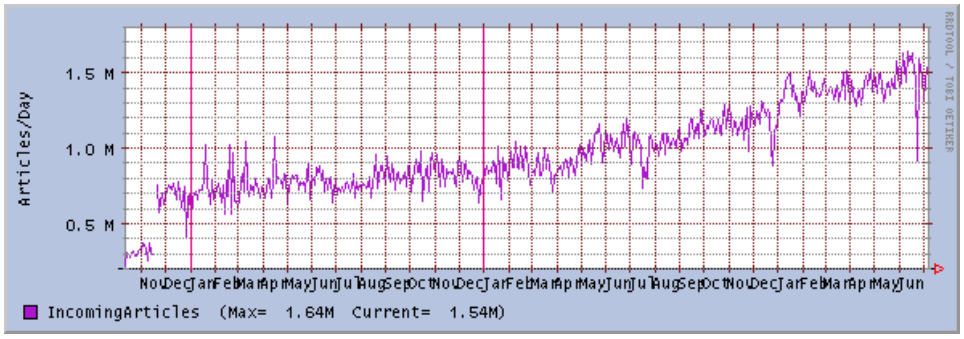

Figure 1.10: Count of articles in a full Usenet feed November 1998 – June 2001 [Kondou, 2001]

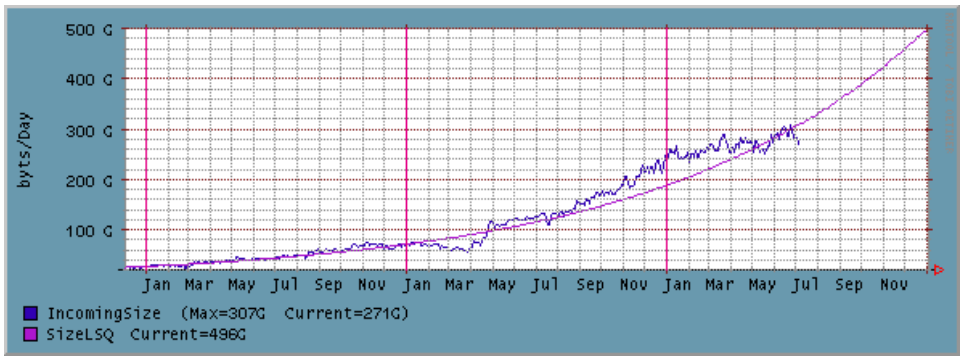

Figure 1.11: Projected size of a full Usenet feed to December 2001 [Kondou, 2001]

Source	Date	Size/Day	Growth	
			Yearly	Monthly
[Spafford, 1990]	1988 Oct	4 MB	60%	4%
[Van Hees, 1993]	1993 Oct	56.8 MB	1 70%	4.5%
[Kondou, 2001]	1998 Dec	22 GB	218%	10%
[Kondou, 2001]	2000 Feb	64 GB	150%	8%
[Kondou, 2001]	2001 Feb	240 GB	275%	11.5%
[Kondou, 2001]	2001 Jun	280 GB	76%	4%

Accuracy is 1% yearly, 0.5% monthly

Table 1.10: Usenet feed size 1988 to 2001 (based on [Kondou, 2001])

are taken from. Steady newsfeeds are too dependent on active news administrators, and a decline in incoming feeds followed. Therefore, the numbers for July cannot be trusted, and I have chosen to focus on the days around June 1 instead. A look at figures 1.9 on the preceding page and 1.10 on the page before seems to confirm that situation; there is a noticeable gap in the number of articles and the volume of news shortly after the middle of June.

It is interesting to see that the growth rate for the feed size seems to have diminished between February and June of 2001. Figure 1.9 on the preceding page and 1.10 show that there has been a much lower growth rate for those four months than earlier. Figure 1.11 shows the projected feed size according to [Kondou, 2001], where there also was a similar decline in growth rate between November 1999 and April 2000. The growth in number of articles is a bit more steady, although it saw a decline from 1998 and onwards, from nearly 60% a year to between 30% and 50% a year.

Table 1.10 shows the daily feed size for several periods of time since 1988 until now. The number of 30 MB/day from [Hardy, 1993] is an estimate for a “typical site”, while he later estimates between 25 and 50 MB over 13 000

Source	Date	Arts/Day	Growth	
			Yearly	Monthly
[Spafford, 1990]	1988 Oct	1 800	80%	5%
[Van Hees, 1993]	1993 Oct	16 800	56%	4%
[Kondou, 2001]	1998 Dec	700 000	58%	6%
[Kondou, 2001]	2000 Feb	950 000	30%	2%
[Kondou, 2001]	2001 Feb	1 350 000	47%	3%
[Kondou, 2001]	2001 Jun	1 500 000	37%	2.5%

Accuracy is 1% yearly, 0.5% monthly

Table 1.11: Usenet articles 1988 to 2001 (based on [Kondou, 2001])

articles per day across 62 000 sites. These numbers are inaccurate at best, while the numbers from [Van Hees, 1993] are exact measurements over several news servers in Europe from September to November 1993. I used the middle of the period (33 days), because only a partial feed was available to Van Hees in the first and last periods.

Vacation times may disturb the most recent and detailed results (from [Kondou, 2001]) slighty; historically, there have been dips in the growth during holidays and vacations.

As mentioned earlier, we see a downshift in the growth rate for feed size around February 2001. Informal comments made by participants in the newsgroups **news.software.nntp** and **alt.binaries.news-server-comparison** indicate that one possible reason may be that the Satellite based NSP Cidera has lowered the maximum article size they allow through their feeding servers. Cidera has not responded to questions per e-mail on this subject.

It seems safe to assume that the trend of near exponential growth will continue, although possibly with the currently slightly diminished growth rate (5-6% monthly increase in size, 2-3% in number of articles).

However, if the rate of growth picks up again to more than a doubling every year, storing full Usenet feeds for even shorter periods of time may become increasingly expensive, because the traditional doubling in size of harddisks every thirteen months (90% yearly increase) seems to stop around 2005, and replacement technology is not expected to be ready until two or three years later [Toigo, 2000]. Bandwith is perhaps the limiting factor here, since bandwidth growth traditionally has been lower than the growth in disk size. In 1969, the ARPAnet had a backbone of 56 kbps, and in 1999, the US backbone is upgraded to 2.5 Gbps [Zakon, 2001]. This equals a yearly increase in bandwidth of 43%, too little to sustain continued growth in the full feed size.

1.4.2 Spam and Filtering

A large ISP and NSP — for instance Nextra[18] in Norway — can have several hundred thousands of customers. Still, the customers do not read all the articles. According to interviews with Nextra's news administrator, Stig S. Mathisen, Nextra's customers read 450 000 articles a day within the *no* hierarchy in October 1998. Only 150 000 unique articles were available in that hierarchy at that time, but not all of those 150 000 articles were read, and a great deal of the articles were read several times. Section 5.1 on page 59 shows and discusses some statistical data from Nextra.

In total, the customers read less articles than what is received every day, which (combined with the numbers for the *no* hierarchy) means that a lot of articles are left unread.

News administrators I have been in contact with believe that a lot of storage space and bandwidth is wasted on unread articles. They try to reduce disk usage by avoiding groups and hierarchies with contents that may be illegal, filtering out so-called "spam" (massive posting of irrelevant articles to many newsgroups) and binaries (articles containing pictures, movies, computer programs etc., i.e. anything that is not plain text). According to these news administrators, about 95-98% of the volume of a full newsfeed comes from binaries, but these are only a small part of the number of articles.

However, the filtering techniques the news administrators use are not a part of the news system itself. They are a collection of arbitrary utilities they find handy, combined with "blacklists" that are sometimes distributed through other channels, and other times generated manually by the news administrators in question. One such blacklisting system is the *NoCem* advisories posted to the newsgroups **news.lists.filters** and **alt.nocem.misc**.

1.5 Handling the Challenges

Compared to Usenet, documents on the web live a long time. The web site Deja[19], later bought by Google[20] and renamed Google Groups[21], have attempted to store all news articles, with the exception of most binaries, for eternity. They have failed in that they do not have all news articles for the time period they are covering. Some of these are missing because authors have reserved themselves against being stored by use of the optional **X-No-Archive** header, which Google honors by not storing these articles. It is not uncommon for regular news servers to not get all articles that are posted to Usenet, but it is regrettable that those who set out to store and provide "everything" are unable to do so. Note that Google Groups does not try to store binary

[18] Telenor Nextel until a name change medio 2000
[19] www.deja.com
[20] www.google.com
[21] groups.google.com

articles, which makes their task more manageable. The National Library of Norway preserves articles posted to the *no* hierarchy as a part of Norway's cultural heritage.

A news article cannot be changed once it is posted, but it can be cancelled and replaced by other articles, or simply be expired (deleted) because the news server attempts to conserve storage space. Such removal of articles happen all the time, since news administrators want to limit the use of storage space, and partially because there are automated utility news programs which cancel spam (see section 1.4.2 on the preceding page).

To handle the ever-increasing traffic on Usenet, I propose the introduction of advanced caching proxies. While caching proxies have been common for the WWW for a few years, this has not been the case for Usenet .

DEFINITION 2 (PROXY)
In the context of Usenet, a proxy is an intermediate server that transparently to user agents or downstream peers provides articles that it itself does not have, but are available from one of the proxy's upstream peers.

DEFINITION 3 (CACHING)
For Usenet, caching means copying and storing incoming data, and keeping that data for a period of time.

Introducing such caching proxies may produce a challenge in itself, since Usenet works well — in spite of the enormous amounts of data transmitted every day — because of the flooding algorithm. Changing the algorithm may have unforeseen and undesired consequences, in terms of usability, flow and group control.

DEFINITION 4 (PREFETCHING)
Fetching data from an upstream peer before it is requested by user agents or downstream peers.

It is useful to note that Usenet's flooding algorithm can be viewed as a time based prefetching caching mechanism, in that everybody gets a recently posted article as soon as possible after it is posted, and that it is then only up to the leaf nodes — the reading servers — to decide how many of these are available to their users.

The following subsections are based on ideas of the author, as well as some input from several experienced news administrators.

1.5.1 Requirements for Caching Strategies

If a caching proxy would be used, it is important for the users that they do not notice a negative difference between it and an ordinary reading server. If they do, they will want to use the ordinary servers instead.

It is difficult to say whether the users will even notice the presence of a caching proxy in a negative way; we do not know what the average user expects when reading Usenet. With traditional Usenet reading servers, articles seem to disappear after arbitrary periods of time, decided upon by the news administrators. In addition comes the unpredictable "holes" in newsfeeds, where articles do not show up when the users expect them to.

We also must not forget the other concerns, which also influence the users' view of their Usenet service. If the NSP cannot afford to sustain the incoming number of articles the users want, the users are more likely to see such "holes" among the available articles. Bandwidth suffers when many users try to get large amounts of data from the Internet, and the pushed newsfeeds ensure that at least some of the things people want are available much closer to the base of their network connection, possibly saving costly long distance bandwidth. But the availability of that data depends on storage, and as mentioned in section 1.4 on page 17, this has been an increasing problem in the later years. Although it seems more bandwidth friendly than storage friendly, that may also not be true. Since bandwidth capacity is growing slower than storage capacity (see the end of section 1.4.1) and there are many unread articles among those the NSP receive. There are probably important gains to be made here as well.

Caching proxies for news is a compromise that at the same time tries to cater for bandwidth, storage, and the users, by adapting the availability of hierarchies, groups, and articles according to what people actually read. Because of this compromise, caching will not help the backbone servers.

The general idea is to have a caching proxy which initially does not carry any articles, except in a few hierarchies or groups selected by the news administrators. Several of the approaches mentioned below depend on a header-only feed (dicussed in section 3.3.4 on page 45), so that the proxy has a chance of keeping track of the information it bases its caching algorithm on.

A caching proxy for Usenet would store and keep individual articles, newsgroups and/or hierarchies until a certain lower limit of available disk space in terms of blocks, files or bytes is reached (e.g. in a ring buffer), or it would expire articles, groups and/or hierarchies after a certain time has passed, similarly to how news expiry works with a regular, pushed newsfeed today.

Whether it is a distributed cache that uses partitioning between different caching servers ([Danzig, 1998], see also section 3.1.1 on page 35) or a cache cluster consisting of frontend and backend servers to balance the load ([Kurcewicz et al., 1998], see also section 3.1.2 on page 36) is not something I will attempt to evaluate. It may be of interest that the metadata in the case of Usenet is either NOV data or complete article headers.

DEFINITION 5 (METADATA)
Metadata is data about data. In the case of news, this is NOV data or complete article headers, which allows news servers to utilize advanced

caching techniques. It can also be other data, such as blacklisting filters.

1.5.2 Different Caching Strategies

Based on my own experience with and observations of news, as well as feedback from other experienced news users and administrators, I have identified a few different approaches to caching for Usenet. As far as I know, these are not described in literature. These approaches are briefly presented in the rest of this subsection, and more thoroughly discussed in chapter 4.

Simple Caching Strategies

1. **Single article caching**
   The proxy only fetches the article requested, and keeps it in the cache.

2. **Protocol command caching**
   The proxy caches specific protocol commands, and the results of these commands, e.g. fetching of newsgroup overview data, article headers or article bodies.

3. **Time based caching**
   The proxy fetches the article requested, as well as prefetching other articles within a given timeframe, before or after the requested article.

Metadata Dependent Caching Strategies

1. **Author caching**
   Articles with the same author as the article just read are prefetched and cached.

2. **Thread caching**
   The proxy keeps track of all articles in a thread in a separate datastructure, in order to know which articles have been posted as followups to any article. When an article is requested, the proxy fetches that article, and any other articles belonging to the same thread. If there are no other articles, the single article that was requested is still cached, as with single article caching.

3. **Subject caching**
   Articles with the same or similar subject as the article just read are prefetched and cached.

4. **Group caching**
   When an article is requested, the proxy prefetches all the other articles in the same group(s).

5. **(Sub)hierarchy caching**
   The same as group caching, except the proxy prefetches the other groups in the same (sub)hierarchy.

6. **Prefetching groups/hierarchies**
   Without any action on the user's behalf, the proxy prefetches (or is fed, like a normal news server) some selected groups and/or hierarchies, which will be available to the user as if it were a normal reading server.

Complex Strategies

1. **General header caching**
   The proxy prefetches (or is fed) headers for a selection of groups and/or hierarchies. One way to do this is to get all the headers of all the groups or hierarchies which are not among some selected unwanted groups or hierarchies, another is to get the headers of all explicitly wanted groups/hierarchies. If it is fed, this is a header-only feed.

2. **Statistical caching/prefetching**
   The proxy analyzes usage patterns and adapts a combination of other caching strategies based on this.

3. **News reader controlled caching/prefetching**
   The user agent specifies general criteria for which articles the user may be interested in. As an example, the user may know whether he could be interested in reading other articles posted at around the same time as any article he is trying to read, or if he is interested in the same thread, etc. It can also include statistical analysis of the user's usage pattern.

Any of these may or may not produce a decent caching result for the users, and most can be combined.

Article Retrieval and Storage Strategies

The proxy can fetch/be fed articles, groups and/or hierarchies from different types of news servers. In the case of a cache miss, the proxy must fetch articles itself from a peer, usually from its upstream peers. Mentioned below are a few different ways of reaching the articles the proxy does not have in its cache.

- **Group/hierarchy repositories**
  The proxy is aware of one or several group and/or hierarchy repositories, where all articles ever posted to that group/hierarchy should be found. These do not exist in an organized way today.

- **Injecting server repository**
  The injecting server (the server where the article was posted to first) keeps all articles posted on that server forever. These do not exist in an organized way today.

- **Reverse path lookup**
  The proxy looks up news servers along its feed path. This requires that it knows which servers are feeding it news (or allow it to fetch news), which is common for news servers today.

1.6 Research Questions

While it would be interesting to find ways to limit the size of a full feed, this is not a primary goal for this thesis. By limiting the amount of binaries one's news server accepts, there are big gains in terms of bandwidth, without having to resort to any kind of advanced caching. Efforts with these and other improvement suggestions are discussed in chapter 3.

It is a matter of policy which newsgroups and hierarchies the news administrator finds acceptable, and which the administrator does not. If, however, the reason for unacceptability is that the groups in question are not read, caching may provide a more flexible solution in the long term.

From the viewpoint of a news administrator, there are three different kinds of newsgroups:

Required Newsgroups that the reading server **must** offer to users, no matter what.

Unwanted Newsgroups that the reading server **will not** offer to users, no matter what.

Other Newsgroups it does not matter for anyone but the users whether are available or not, or that are unwanted because they are not being read.

It is this last category, the other newsgroups, that we can hope to accomplish something with. My general idea is that if the availability of articles, newsgroups and hierarchies in this category is dynamically adjusted by a caching proxy, then it is possible to save bandwidth and storage without any loss to the users.

I aim to establish the following:

- **Which strategy or strategies are better for bandwidth?**

- **Which strategy or strategies are better for disk usage?**

- **Which strategy or strategies are better for user perceived performance?**
- **Will caching proxies (as suggested) be a general improvement to Usenet ?**

Bandwidth usage would benefit most if there are no cache misses, and that as few and as small as possible articles are fetched by or fed to the proxy as seldom as possible.

Disk usage has most to gain if no articles are ever stored in the proxy except in main memory, so that nothing is stored on disk.

The users would gain the most in perceived performance if all articles they could possibly want are prefeteched and stored locally, and if the strategy in question does not use too many system resources in providing articles to the users.

Caching proxies as they are suggested in this thesis would be a general improvement to Usenet if the sum of bandwidth usage, disk usage and user perceived performance is better than that of traditional news servers.

In chapter 4, I discuss the caching strategies presented in the previous section in light of the above questions. I also suggest some methods for testing. I present, evaluate and discuss my findings from data collection in chapter 5.

Chapter 2

Methods Used

The thesis is based on ideas I developed through several years' use of Usenet, a year of administering a news server, and discussions with various news administrators about the workings and challenges of news distribution.

During work with the thesis, I faced several problems. In this chapter, I first present how I searched for prior art, followed by a section dealing with conscious assumptions I consider significant. The next section is about data collection and interviews, and the last about data analysis.

2.1 Search for Prior Art

Searching for prior art was done in several ways:

- Discussions with news administrators, my advisor, and associate professors at the Department of Informatics at the University of Oslo about possible prior art
- Discussions and searches on Usenet
- Internet searches
- Library searches

My discussions with news administrators and associate professors revealed no scientific articles about Usenet caching and Usenet proxies, nor did the discussions on Usenet. My advisor gave me the GroupLens reference in association with an earlier essay on Usenet.

The Usenet searches were done with the news search engine Deja/Google Groups. These searches revealed only technical publications in the form of Internet standards that I already was aware of, Hardy's document [Hardy, 1993] and the Inktomi solution [Inktomi Corporation, 2000].

For Internet searches, I used the search engines Altavista[1] and Google. Library searches were done via BIBSYS, Inspec and ISI. The Internet searches and library searches revealed very few relevant article abstracts, and of the few that did, several proved to be irrelevant upon closer examination.

From what I have found, there are no peer reviewed sources dealing with caching of news in particular, nor with the growth of Usenet in a historical perspective. One article dealt with collaborative filtering of Usenet.

For this reason, I have looked at and commented some articles about caching web proxies for the purpose of finding usable similarities with what I am trying accomplish with caching of news.

I have also used a variety of other sources, including documentation for software, descriptive documents and statistics found on the web, technical documentation, Internet standards in the form of RFCs, articles found on Usenet as well as an essay I wrote in early 1998.

2.2 Significant Assumptions

The most significant assumption is that from statistical data from traditional news servers, it should be possible to test how suggested caching strategies compare to traditional news servers, by examining varying granularities of data and calculating how the number of articles and total size of articles changes with the different strategies.

I have also assumed that as long as it is technically possible for users to generate and send more and bigger news articles, they will. This assumption is based on observations on Usenet's growth in section 1.3.3 on page 16.

2.3 Data Collection and Interviews

To my knowledge, there were no existing surveys on Usenet reading habits when I started work on this thesis. In consultation with my advisor, I made an early decision not to perform any quantitative surveys to establish Usenet reading habits, in order to limit the scope of the thesis. In retrospect, this has limited the usefulness of my data collection and analysis.

Some news statistics were downloaded from the web sites of major news distributors (Supernews [Nixon, 2000] in the US and Mesh [Kondou, 2001] in Japan), other through the cooperation of anonymous news administrators.

I conducted several informal interviews via e-mail and personal conversations with news administrators that revealed what they perceived as key challenges, how they dealt with the Usenet flow before, how they deal with it today, and how they plan to deal with it in the future.

[1] www.altavista.com

These interviews also helped develop the possible and suggested strategies, which were decided upon before proceeding with gathering statistical data and testing.

In addition to these interviews, I have posted numerous questions to the discussion group **news.software.nntp**, to collect data on how news administrators perceived the issues, and how they dealt with them. Only three responses were received.

I contacted the biggest Norwegian ISP, Nextra, for usage statistics on their reading server. They helped providing a general overview of article article requests and XOVER commands, through a simple group-by-group statistic on reading usage. Time constraints and limited resources limited the detail of the data I could get from their busy servers.

I also wrote a set of patches[2] for INN. The patches cause INN to write detailed information on usage to log files. The intention was to have these patches installed with an upgrade of Ifi's reading server for two to three weeks before the Easter of 2001. This upgrade did not happen. After long delays, in July Ifi allowed me to apply a patch to their running reading server software, INN version 2.2. This has caused a problem with evaluation of the data, since I then had to collect data during summer vacation. I collected information on what articles were read, at what time, what the content of these articles' headers were, plus general NNTP commands sent to the server.

2.4 Data Analysis

Testing was performed by analysis and comparison of the statistical data points collected.

Nextra's data was analyzed for reading patterns on a group level, to see whether many groups are unread over time or not.

For this examination, Nextra's news administrator had extracted summary data. Further examination was done with Perl[3] scripts I wrote.

At Ifi, I wrote several Perl scripts for extracting data from logs. I also wrote other scripts for examining the extracted data. Errors in the log format combined with data corruption caused variances in the basis for evaluation.

Ifi's data was also analyzed for reading patterns. The more detailed information on reading made it possible to split most of the read events into two time periods: one for articles written before logging of commands started, and one for articles written while commands were logged, allowing shallow statistical examination of data. Time constraints did not permit more thorough analysis or evaluation.

[2] Patches are modifications to software source code

[3] Perl is a scripting programming language.

Chapter 3

Prior Art and Other Work

In my search for prior art on caching of news, I mentioned in chapter 2 that there were few relevant sources of information to be found. In this chapter, I first present two web proxies to show some general features of caching. Later, I present the few attempts at Usenet performance improvements, followed by other work for dealing with large Usenet feeds.

3.1 Caching Web Proxies

A web proxy is a server that acts as an intermediate server between one or several users agents on one side and one or several web servers on the other. The proxy server forwards URL requests to these web servers, and the result of these requests back to the users, unless there are filtering rules preventing that in the proxy.

A caching web proxy is a proxy that in addition to forwarding URL requests also stores a copy of — caches — each unique request and the result of that request. The copy is kept in the cache for as long as the configuration of the proxy specifies. When another user agent connects and requests the same URL, the proxy gives the user agent the cached copy of the result instead of forwarding the connection to the web server at the other end.

3.1.1 Network Appliance's NetCache

Network Appliance's NetCache is an architecture developed to be a "scalable, commercially supported, highly available Web cache" [Danzig, 1998]. It uses separate state machines for HTTP, FTP, Gopher, and supports persistent connections. The main focus is on WWW caching. A NetCache implementation holds "fetch modules" for the varius TCP based protocols it supports, a storage manager (to optimize I/O[1] for as few disk accesses as possible), a TCP/IP[2]

[1] Input and output
[2] Internet Protocol

transparency stack, web page filters and client access controls, and client side protocol processing for diverse protocols (HTTP, SSL[3] tunnels, DNS lookup, and extensible protocols).

For WWW, NetCache uses the GET if-modified-since operator to guarantee that it returns up-to-date URLs[4], but does not support HTTP 1.1's opaque cache validators yet. URLs can be verified on every reference or at configurable time intervals, but will not be cached if the MIME[5] headers specify negative expire times, are carrying cookies, are password protected pages or is a result of queries or CGI[6] like programs.

The cache can be scaled upwards by partitioning (division of the work load between several proxies), partially because the most popular web browsers support automatic configuration of proxies. Another architecture feature is hierarchically partitioned cache-to-cache workload, based on a hashing algorithm that uses the string length of the host name in the requested URL, while specifying primary and secondary cache servers for each hash bucket. If needed, the cache can be made transparent to the user agents so that no proxy configuration is necessary at all.

3.1.2 A Distributed WWW Cache

[Kurcewicz et al., 1998] suggests and implements a prototype for a distributed WWW cache cluster, based on functional decomposition, filtering algorithms for what is cached in main memory, and a two-level routing scheme for load sharing by routing arrays to avoid a single point of failure.

They criticize the solution based on automatic configuration that is presented in [Danzig, 1998] (see section 3.1.1 above) for having problems when caching popular web servers; the load distribution becomes uneven because the hash function directs requests for the one server address to the same proxy every time, and that the auto-configuration scheme relies on the client web browser, although it offers a workable solution for persistent HTTP connections. Likewise, they criticize traditional hierarchically connected caches for using available main memory too liberally, and for creating problems with peering relation communication, since ICP (Internet Caching Protocol) does not scale well within a hierarchical structure. The fundamental problem is, according to the authors, that these solutions implement each cache as a monolithic entity:

> No attempt is made to coordinate meta-data and document place-

[3] Secure Sockets Layer

[4] Uniform Resource Locator

[5] Multipurpose Internet Mail Extensions is a standard mostly used for sending more complex content than plain text via the Internet, and is mainly described in RFCs 2045 [Freed and Borenstein, 1996a], 2046 [Freed and Borenstein, 1996b], 2047 [Moore, 1996], 2048 [Freed et al., 1996], and 2049 [Freed and Borenstein, 1996c].

[6] Common Gateway Interface

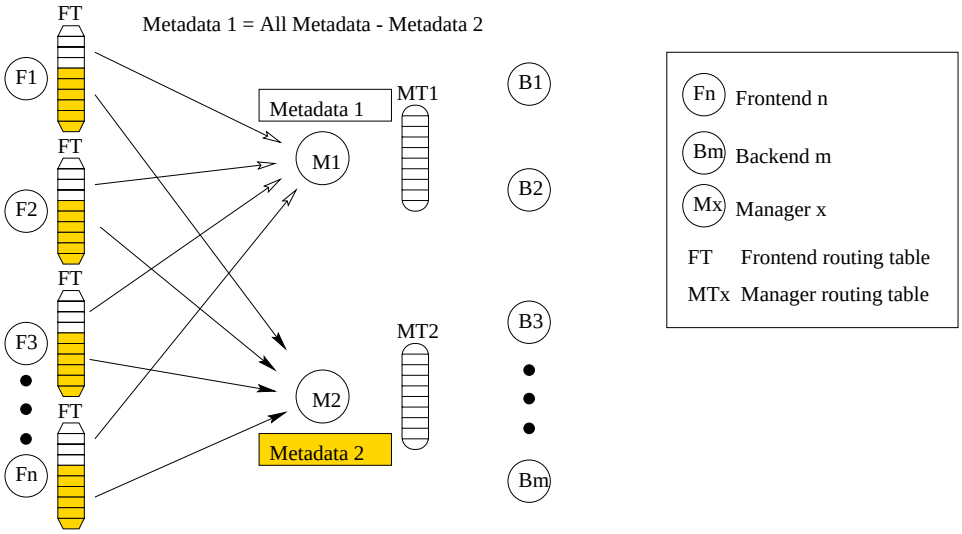

Figure 3.1: A distributed WWW cache with two metadata managers [Kurcewicz et al., 1998]

ment globally.

Their architecture is based on three different types of server functions: frontend, backend and metadata manager. In such a distributed cache system, there can be several of each of these, but each manager manages a different part of the metadata. Figure 3.1 illustrates this.

3.2 Usenet Performance Improvements

To my knowledge, there are only two commercial products for news caching, Inktomi's Traffic Server and DNEWS. There are also a few non-profit initiatives, KNews, NNTPCache, Leafnode and DeleGate. In table 3.1 on page 41, these products are compared with the caching strategies suggested in 1.5.2 on page 27.

3.2.1 DNEWS

DNEWS [NetWin Ltd., 2001] is a popular news server software package that is used by many large scale sites. It can be administered remotely via a web interface. Access to the news server is configurable by pattern matching on domain name, on IP address, and on newsgroup name. It also allows this to be configured on a per user basis.

Articles can be expired not only by age, but also by limiting the number of articles in a group.

Filtering of spam can be done by configuring rulesets. For instance, one can check for excessive crossposting, too many too similar articles in succession, and filter on patterns in any article headers or article body.

For caching and prefetching, DNEWS can do group caching by fetching newsgroup lists from an upstream peer at regular intervals. It allows its clients to use this list for choosing newsgroup subscriptions as if they were directly available on the DNEWS news server. Articles in these groups are prefetched only if the groups are actively requested by clients. From what I have learned, the first article in a so far unread group is not immediately available when it is requested; the user gets a message back explaining that articles in that group will be availble the next time he or she attempts to read a message. This is then a limited form of group caching.

DNEWS also supports another mode of operation with header-only feeds, for offering binaries without storing them on the reading server. In these cases, large articles are fetched from the upstream peer when they are requested. DNEWS claims 90% saved bandwidth and storage space for this mode. I assume this also means that DNEWS caches these articles once they have been fetched, so that it is a form of single article caching.

3.2.2 Inktomi Traffic Server 2.0

Inktomi Traffic Server 2.0 [Inktomi Corporation, 2000] is a proprietary solution for news servers by the Inktomi corporation. It employs group caching, and the cache can be updated on demand, at configurable intervals and, if necessary, by receiving a conventional newsfeed.

The server is designed not only to act as a replacement for the reading or injecting server, but can cooperate with other Traffic Server installations in the same network for load distribution or redundancy, as well as serve upstream of other installations.

An interesting feature is the ability to control the amount of bandwidth dedicated to a client connection for download.

3.2.3 KNews

Robert Krten designed a news server for QNX[7], named KNews, with two basic components: One to get and store articles, and another to keep track of the articles [Krten, 1996].

Krten's improvements are mostly on a QNX-specific level, yielding better efficiency in handling articles in one server, while leaving the problem of distribution alone.

[7] A Unix-like realtime micro-kernel based operating system, [QNX, 2001]

3.2.4 NNTPCache

NNTPCache [Assange et al., 2001] is a program designed for running on the same computer as the user agent, pretending to be a reading server. It hides one or several actual reading servers based on pre-defined patterns. This makes it possible to pretend that one has a virtual newsfeed that is much larger than the actual newsfeed.

NNTPCache is thus a similar solution to Inktomi's (see section 3.2.2 above), but it works transparently with or without other NNTPCache installations. It supports cryptographic solutions for stopping/removing "spam" through automatic treatment of NoCem spam advisories (mentioned briefly in section 1.4.2 on page 24 and 3.3.3 on page 44). In addition, it is possible for the users or groups of users (e.g. based on hostname/IP address) who want it, to turn on regular expression based filtering of headers and content. NNTPCache can also function as an NNTP application proxy.

Whenever a newsreader or other program claiming to be a newsreader connects to NNTPCache and issues reading commands, NNTPCache caches these commands and the resulting data, so this is protocol command caching (see section 1.5.2 on page 27). No other strategy is involved.

NNTPCache expires articles from its cache based on available disk space (counted in terms of blocks, files or bytes) or time, all configurable.

3.2.5 Leafnode

Leafnode [Krasel, 2001] is a news server designed for few users (up to a few tens of newsreaders) on a slow connection (modem based or similar speeds). It works by fetching articles in newsgroups that have been read within a time period (default being one week), and posting articles to that upstream peer on behalf of the users.

Leafnode does not scale very well, and sometimes loses articles as a side effect of how it handles errors.

This is a combination of single article caching, group caching, and time based caching.

3.2.6 DeleGate

According to [Sato, 2001],

> DeleGate is a multi-purpose application level gateway, or a proxy server which runs on multiple platforms (Unix, Windows and OS/2). DeleGate mediates communication of various protocols (HTTP, FTP, NNTP, POP, Telnet, etc.), applying cache and conversion for mediated data, controlling access from clients and routing toward servers. It translates protocols between clients and servers, merging several servers into a single server view with

> aliasing and filtering. Born as a tiny proxy for Gopher in March 1994, it has steadily grown into a general purpose proxy server. Besides proxy, DeleGate can be used as a simple origin server for some protocols (HTTP, FTP and NNTP).

An *origin server* is the same as a regular server for the services it delivers. In the case of Usenet, this means that it would receive and send regular newsfeeds with its peers.

For NNTP, DeleGate can transparently merge connections to different news servers (also based on hierarchy or group), share these connections with several users, whose access is allowed or denied on a per user, host, newsgroup or hierarchy basis.

It also allows creation of aliases for newsgroup names, a feature not supported by regular news servers.

Other features include automatic MIME, PGP encoding/decoding, character encoding/decoding, and protocol conversion for clients (such as NNTP client to POP[8] mailbox and HTTP client to NNTP server).

DeleGate caches data in its "inherent format". What this means for news, is that when someone requests a news article through DeleGate, it caches the raw article data as is, without additional work. As with NNTPCache, this is a form of protocol command caching or single article caching.

Data is stored for a period of time on the system DeleGate runs on, similarly to how NNTPCache does it, or it can connect to another caching server using ICP.

3.3 Other Work

This section presents other work than caching and specific news servers that has been done to help relieve the problems with the size of Usenet feeds.

3.3.1 Satellite Newsfeeds

There are a few commercial actors that deliver streaming Internet data over satellite to cheap satellite ground stations, e.g. Cidera[9], iBeam[10] and Loral[11].

The advantage of a satellite stream for a newsfeed is that it offers reasonably high bandwidth from the satellite feed provider to the satellite ground station. These low-cost stations are sometimes called VSATs[12] [Tanenbaum, 1996] (section 2.8.1, p. 165). Such satellite feeds are one-way only, and traditionally

[8] Post Office Protocol
[9] www.cidera.com
[10] www.ibeam.com
[11] www.loral.com
[12] Very Small Aperture Terminals

Strategy	DNEWS	Inktomi	KNews	NNTPCache	Leafnode	DeleGate
Single article	X			(X)	X	(X)
Protocol command				X		X
Time based					X	
Author						
Thread						
Subject						
Group	X	X			X	
(Sub)hierarchy						
Prefecthing groups/hierarchies						
General header						
Statistical						
News reader controlled						

[tb]

Table 3.1: Strategy comparison for Usenet performance improvements
The table separates from top to bottom between simple strategies, metadata based strategies and more complex strategies.

downstream. Feeding data upstream is usually done via ground-based connections, but it is of course technically possible to have two one-way satellite feeds; one upstream and one downstream.

Satellite feeds are vulnerable to weather and atmospheric disturbances, and have a comparatively high latency because of the geosynchronous orbit, which is 36 400 km above ground level. The minimum round-trip latency is about 480 ms, because of the speed of radio waves[13] [Pratt and Bostian, 1986]. Comparatively, ground based feeds will not have to "travel" more than a fraction of this distance.

For satellite feeds to take off the load of existing ground based lines, it is therefore necessary that the ground based newsfeeds are artificially delayed for a few seconds to have a good margin for the articles to arrive through the satellite feed first.

In order to receive a satellite feed, Cidera provides a satellite dish, a satellite receiver box, a switch, and a special service adapter.

Cidera uses 52" dishes, a bit larger than the minimum for VSATs per 1994, but it also offers a better downlink speed; 22 Mbps, compared to 512 kbps in 1994 [Tanenbaum, 1996].

According to [Cidera Inc., 2001], they are able to supply up to 22 Mbps

[13] $\approx$ 300 000 km/s in vacuum

this way, which is close enough to the needed 30 Mbps for a full newsfeed that it ought to suffice for most uses.

This kind of performance is exactly what a reading server for Usenet needs, reducing bandwidth costs significantly compared to ground based feeds. (See [Phifer, 2001] and [Tanenbaum, 1996] section 2.8.3 p. 169.)

3.3.2 Limitating Binary Distributions

A common solution to bandwidth and storage problems is to limit the incoming flow of binaries. In Usenet's hierarchical structure, the subhierarchy *alt.binaries* is there for the purpose of distributing such binaries. In [Salz, 1991], one way of limiting binary distribution is suggested, simply by excluding all newsgroup names containing "pictures" or "binaries". However, this method is not reliable, since there is nothing that inherently prevents people from posting binaries to other groups.

Curt Welch of NewsReader.Com[14] estimated in December 1999 that a text only newsfeed was 2 GB/day, when a full newsfeed — including binaries — was between 50 and 100 GB/day (see section 1.4.1 on page 18 and [Welch, 1999b], section B.2.2 on page 115)). **If** it safe to assume that binaries are the most significant part of volume growth, and thus that text volume has increased in proportion with the number of articles rather than with the total volume of Usenet, then it is reasonable to expect that a text-only feed is around 5 GB/day in June 2001.

3.3.3 Encoding, Compression and Filtering

Most news servers today support 8-bit transport, although the RFC 977 strongly suggests that everything transported as news should be 7-bit data, and RFC 1036 defines the character set as the 7-bit ASCII. Because of this and that 7-bit transport is still common for e-mail, and many newsreaders also are mailreaders, texts using 8-bit character sets such as ISO Latin-1 are often encoded by newsreaders.

Pictures, sound or movies in binary form are not immediately transferable as articles or e-mail, because they are not *plain text* ASCII. Such content — *binaries* — is also encoded in plain text ASCII.

As I mentioned in sections 1.3.3 on page 16 and 1.4.2 on page 24, most of the volume comes from binaries, and news administrators who do not want all of these try to use filters to save disk space.

In addition to filtering, more size efficient encoding and compression have been suggested as means of reducing Usenet volume.

[14] NewsReader.Com is a web based NSP

Encoding and Decoding

There are several typical encoding schemes in use on Usenet today. For binaries, the most common is probably "uuencode", followed by "base 64". 8-bit text is either sent unencoded or encoded with quoted-printable or base 64.

Quoted-printable encoding — a part of the MIME standard — replaces ASCII control characters and characters not in the ASCII character set with an escape character followed by two hexadecimal numbers. The escape character is '='. If this character appears in the text, it is encoded as "=3D", "3D" being the hexadecimal code for the character number of '=' in ASCII. This scheme is rarely used for encoding binaries, because of a huge overhead, often more than 100% of the original file size.

Base 64 and uuencode expand the size of the binaries so that, roughly, 3 bytes of binary data becomes 4 bytes of text data, which means an overhead of 37% to 40% in practice.

This year, Jeremy Nixon of Supernews[15] has come up with an alternative encoding scheme, called *QPLite*[16]. The scheme takes advantage of the common support for 8-bit transport, and does not encode every non-ASCII byte. Instead, it encodes only the three characters harmful for news transport: NUL to '=00', CR to '=0D' and LF to '=0A', plus the quoted-printable escape character '=' to '=3D'. The overhead for encoding NULs for some types binaries has been estimated to up to 7%, but Nixon claims a common overhead of about 3%.

Decoding depends on the decoding software recognizing the part of the article body that contains the encoded binaries as different from the rest of the article body. [Spencer, 1994] recommends using MIME for specifying character set and encoding, so that it is easier for software to treat other than 7-bit articles correctly. MIME can also be used for encapsulation of combinations of binary and text as different parts of messages. This encapsulation makes it straightforward to treat each such part separately.

Material encoded with uuencode, base64 and quoted-printable are easily recognizable for software even without MIME encapsulation, and this has probably stopped such encapsulation from being used frequently. A series of tests performed in **alt.binaries.news-server-comparison** revealed that only a few newsreaders failed to decode QPLite encoded material with their regular quoted-printable decoders. It is probably trivial to modify these to also support decoding of QPLite.

Compression

Recent discussions in the newsgroups **alt.binaries.news-server-comparison** and **news.software.nntp** have led to suggestions on using end to end (usu-

[15] Supernews is a major NSP

[16] "quoted-printable light"

ally user agent to user agent) compression of article bodies and some headers to improve the conditions for reading servers.

Many of the binaries are already in some compressed form, as is the case for MPEG movies, JPEG images, and MP3 sound. This makes it difficult to compress them further.

On the other hand, text content including article headers compresses well.

This can be combined with other efforts, such as more efficient encoding in the form of QPLite or similar standards, and more intelligent caching or prefetching.

Filtering

As mentioned in sections 1.4.2 on page 24 and 3.2.4 on page 39, news administrators use various filtering techniques to limit the impact of unwanted traffic, such as spam, binaries in newsgroups that are not intended for binaries, and HTML postings to newsgroups where that is not allowed according to the newsgroup charter.

For years, there have been heated debates about the efforts against spam on Usenet. One attempt of dealing with the spam was to send out cancel messages for nearly each spam message that went out. In several waves, this has escalated in to periodic wars, with tens of thousands of simultaneous cancel messages and re-posting of original messages across the net. The problems for news sites around the world varied greatly. On at least three occasions, a site I administered collapsed under the flow of such messages.

These control message attacks have partially been caused by the HipCrime software, which is constructed for generating and sending massive amounts of control messages. News administrators have responded with creating NoCem spam advisories and software for dealing with these advisories automatically, removing the spam without going through the built-in control mechanisms of Usenet. Other software, such as Jeremy Nixon's Cleanfeed[17], filters incoming and outgoing articles before the news server sees them. Spam Hippo is yet another approach, and implements filter persistence, while Cleanfeed stores everything in main memory.

A few years back, there was also a research project called GroupLens that applied collaborative filtering to Usenet [Konstan et al., 1997]. It worked by using metadata servers external to Usenet for evaluation based filtering of Usenet content. These servers kept track of user evaluations about articles that had been read, and from these results predicted what the users would want to read. User agents were modified to transparently connect to these servers and display a prediction of whether the user wanted to read the articles in question or not. In [Ingvoldstad, 1998], I saw problems with the

[17] Currently maintained by Marco d'Itri

scalability of the GroupLens model from one modest workstation for a fairly large number of users (10 000) reading a small number of newsgroups (10-20). This model remains an interesting experiment, but it requires that someone reads and evaluates the articles for it to work. Therefore, it will not work if only a single NSP implements it; several will have to cooperate, and then we run into scaling problems.

3.3.4 Header-Only Feeds

Between December 1999 and March 2000, there were discussions in the newsgroup **news.software.nntp** on so-called "header-only feeds", and the implications that would have for news distribution performance.

A header-only feed implies that only the headers of articles are distributed instead of both headers and body. News server software such as **Diablo** support this option already.

In its simplest form as suggested by Curt Welch in an article to **news.software.nntp** in December 1999 ([Welch, 1999a], included in section B.2.1 on page 111), the header-only feed means that each news server at first only receives the headers of the articles posted. When the user requests an article through the user agent, it asks a specified server (listed in a new header, "Body"), for the article body. The server then caches the article for other users. The *body server* would be the injecting server, or a separate server (at the same site, apparently) that gets copies of all articles posted through the injecting server.

In other words, this changes at least parts of the distribution system to be pull-on-demand, instead of push, and it is a suggestion along the lines of a combination of the **Single article caching**, **General header caching** and **Injecting server repository** strategies mentioned in section 1.5.2 on page 27 and discussed in sections 4.1.1 on page 49, 4.2.7 on page 54 and 4.4.2 on page 56.

Followups to Curt Welch's initial article have pointed out several weaknesses, such as server capacity and bandwidth problems when one body server is asked by a multitude — up to several thousands — of servers for article bodies.

Other problems include lack of redundancy; news becomes more like the WWW, because availability depends on the stability of the body servers in terms of individual line capacity, computing power, system stability, and uptime.

Additional suggestions on **news.software.nntp** include the use of additional strategies, such as **Reverse path lookup** and a central news backbone, where servers generally keep everything.

In general a header-only feed is at a comparatively manageable size compared to that of a full newsfeed. The volume of these headers are between 1 and 5 GB/day. [Welch, 1999a] has 1 GB/day, and a few sampled measurements

of NOV data indicates that the average size of those are somewhere between 300 and 500 KB per article. It should be safe to assume it is less than 5 GB/day, unless header sizes are more than six time greater than what is stored in NOV. See also section 1.3.2 on page 16

3.3.5 Administrative Improvements

The following software is mentioned in "Usenet Software: History and Sources" [Spafford and Moraes, 1998].

Gup

Gup — the Group Update Program — is a semi-automatic system for updating newsfeeds remotely. This adresses part of the problem with the grey area of groups mentioned in section 1.6 on page 29, in that it becomes easier for news administrators to add groups on request from users. This effort does not act directly upon request from users, though; the administrator in question will have to approve the request.

The README file for Gup [Delany and Herbert, 1993] explains the operation of the program in the following way:

```
Gup, the Group Update Program is a Unix mail-server that lets a
remote site change their newsgroups subscription without requiring
the intervention of the news administrator at the feed site.

Once installed, the news administrator at the remote sites simply
mails commands to gup to make changes to their own site's
subscription list.  Not only is no intervention required at the
feed site, but gup checks the requests for valid newsgroup names,
patterns that have no effect and so on, giving the requester
precise information about the effect of their subscription.
```

Dynafeed

Dynafeed [Templeton, 1993] allows news administrators to give remote control over feeding of news on the group level. There is one set of programs for the upstream peer, and one program for the downstream peer. The upstream peer checks incoming e-mail that controls the feeds, similarly to Gup. The downstream peer may run a program for scanning of the user agent's subscription list, so that it only receives those groups that are actually being read. This information can also be maintained manually by the news administrators. The weakness of this is that different user agents keep subscription lists in different ways, and that it would be difficult to maintain.

	Site A	Site B	Site C
Number of peers	5	30	"enough"
Number of users	< 2 000	< 10 000	"many"
Articles accepted/day	1.2-1.3 million		
Volume accepted/day	140 GB	190 GB	220-250 GB
Reduction by filtering on articles	2%	*not answered*	*not answered*
Reduction by filtering on volume	50%	*not answered*	12-14%
Load balancing	no	maybe	yes
Header-only feed	yes	no	*not answered*

Table 3.2: Responses to questions in news.software.nntp on how news administrators handle Usenet feed size

3.3.6 Current Practice

I asked some questions in **news.software.nntp** at 2001-04-30, and got answers from three news administrators. Two answered by e-mail, and a third answered in the newsgroup shortly after I had posted an anonymized summary of the two responses 2001-05-27. Table 3.2 shows key data from the answers, which are included in the appendix, section A.4 on page 105. The sites and their news administrators were promised anonymity. Site A is medium sized, Site B is fairly large, and Site C is huge, and could be considered part of the Usenet backbone.

Since there were so few answers, it is difficult to say anything conclusive with basis in this alone. Thanks to prior discussions and informal interviews with other news administrators, I have been able to establish that at least limiting binary feeds and filtering are common practice among news administrators today. The combination of prefetching and caching is considered interesting, but not much has been done to experiment with it. The exception is group caching, as implemented by Inktomi and Leafnode (sections 3.2.2 and 3.2.5).

Chapter 4

Discussion of Caching Strategies

Unless we choose against supporting existing user agents, including legacy software, which will make the introduction of a new news distribution system rather unrealistic, it is important for the usability of a new news distribution system that the structure of the underlying system remains transparent to the user agent. I will therefore mainly focus on solutions that will not necessitate changes in the user agent nor additional user interaction.

For deciding which solutions are good and which solutions are bad, data on how users read articles must be collected. This usage pattern must then be compared with what patterns the different solutions depend on for working.

For each caching strategy below, see also the introductory explanations in section 1.5.2 on page 27. As mentioned there, these strategies are derived from my own experience.

Most of the caching strategies depend upon the use of an upstream peer that can provide articles on request. The last section in this chapter discusses those strategies mentioned in 1.5.2 on page 27.

4.1 Simple Caching Strategies

Simple caching strategies do not require metadata to function properly.

4.1.1 Single article caching

Since the proxy only fetches the article requested, that is, only single articles, this method is effective only if an article is likely to be read by more than one person or more than once.

I believe the effect of this kind of caching — as long as it is not combined with any other methods — is good but limited. The cache will get a miss for each new unique article read event. If unique articles usually have several read

events within a reasonable timespan, the comparative overhead for fetching single articles may be relatively low.

I expect this method to be decent to good in hit rate, bandwidth, and disk usage. User perceived performance can range from good in the case of cache hits, to bad in case of misses where the article requested is unavailable from the nearest upstream peer.

4.1.2 Protocol command caching

This method is simple to implement, in that it does not try to be too smart about electing things to cache, but rather depends on actual usage patterns as they occur. It does not force the proxy to fetch additional articles ahead of time.

As with single article caching, it presents a minor overhead whenever a previously unencountered command is received. It then has to pass the command along to the actual reading server. While it is possible to uncertainly predict some of the commands that will be encountered in the future, it does not make prefetching based on these commands alone a good choice, as the commands are on a lower level than that of article, group, and hierarchy. Those are arguments to the commands, and specific examination and caching of those arguments belong to more coarse grained caching approaches.

These problems may make it hard to make effective use of protocol command caching for sites with many users with varied usage patterns, but it works well for smaller sites, according to [Assange et al., 2001].

This method should, like single article caching, perform decently in terms of hit rate, bandwidth and disk usage. However, in terms of user perceived performance, I expect it to perform poorly, because of the lower level of caching compared to single article caching.

4.1.3 Time based caching

This approach assumes that if someone reads an article, other articles posted around the same time as that article are likely to be read as well. Since articles are spread over a great number of newsgroups, this is probably most efficient when users subscribe to many newsgroups and read them often.

The approach may prove to be more efficient if it is done on a group by group basis, if users are loyal to a specific set of newsgroups and follow them. If implemented with current NNTP commands (listed in tables 1.4 on page 11 and 1.5 on page 11), it can easily be done on a per group basis, since the NEWNEWS command only checks for new articles in a specific group. Checking all groups — as this approach calls for — can also be done, but the overhead increases with the number of groups available on the server, so I do not think this method is very efficient if used alone.

Another point is that this is nearly exactly what news is about today, since it prefetches an amount of articles, but it is pulling articles rather than having them pushed, which will result in a small overhead in network traffic because of the extra commands required.

Considering that there are over 20 000 different newsgroups, there may not be any relation at all between when an article is posted (or when it arrives) and the articles people read, unless this is limited on a per group basis. Also, if the news provider has a fairly large mass of users, this may quickly break down if the provider has users reading news constantly. The effect of the proxy would then be that of a pulling news server instead of one receiving a push, and instead of decreasing, the bandwidth and storage demands remain or increase.

Compared to the two previously mentioned strategies, I think this method has a higher degree of bandwidth and disk usage, as well as bad user perceived performance.

4.2 Metadata Dependent Caching Strategies

The following strategies require a feed of metadata from their upstream peer. As a minimum, NOV data must be available, and for some methods additional headers are needed. It is probably practical to use a complete header-only feed for that purpose.

4.2.1 Author caching

It may be a reasonable assumption that when someone has requested an article by one particular author once (or a given number of times), that someone wants to read more articles by that person. It is difficult to say whether this is effective or not in general without measuring usage patterns, but it is pretty certain to be effective in limited cases, such as Linux developers looking for articles by Linus Torvalds. If this method should be effective, each article prompting a read event should be written by a comparatively small group of different authors.

A negative effect is that the pattern matching necessary for performing author caching will require substring matching in news headers, and this incurs a performance penalty in terms of processing power.

Author caching should have about the same degree of bandwidth and disk usage as time based caching, but user perceived performance ought to be even worse because of the pattern matching.

4.2.2 Thread caching

Caching per thread is effective if users can be assumed to be interested in a specific discussion when they read one article in that discussion. Since a

thread is supposed to contain all articles in that discussion, the user would then be able to find the rest of the discussion already in the cache. Other users may be interested in the same discussion because of word-of-mouth, or because they already have read articles in the discussion before.

The methods probably needs adjustable parameters, such as for how long after an article in a thread is read the proxy should keep track of a thread and add articles to the thread cache when they become available at the upstream servers. Differentiated parameters for low-traffic and high-traffic periods, such as vacations and semester start at a school are also something to consider.

For all read events, this method seems to be able to get an even better hit rate than single article caching, because it has the chance of getting articles from the read threads into its cache before other articles in the thread are requested. Another way to look at this is to have a standard pushed feed to the proxy, where the proxy simply gives a negative response to any articles that do not belong to the threads it wants to cache, which may save some response time if the other articles in the thread are offered after the first. With the pushed feed, the proxy will not attempt to fetch any articles itself unless there are cache misses. In the event that an earlier rejected article is requested, the pushed feed solution will cause additional network usage compared to only pulling articles.

It should be possible to measure some of the potential success of thread caching by building the reverse References tree manually from the References header of articles read by users and comparing the number of such threads and subthreads with the number of article read events.

I believe this strategy has the greatest potential of all single strategies mentioned here, provided it is implemented by pulling articles. This is difficult to back up with hard data, and would require detailed statistical data from a practical implementation to test its limits.

I expect thread caching to have a high hit rate and user perceived performance, as well as decent bandwidth and disk usage.

4.2.3 Subject caching

Caching by subject is a different strategy than caching by thread. A thread may continue while the subject changes, and the same or a similar subject may be found across several threads. Sometimes, threads are broken up (no References header) when the subject changes, yet the subject retains a part of the original subject. The new subject can then be called a continuation of the old subject.

For instance,

```
Subject: Netscape doesn't load my home page
```

---

could typically be changed from one article to the next to:

```
Subject: Stupid users (was: Netscape doesn't load my home page)
```

If the thread was broken up, the text following "was" would be our only clue that this was a followup article to the original one. Some people break up threads on purpose, as per RFC 1036 recommendations, others do it by accident because they do not know how to use their user agent or their user agent does not do things the right way, leaving the subject unchanged.

Considering that the topic of a thread may change without the References header being cut down on, thread caching may cache a lot of articles that are not of interest. Subject header caching would help, provided that we could rely on people changing subjects when they change topic. Unfortunately, we cannot rely on that, so the effectiveness of this method is dubious.

There is another problem with subject caching, and that is that user agents do different things when treating subjects. In the cases where non-ASCII characters are present in the subject, these are sometimes encoded with quoted-printable (see section 3.3.3 on page 43 for more about this encoding scheme), and other times with other schemes. If subject caching should take this into consideration, it would require more processing power when examining subjects, and therefore reduce performance.

Another way of doing subject caching could be to look at similarities between articles each user agent is requesting. If there is an overweight of articles with "madonna" in the subject, it is not unreasonable to assume that the users might be interested in more madonna articles, and that pre-fetching those articles would be useful. This does demand a higher degree of analysis work on the server's part, and may not really give a net "profit" in terms of performance, because of the extra processing power required.

Whatever solution is chosen for subject caching, the same problem as with author caching remains; for every single cache miss, the server must perform resource intensitive pattern matching on text strings.

Subject caching can be expected to have poor user perceived performance, perhaps even worse than author caching. Bandwidth and disk usage performance is hard to predict, as it depends on how much of a subject one does pattern matching on.

4.2.4 Group caching

Caching all the articles available in an entire group may be effective if the volume of the group is low, and it is a fair assumption that users reading one article in a group would be interested in other articles in the same group. If the group has a high volume, pulling down the articles will be relatively slower,

and the user might not get the articles requested unless these are fetched on a per-article basis until they are in the group cache. It could also be possible to start the caching process when a user requests headers for the group.

As with thread caching, the method may need adjustable parameters on when the group caching process is supposed to "time out" if nobody continues reading it. Whole groups may be expired if the caching method is exclusive, that is, no other methods than group caching is employed.

One suggested implementation of group caching, is to read each user agent's subscription list. This is impractical, since different software keeps the subscription lists in different ways, and not necessarily in a place that is readable for the reading server. Another way to do it would then be to request the subscription list from the user agent, but that would require changes in all news reading software I am aware of.

Group caching should be effective when there are read events for most of the articles in a group. To measure this, one must know exactly how many articles there are in a group, what their Message-IDs are, and compare those with requests for articles in that group.

While being good for hit rate and user perceived performance, I do not think it will be good for disk usage. Bandwidth usage is also likely to suffer, unless the hit rate is very high.

4.2.5 (Sub)hierarchy caching

This method shares the same problems and benefits as group caching, on a larger scale. It increases the risk that there are unread groups with many articles in them. I believe this method is too coarse grained to be of good use. I will not look further into this one.

4.2.6 Prefetching groups/hierarchies

Since this method works without interaction with a user agent, there are no big differences between this approach and today's, with the possible exception that the proxy might be pulling articles instead of receiving a feed.

If the proxy is going to get all the articles in a group or an entire hierarchy, I do not see the point in pulling the articles, when today's method seems to be working adequately for that purpose.

I expect prefetching of groups and hierarchies to be bad for bandwidth and disk usage, while very good for user perceived performance.

4.2.7 General header caching

Caching only article headers is an approach that will reduce the volume distributed by a great deal. Estimates made by news administrators (see section 3.3 on page 40) indicate that if only headers were distributed, the total

volume of articles would be reduced by a factor of 20 to 100. This is for a full header distribution. Distributing only a limited set of headers, for instance only those that are by default in the NOV format plus the Path header should provide an even more effective reduction in bandwidth and storage demands.

While it is unrealistic to hope that users will be satisfied with only headers, this will mean that unread articles only appear as headers on the servers in question. For most caching methods mentioned earlier in this chapter (not necessarily including group and hierarchy caching, which can be initialized without having the headers first), either some or all headers must be known to the proxy, and so must the group and hierarchy structure.

The general header caching approach is therefore a sound basis for the other caching methods, but not good in itself.

4.3 Complex Strategies

The remaining two strategies are more complex strategies than those mentioned so far. They can combine different strategies mentioned earlier, and possibly provide a higher hit rate for read events.

4.3.1 Statistical caching/prefetching

Automatic measurement of statistical data to figure out how the users read news is a tempting way of combining other strategies without manual reconfiguration. It eases administration of the caching proxy, as well as probably offering the best kind of hit rates for the kind of reading pattern the news administrators want to encourage; low disk usage, low bandwidth usage, both, or neither.

However, if the statistical model is static itself, it could be vulnerable to changes in user behavior patterns. These pattern can change if we are a growing NSP, or a new trend in what is popular among readers appears. This effectively reduces the efficiency and effectiveness of the statistical model until the pattern stops changing, unless it can predict these changes.

4.3.2 News reader controlled caching/prefetching

Relying on the user agent to do anything in a pseudo-intelligent manner for caching is probably not a good idea, but parts of a caching scheme could be done by newer software with such options.

If the user in combination with the user agent can help the proxy with detailing the expected pattern the user will display in requesting articles, it could be made easier to make sure that the articles are already in place when the user wants them, while other articles would have to be fetched by automatic rules, like the ones mentioned earlier in this chapter.

A serious drawback with this solution is that it requires changes in user agents, and that the method only will work for these changed user agents.

4.4 Fetching articles when there is a cache miss

In the event of a cache miss, the proxy will usually try to fetch the article from some other news server. This section deals with the three methods mentioned briefly in section 1.5.2 on page 27.

4.4.1 Group/hierarchy repositories

Group/hierarchy repositories require that there are several, redundant, central news servers, providing some kind of news backbone where articles within a given time frame are available to all the proxy servers.

To provide sufficient redundancy — in case servers, hierarchies, groups or articles for some reason become unavailable — this will probably mean that a few of the news servers with a very high load today will have that very high load in the future also, unless there is a way of distributing that load on other servers.

For a proxy belonging to an NSP that does not keep a group or hierarchy repository, this method is good for disk usage, but probably bad for bandwidth and user perceived performance. In the cases where the NSP serves the repositories itself, the bandwidth will be local to the NSP's network and therefore not a problem. Disk usage may be high for the repository, but low for the proxy.

4.4.2 Injecting server repositories

An injecting server repository seems a lot like the way the WWW works today. A single server (or set of servers) holds the original article (each comparable to a web page), and in order for a user agent to read it, the client software (in this case, the news proxy) has to retrieve the original article.

This may cause great problems for injecting servers with low bandwidth but popular articles, as news proxies around the world attempt to get them. Injecting servers may not be able to guarantee the presence of articles "forever", and this solution is also very fault intolerant. Articles may simply become unavailable because of temporary network problems, and the lack of redundancy can make access time more than noticeably slow.

Injecting server repositories are probably very good for storage, but obviously bad for user perceived performance and bandwidth.

4.4.3 Reverse path lookup

Looking up along the feed path is an interesting technique that has a similar implementation in the Diablo news server. The Diablo news server allows the news administrator to define a set of servers that can provide articles within a certain group or hierarchy. When the Diablo server does not have an article, it forwards the article request to one of the servers in the set.

The reverse path lookup method would expand on this method by defining those willing to provide articles as the same news servers that feeds the relevant hierarchy/group/articles to this server. If those servers do not have the article, they ask their upstream peer. It is of course necessary to look out for loops in the feed graph, to make sure a server that has previously been asked is not asked again.

Another implementation could use the Path header in the article format, and ask each server given an entry there. A problem with this is that the news server itself may not have exactly the same host name as the name given in the Path header.

The news article format only requires that the name given in the Path header is valid and unique for the site. Figure 1.1 on page 5 shows an example of this; "`uio.no`" is not a host name, yet it uniquely identifies the feeding server at the uio.no domain. Therefore, to ensure that such an implementation would work, new syntax would have to be added to the Path header to tell us when a news server there also is willing to serve hierarchies/groups/articles on special request.

Looking up along the reverse feed path to one's upstream peers is likely to be quite efficient in terms of bandwidth usage, and not too bad in terms of user perceived performance. Disk usage is not an issue for anyone but the upstream peer that keeps the articles.

Chapter 5

Findings

In this chapter, I will show and examine the results of data collection from Nextra and Ifi. I do not separate between read events and actual articles being read, unless it is possible and practical to do so. For Nextra, there was only a small amount of data collected, while for Ifi, I had the opportunity to examine each connection to the reading server more closely.

5.1 Statistics from Nextra

The Norwegian ISP Nextra has a customer base of around 500 000, which makes them a fairly large ISP. I have used them to get information on reading habits on a newsgroup level.

They use a fairly simple server setup, with one feeding/numbering server and two reading/injecting servers in a simple DNS based round-robin load balancing scheme. The feeding server numbers the articles, and sends duplicate feeds to the reading servers. This ensures that the same articles have identical numbers on each reading server, which in terms of available articles makes it transparent to the users which server they connect to. Reading Server 2 has comparatively little article storage, and acts as a slave towards Reading Server 1 in the cases where a user agent requests an article only Reading Server 1 has. See figure 5.1 on the following page.

The feeding server performs filtering of newsgroups, binaries and other unwanted articles before delivering articles to the reading servers, reducing the total size of incoming articles by at least 80%, according to Nextra.

Nextra's news administrator, Stig S. Mathisen, provided me with statistical data from their reading servers. A small test in advance revealed that if we were to extract data for all the available groups, it would take two or three days, not including manual work. For the sake of simplicity, we selected 309 groups (see appendix A.3 on page 99) that were evenly distributed from an alphabetical list of the groups that were accessed by users.

Data was collected for the time period 2001-05-12 to 2001-05-18 as a

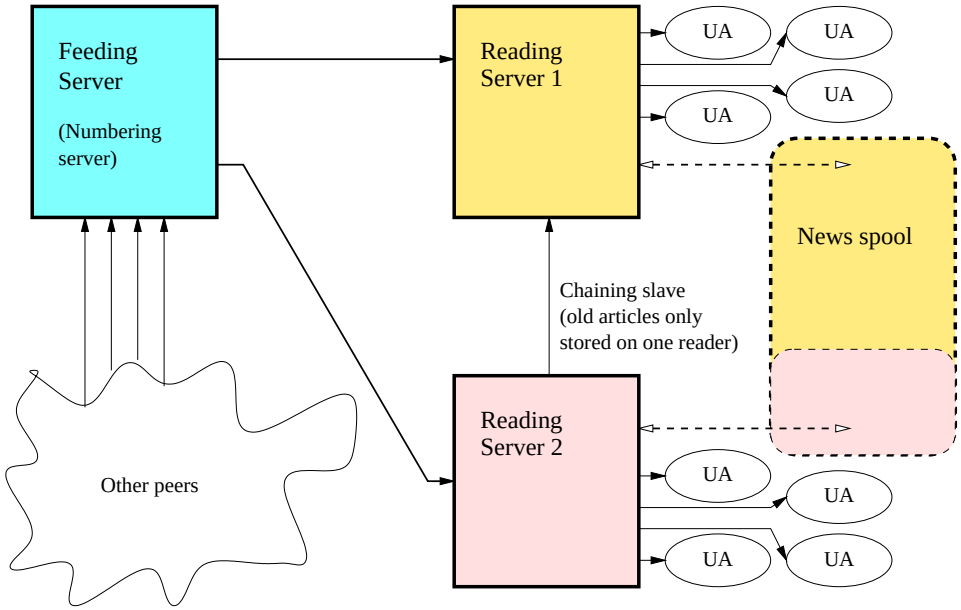

Figure 5.1: Nextra: News system configuration, mid May 2001

whole, showing the number of ARTICLE and XOVER commands per group. Another term for ARTICLE command is *article request*, which I will be using for variation.

In the section about storage methods in the introduction (1.2.2 on page 5), I explained briefly what the XOVER command does. The count of XOVER commands show how often user agents lists information (table 1.6 on page 14) about the available articles in a group, while the ARTICLE commands show how often an article with both headers and body is required.

5.1.1 Error Sources

Because of the coarseness of measurements and that the measurements were made at the reading servers, I can identify the following significant source of errors:

Reading Server 2 acts as a slave towards Reading Server 1. When an article does not exist on a slave, it will pass along the request for that article to its master, in this case Reading Server 1. These requests are registered in both Reading Server 1 and Reading Server 2, and were not possible to separate on Reading Server 1.

Because of this uncertainty, I have kept the data in the three separate data sets they were generated in: one for each reading server, and an additional for the articles on Reading Server 1 that were not fetched in slave mode (“noslave”).

Server	Groups count	Groups %	Commands ARTICLE	Commands XOVER	ARTICLE/ XOVER
Reading Server 1	165	54%	92 429	244 988	37.7%
Reading Server 2	123	40%	49 828	120 603	41.3%
Reading Server 1, noslave	140	45%	54 438	164 802	33.0%

Accuracy is 1% for groups, 0.1% for ARTICLE/XOVER commands

Table 5.1: Nextra: Total ARTICLE and XOVER commands

Server	Groups count	Groups %	XOVER commands count	XOVER commands %
Reading Server 1	20	6%	1 556	0.6%
Reading Server 2	16	5%	6 020	5.0%
Reading Server 1, noslave	22	7%	1 753	1.1%

Accuracy is 1% for groups, 0.1% for XOVER commands

Table 5.2: Nextra: Groups with XOVER commands and no ARTICLE commands

5.1.2 Data

Table 5.1 shows the total number of ARTICLE and XOVER commands with the number of groups these commands were issued in. On average for the two servers, less than 50% of the groups saw both ARTICLE and XOVER commands.

Table 5.2 shows the number of groups with XOVER commands but no ARTICLE commands, and the total number of XOVER commands for those groups. These XOVER command counts tell us that the groups were accessed, but that the user or user agent (by kill file or score file) decided that the articles were not wanted during the period of measurement.

Table 5.3 shows the total number of read events for groups that had no

Server	Groups Count	Groups %	ARTICLE commands count	ARTICLE commands % of total
Reading Server 1	25	8%	1 548	1.7%
Reading Server 2	16	5%	84	0.2%
Reading Server 1, noslave	18	6%	511	0.9%

Accuracy is 1% for groups, 0.1% for ARTICLE commands

Table 5.3: Nextra: Groups with ARTICLE commands and no XOVER commands

	Groups	
Server	**Count**	**%**
Reading Server 1	144	47%
Reading Server 2	186	60%
Reading Server 1, noslave	169	55%

Table 5.4: Nextra: Groups with no ARTICLE or XOVER commands

Server	**Sum**	**Min**	**Avg**	**Max**	**SD**
Reading Server 1	244 988	0	792.84	109 734	6767.99
Reading Server 2	120 603	0	390.30	50 379	3067.04
Reading Server 1, noslave	164 802	0	533.34	69 638	4401.31

Table 5.5: Nextra: Statistics on XOVER commands per group

XOVER commands. This means that these article requests are based on other information than what the user agents would have gotten from issuing XOVER commands on that server. Some of these article requests for Reading server 1 are likely to come from the slave, and could be related to the lack of article requests seen in 5.2 on the page before. Groups that have seen no interest from users, that is, groups without both ARTICLE and XOVER commands, are shown in table 5.4.

Figure 5.2 on the next page illustrates the number of XOVER commands for each group in the sample, sorted for each reading server by number of XOVER commands. Similarly, figure 5.3 on page 64 show the number of ARTICLE commands. The numbers have been put on a logarithmic scale to make the graphs readable. A few groups have the available articles examined often and also have many associated article requests.

The most popular group — **no.alt.frustrasjoner** — has 109 734 XOVER commands and 58 949 ARTICLE commands on Reading Server 1 (69 638 and 29 354 in noslave mode, respectively), and 50 379 XOVER commands and 32 709 ARTICLE commands on Reading Server 2. The next groups among the top five differ in popularity for the two servers and noslave mode. Tables 5.6 and 5.5 help to complete the picture. The average numbers for XOVER and

Server	**Sum**	**Min**	**Avg**	**Max**	**SD**
Reading Server 1	92 429	0	299.12	58 949	3373.15
Reading Server 2	49 828	0	161.26	32 709	1869.39
Reading Server 1, noslave	54 438	0	176.17	29 354	1703.63

Table 5.6: Nextra: Statistics on ARTICLE commands per group

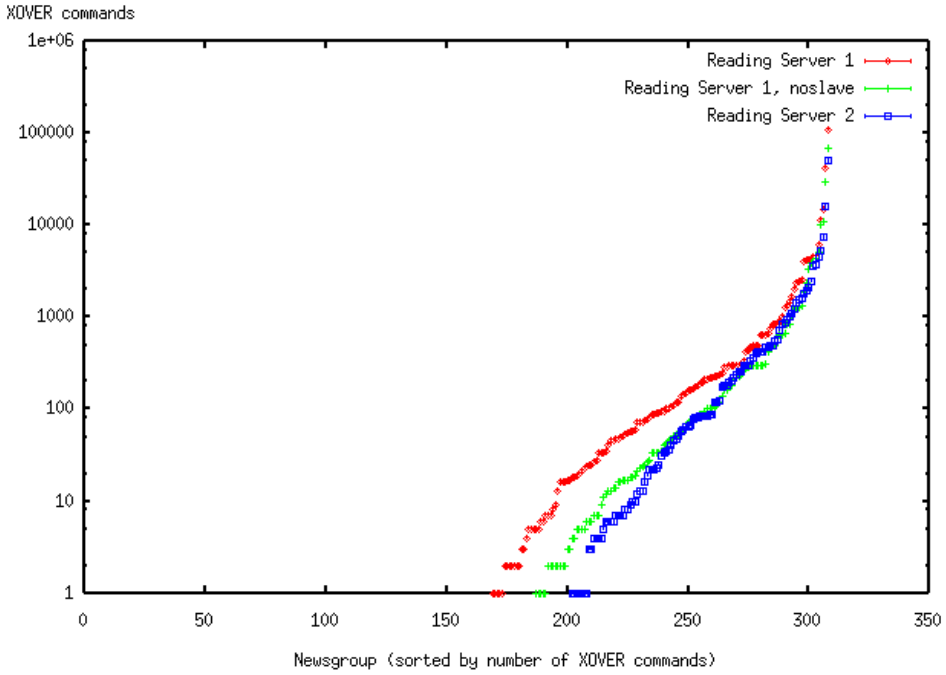

Figure 5.2: Nextra: XOVER commands per group
(Groups are numbered from 1 to 309.)

Server	Groups	Sum	Min	Avg	Max	SD
Reading Server 1	165	244 988	0	1 484.78	109 734	9206.20
Reading Server 2	123	120 603	0	980.51	50 379	4801.34
Reading Server 1, noslave	140	164 802	0	1 177.16	69 638	6480.58

Table 5.7: Nextra: Statistics on XOVER commands per group with both ARTICLE and XOVER commands

ARTICLE commands per group vary between servers, and so does the standard deviation. Reading Server 1 has both the greatest number of XOVER and ARTICLE commands, and seemingly the greatest variation if we take only the standard deviation numbers into account, but figures 5.2 and 5.3 on the following page give a different presentation; the differences between servers are not very great.

5.1.3 Preliminary Evaluation

Although the data does not tell us anything about what to think about reading patterns in the terms of threads, subject, author, etc, it does seem to show that there is room for improvement on the group level. If all groups with zero

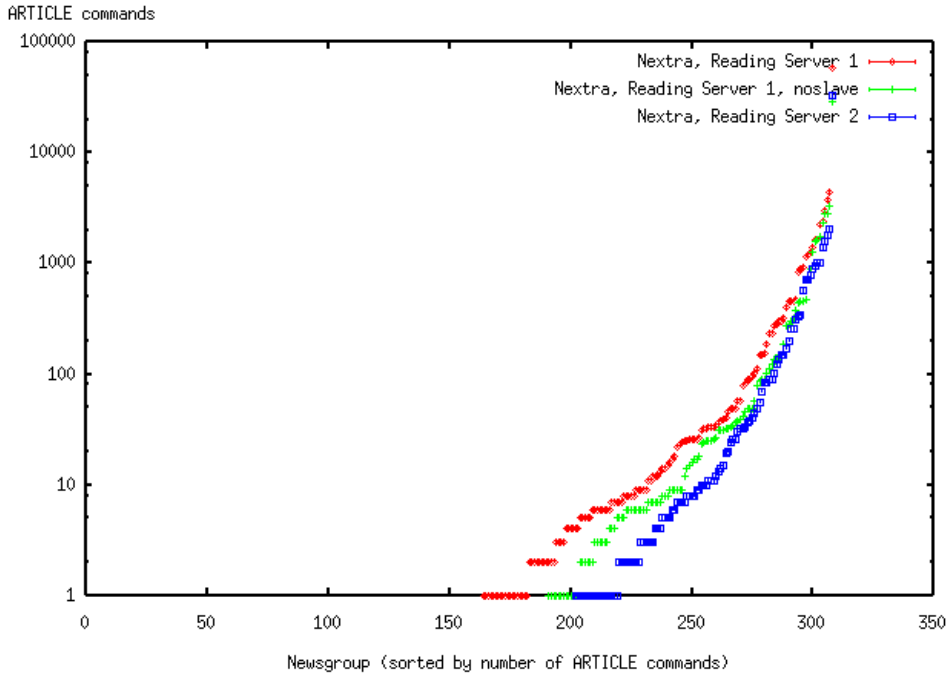

Figure 5.3: Nextra: ARTICLE commands per group
(Groups are numbered from 1 to 309.)

Server	Groups	Sum	Min	Avg	Max	SD
Reading Server 1	165	92 429	0	560.18	58 949	4600.21
Reading Server 2	123	49 828	0	405.11	32 709	2946.25
Reading Server 1, noslave	140	54 438	0	388.84	29 354	2514.60

Table 5.8: Nextra: Statistics on ARTICLE commands per group with both ARTICLE and XOVER commands

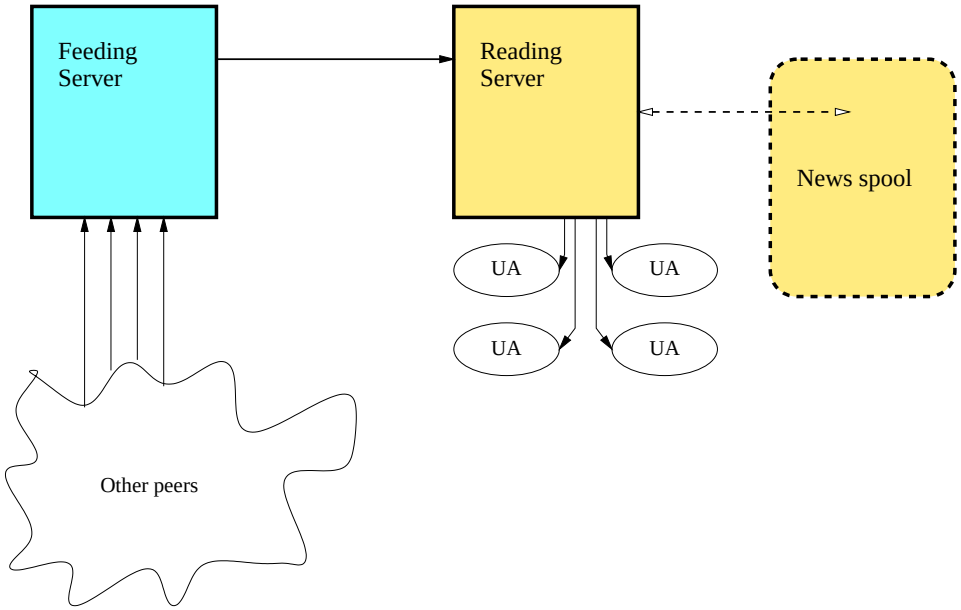

Figure 5.4: Ifi: News system configuration, July 2001

XOVER and ARTICLE commands were removed, the statistical data would be as presented in tables 5.7 on page 63 and 5.8 on the facing page. The standard deviation would have increased a lot, but we would also have reduced bandwidth and storage demands for the reading servers. Group caching seems to be a good choice here, but other strategies may be better over all, considering that user agents check the contents of groups far more often than they ask for articles.

5.2 Statistics from Ifi

Ifi has a relatively small user base of 3-4 000. They can satisfy the news reading users with a very simple setup, based on one feeding server (Diablo) and one reading/injecting server (INN 2.2). See figure 5.4. As opposed to Nextra, the feeding server does not number the articles.

Ifi has several different group retention policies. One local hierarchy — *ifi* — is archived indefinitely, a few others are archived between seven and 90 days. The rest of the groups and hierarchies are stored in different cyclic news spools, depending on how important they are considered for the users.

The collection of data at Ifi started 2001-07-07 01:50 MET DST and ended 2001-07-12 16:11 MET DST. Since Ifi is an academic department of the University of Oslo, the timeframe may be unrepresentative of typical Usenet reading habits, because a lot of students and some employees are on summer vacation. I am not aware of any studies that show how this would influence reading

habits. I think it is reasonable to claim that this is a low traffic period.

Data were collected from 3 538 sessions with significantly more detail than at Nextra. A session is initiated when a client connects to the reading server. This means that it is possible to have a session without having any commands executed by INN. For each session, I logged to file each of the NNTP commands issued by the client. Whenever an ARTICLE, BODY or HEAD command was issued by the client, I also logged the headers for the articles in question, since these commands force a read event in the reading server. This generated a total of nearly 900 MB of data. After data collection, the data was re-formatted to reduce unnecessary redundancy.

I also tried to use data from the reading server's active file to get information on available articles in the news server, since the bulk of Ifi's articles are stored in cyclical spools that cannot be read without a major programming effort. The active file is a file that contains a list of all available newsgroups, along with the article numbers of the last and the first available articles in these groups. However, some random samples revealed that at in many groups, these numbers are imprecise at best. Between the first article number in the group and the last, there can be a gap of several thousands before the next article. In my random sample of 21 groups, there were a claimed total of 31 743 articles available, but according to extraction of headers from the overview database, there were 28% less, 22 663. Extraction of overview data for even such a small number of groups took between ten and fifteen minutes (I did not time it exactly), while it also put additional strain on the reading server. If I were to extract overview data for the groups that were requested in the time period of measurement, that would have taken many hours, effectively also disrupting usage of the reading server for Ifi's users. I therefore chose to let it be.

5.2.1 Error Sources

The patches to the news server software has a small shortcoming; it writes its data to log files with file names derived from the time a session was initiated, with a coarseness of one second. In the case of 319 article read events, this resulted in corrupt output when two or more sessions were writing to the same file simultaneously. For an additional 2 article read events, the Message-ID header was treated as unique when counting Message-IDs, but they were in fact duplicates.

If a command is issued without a positive response from the server, it still counts as a command in my code. In the case of read events, this would happen if an article is requested, but it does not exist in the reading server's news spool.

My scripts for extracting data from the log files also have a few shortcomings, in that they are not very tolerant towards minor corruption in the log files. In addition to this, NNTP command counts presented later are the result

Command	Count
ARTICLE	51 284
AUTHINFO	14
BODY	622 899
GROUP	41 927
HEAD	624 992
HELP	1
IHAVE	0
LAST	1
LIST	359
LISTGROUP	632
MODE	3 464
NEWGROUPS	762
NEWNEWS	0
NEXT	0
POST	20
QUIT	0
SLAVE	0
STAT	1
XGTITLE	10
XHDR	13 941
XOVER	11 171
XPAT	0
XPATH	0

Table 5.9: Ifi: Total NNTP command count

of a count of substring matches for "] `COMMAND`" (where COMMAND is the NNTP command in question). If these strings occurs in the headers of a news article, they are counted one additional time.

Since I run calculations in several steps, based on data already re-formatted by other scripts, there are some points of corruption that sometimes lead to differing numbers between different kinds of measurements. Usually, these differences are fairly small and negligible.

5.2.2 Data

Table 5.9 shows the count of NNTP commands by clients connecting to Ifi's reading server. See tables 1.4 on page 11 and 1.5 on page 11 for descriptions of these commands.

Five of these commands cause a read event in INN 2.2. These are:

- ARTICLE

Groups		Commands		
count	**%**	**ARTICLE**	**XOVER**	**ARTICLE/XOVER**
1 099	9.8%	51 824	11 171	464%

Table 5.10: Ifi: Total ARTICLE vs XOVER commands

- BODY
- HEAD
- XHDR
- XPAT

The total number of such read events is 1 313 115. There were no XPAT commands executed. Due to implementation details in INN 2.2, I ignored XHDR in collection of headers based on read event. This means that the the total number of read events measured except XHDR is 1 299 175. However, due to the output corruption mentioned earlier, only 1 182 605 read events were recognized, and of those, only 856 091 caused my code to exctract headers for the article in question. 855 772 of these succeeded without any noticeable corruption, and that will be the basis for further calculations. It is also possible that the reading server failed to deliver any data in response to some of the read events, for instance if the article asked for did not exist.

These read events were caused by a total of 2 143 sessions. The remaining 1 395 sessions caused other types of events.

In the six days of measurement, Ifi had an average of 11 257 available newsgroups. These newsgroups had on average about 3.95 million unique articles available. The incoming newsfeeds had a total of approximately 1.26 million unique articles with a volume of approximately 20 GB. Only 1 099 of the newsgroups were entered with the GROUP command.

5.2.3 Comparison with Nextra

For comparison with Nextra, I have measured against the ARTICLE commands in groups that were entered at Ifi.

A comparison of table 5.10 compared with table 5.1 on page 61 shows that Ifi has many more ARTICLE than XOVER commands.

There are two possible reasons for this. One is that dialup users — which Nextra presumedly has a lot of — use software that uses XOVER a lot to handle the overview metadata before deciding whether to download specific articles, and that they often do not download actual articles at all. Ifi's users have access to news via Unix computer labs, and most thus read news over the local network, staying online. These users do not have to conserve their personal

Groups		XOVER commands	
count	%	count	%
40	0.4%	408	3.7%

Table 5.11: Ifi: Groups with XOVER commands and no ARTICLE commands

Groups		ARTICLE commands	
Count	%	count	% of total
76	0.7%	12 604	24.7%

Table 5.12: Ifi: Groups with ARTICLE commands and no XOVER commands
There were 50 927 measured ARTICLE commands as a side effect of scripting

bandwidth when reading news, so they have a greater freedom in skimming actual articles rather than settling for the overview information when selecting what to read. Also, if the users use kill or score files (see "Note on Built-in Filtering in Newsreaders" in section 1.2.2 on page 5), the dialup users are likely to filter only on overview data or headers, and if they are offline when reading news, that is also all they have to work with unless they downloaded all the articles in the group before hanging up. The users who have access to the reading server from a local network can filter on additional headers and even the body without worrying about article download time. If the filter attempts a pattern match, it will likely send an ARTICLE command. I know of one such user myself. If he chooses to read a group with 5 000 unread articles, his newsreader's score filter will cause 5 000 read events even before he looks at the first article.

Another point of comparison is that only about 10% of the groups at Ifi saw ARTICLE or XOVER commands, compared nearly 50% at Nextra. The reason may be that Nextra's users are more heterogenous than Ifi's, but I cannot put much weight on this, since the data from Ifi was collected during summer vacation.

Table 5.11 does differs a bit from Nextra's table 5.2 on page 61 in the relative amount of XOVER commands, and significantly in the number of groups.

For groups with no XOVER commands, there is a significant difference between Nextra and Ifi. As table 5.12 shows (compare with 5.3 on page 61), there are a great number of ARTICLE commands for groups where no XOVER commands were issued. Groups that were entered, but without both ARTICLE and XOVER commands are shown in table 5.13 on the following page. The difference from Nextra in table 5.4 on page 62 is great; 96% of the available groups at Ifi saw no particular interest from users, compared to between 47% and 60% in the sample from Nextra. If we look at the number of visited groups instead of the total number of groups, the number is much closer with its 61%

Groups		Visited Groups	
Count	%	Count	%
10807	96.0%	673	61%

Table 5.13: Ifi: Groups with no ARTICLE or XOVER commands

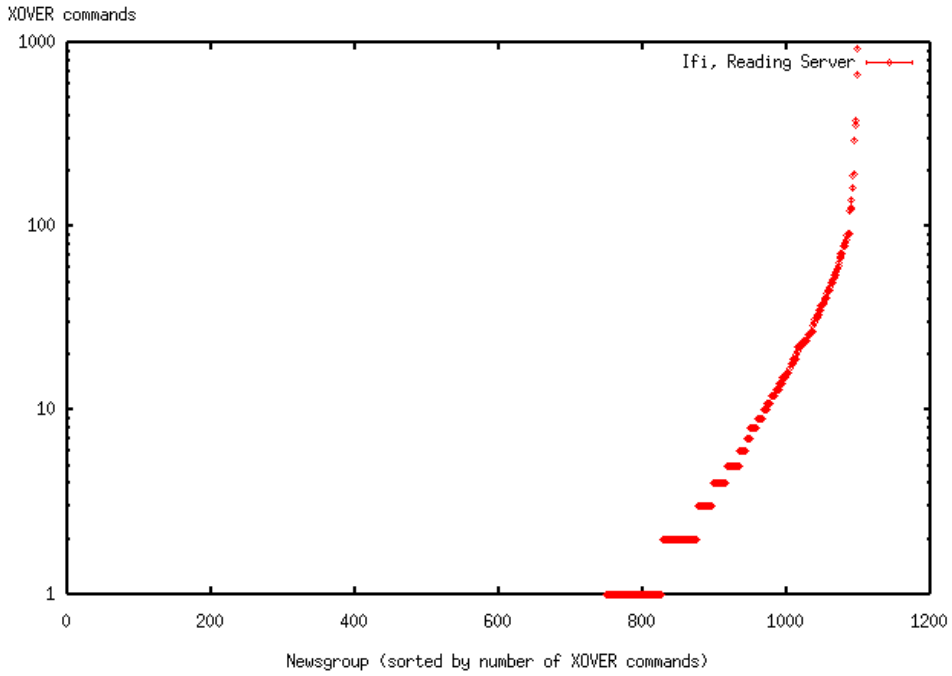

Figure 5.5: Ifi: XOVER commands per group
(Groups are numbered from 1 to 1 099.)

uninteresting groups, but still higher than Nextra.

Figure 5.5 illustrates the distribution of XOVER commands per group for Ifi, and figure 5.6 on the next page show the number of ARTICLE commands. As with Nextra, a few groups are examined often and also have many associated article requests.

Figure 5.7 on page 72 and 5.8 on page 73 illustrate the distribution of XOVER and ARTICLE commands per group with similar data from Nextra, normalized for the number of groups examined at Nextra. The difference is pretty clear in the case of XOVER commands, and show that there is a greater variance for the groups at Nextra. For ARTICLE commands, figure seems to show that there is less difference between Nextra and Ifi.

Tables 5.14 on the facing page and 5.15 on the next page confirm the impression from the graphs while showing that the maximum values are far lower than at Nextra.

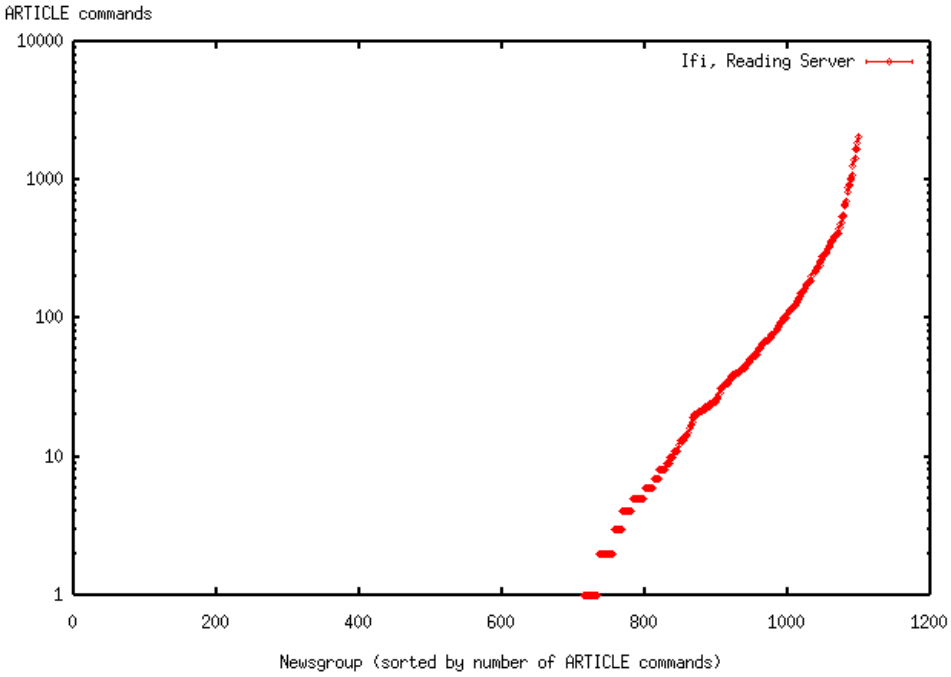

Figure 5.6: Ifi: ARTICLE commands per group
(Groups are numbered from 1 to 1 099.)

Groups	Sum	Min	Avg	Max	SD
1 099	8240	0	7.50	928	42.30

Table 5.14: Ifi: Statistics on XOVER commands per group

Groups	Sum	Min	Avg	Max	SD
1 099	50 927	0	46.34	2 044	177.89

Table 5.15: Ifi: Statistics on ARTICLE commands per group

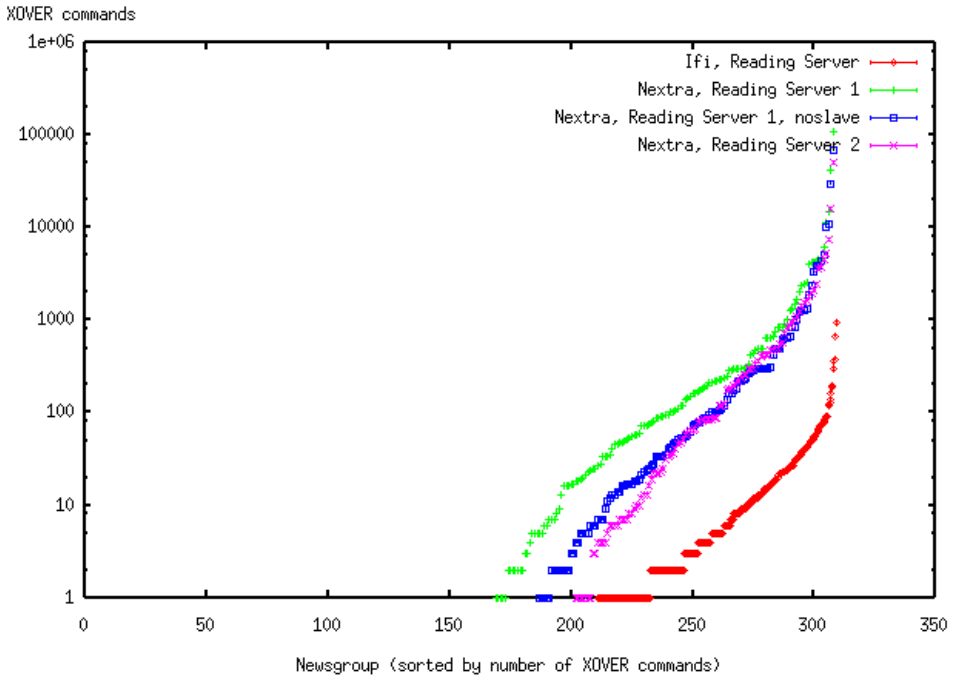

Figure 5.7: Ifi vs Nextra: XOVER commands per group
(Groups are numbered from 1 to 1 099 for Ifi, 1 to 309 for Nextra, but the graph is normalized to 309 groups.)

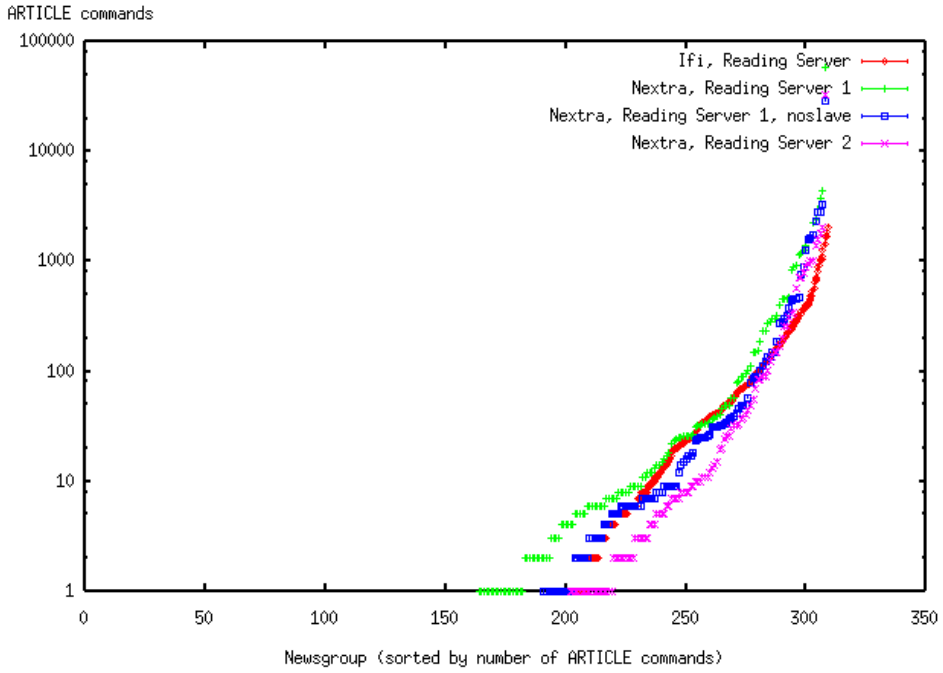

Figure 5.8: Ifi vs Nextra: xOVER commands per group
(Groups are numbered from 1 to 1 099 for Ifi, 1 to 309 for Nextra, but the graph is normalized to 309 groups.)

Groups	Sum	Min	Avg	Max	SD
426	8 240	0	19.34	928	66.23

Table 5.16: Ifi: Statistics on XOVER commands per group with both ARTICLE and XOVER commands

Groups	Sum	Min	Avg	Max	SD
426	50 927	0	119.55	2 044	269.98

Table 5.17: Ifi: Statistics on ARTICLE commands per group with both ARTICLE and XOVER commands

5.2.4 Preliminary Evaluation

Compared to Nextra, Ifi has a very high number of ARTICLE commands relative to the number of XOVER commands. In the previous section, I mentioned kill or score filters as a plausible explanation for part of this pattern, and that Nextra's users may be more heterogenous than those at Ifi. These uncertainties makes it unwise to jump to any conclusion about the differences between Ifi and Nextra.

Tables 5.16 and 5.17 show the statistical data for groups if we ignore those groups with zero ARTICLE and XOVER commands. We see then that the disproportion between Ifi and Nextra in XOVER commands remain, and it does not change the notion that there are important differences in reading patterns.

5.2.5 Data from Ifi Only

Table 5.18 on the facing page shows the number of unique, different headers from articles or article headers read in these sessions, plus the count of actual threads, as measured by the content of References headers and unique Subject headers. A From header's uniqueness is measured without regard to case of characters or whitespace. This also goes for a Subject header, where the string "Re:" is disregarded if it begins the subject. At the bottom of the table are subjects at the start of a thread (as defined by the References header) and Subject headers that are not a continuation of another Subject header. An example of the latter is shown in section 4.2.3 on page 52.

The Date header is a bit tricky, because the date format used differs slightly from article to article.[1] I have therefore not been able to reduce the detail of the Date header from including seconds.

The third and fourth columns are the number of unique headers per session and the number of read events per unique header.

[1] A curious thing: some even use two-digit numbers for the year, only one to one and a half year after the noise about Y2K problems.

Header	Count	Per Session	Read events/header
From	16 738	7.8	51.1
Subject	23 362	10.9	36.6
Newsgroups	1 548	0.7	552.8
Date	87 605	40.9	9.8
Message-ID	88 829	41.5	9.6
Followup-To	286	0.1	2992.2
References threads	17 378	8.1	49.2
Threads (reverse References)	61 155	28.5	14.0
Subject starting threads	16 473	7.7	51.9
Subject, not continuation	17 074	8.0	50.1

Table 5.18: Ifi: Unique header count

Sum	Min	Avg	Max	SD
856 091	1	9.60	233	10.37

Table 5.19: Ifi: Statistics on all read events per unique article having a read event

Per session, there were about 399 read events, and 41 unique articles requested.

The "Threads" number in table 5.18 tells us that there were a total of 61 155 articles that were referenced in other articles requested in the time period. These articles being referenced are not necessarily among the same articles as those that were read. 43 745 of those that were in the threads database were among the requested articles, which leaves 45 084 articles that do not belong to any thread.

The number of unique groups listed in the Newsgroups header of articles fetched because of a read event (1 548) is about 14% of the average of available groups (11 257). However, only 10% (1 099) of the available groups were actually entered. There were read events for 88 829 unique articles, but that is only about 2% of the total 3.95 million available articles.

On average, each unique article that is read has nearly 10 read events. Table 5.19 shows that the spread of read events per article is fairly large. We also see that the users on average enter a group for each 553nd read event, or each 57th unique article.

Table 5.20 on the following page show that there are significant variations on the number of ARTICLE, BODY, HEAD, GROUP, XHDR, and XOVER commands issued per unique group.

A significant difference from Nextra is that it was possible to check the date an article was posted. Tables 5.21 on the next page and 5.22 on page 77

Command	Sum	Min	Avg	Max	SD
ARTICLE	50 927	0	46.34	2 044	177.89
BODY	458 455	0	417.16	224 606	7 569.87
HEAD	460 521	0	419.04	224 551	7 568.62
GROUP	41 474	0	37.74	684	108.15
XHDR	6 008	0	5.47	832	36.88
XOVER	8 240	0	7.50	928	42.30

Table 5.20: Ifi: Statistics on ARTICLE, BODY, GROUP, XHDR and XOVER commands per unique group

Header	Count	Per session	Read events/header
From	9 618	7.9	82.7
Subject	14 068	11.6	56.5
Newsgroups	1 068	0.9	744.6
Date	58 280	48.1	13.6
Message-ID	58 799	48.6	13.5
Followup-To	202	0.2	3936.7
References threads	9 763	8.1	81.5
Threads (reverse References)	31 537	26.0	25.2
Subject starting threads	9 323	7.7	85.3
Subject, not continuation	11079	9.1	71.8

Table 5.21: Ifi: Unique header count, articles from before 2001-07-07

Header	Count	Per session	Requests/header
From	9 058	5.5	6.7
Subject	10 732	6.5	5.6
Newsgroups	1 089	0.7	55.6
Date	29 325	17.7	2.1
Message-ID	30 030	18.1	2.0
Followup-To	140	0.1	432.5
References threads	7 615	4.6	8.0
Threads (reverse References)	12 208	7.4	4.96
Subject starting threads	7 232	4.4	8.4
Subject, not continuation	8 951	5.4	6.8

Table 5.22: Ifi: Unique header count, articles from 2001-07-07 to 2001-07-12

show the stats split between articles posted before 2001-07-07 and articles posted at and after 2001-07-07, respectively. Note that about two thirds of the unique articles requested had a posting date of before 2001-07-07.

There were 1 211 sessions generating 795 217 read events for articles older than 2001-07-07, and 1 656 sessions generating 60 555 read events for articles from the period 2001-07-07 to 2001-07-12. In other terms, there were read events for 13 times as many older articles as for newer articles.

For a better view of how the read events are distributed per unique article, I have prepared statistics for three time periods: Before 2001-07-01, 2001-07-01 to 2001-07-06, and 2001-07-07 to 2001-07-12. The middle period is extracted from the older articles, because I want to know something about articles from a similar time period to the one I measured read events for.

Figure 5.9 on the next page illustrates the distribution of read events per article. Only 86 453 articles were possible to sort in this way; the rest had corrupted date field after parsing. The figure has been normalized for the newer articles, a count of 30 030 (35%). Older articles total 48 482 (56%), and articles from the middle period total 7 941 (9%).

Among older articles, the average number of read events is definitely higher than for others, but there are large "plateaus" of articles with exactly the same number of read events. 24 621[2] articles had exactly 12 read events each, and nearly 13 330 had exactly 30 read events each. Over 80% of the older articles are read ten times or more.

Similarly, 1 610[3] articles from the middle time period had exactly 12 read events, 4 684 had exactly one read event, and 934[4] articles had two read events each. However, only about 25% of the articles are read ten times or

[2]The graph makes it seem like less than 20 000 because of normalization.

[3]Looks like more than 5 000 because of normalization.

[4]Masked by read events for the newer articles in the graph.

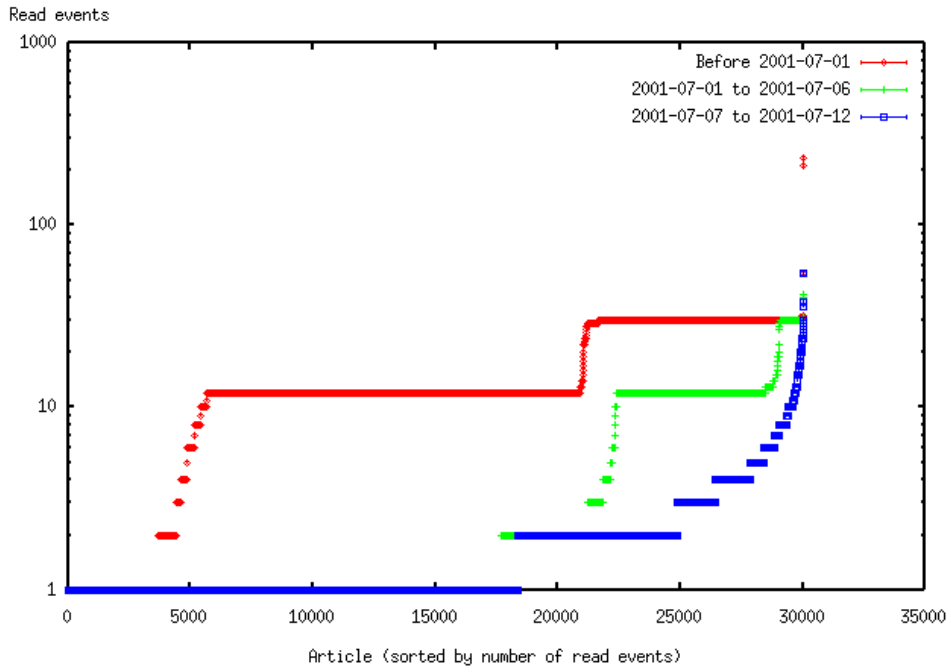

Figure 5.9: Ifi: Statistics on read events per unique article having a read event Articles from before 2001-07-01 are numbered from 1 to 48482, from 2001-07-01 to 2001-07-06 are numbered from 1 to 7941, from 2001-07-07 to 2001-07-12 are numbered from 1 to 30030. The graph has been normalized to 30030 unique articles.

Date range	Count	%	Min	Avg	Max	SD
Before 2001-07-01	754 180	88%	1	15.56	233	10.43
2001-07-01 to 2001-07-06	37 066	4%	1	4.67	42	6.55
2001-07-07 to 2001-07-12	60 555	7%	1	2.02	54	2.39
Total	856 091	100%	1	9.60	233	10.37

Table 5.23: Ifi: Statistics on read events per unique article, three time periods
The total includes 4 290 read events for 2 376 articles that did not have a valid date.

more, while those with one read event each make up 50% of the articles in that time period.

For newer articles, there were 18 352 articles with exactly 1 read event each, and more than 6 516 articles with exactly two read events each. These articles account for approximately 60% and 20% of the total of new articles.

Table 5.23 shows the specific numbers of read events with minimum, average, maximum and deviation for each time period and the total. This shows clearly the greater spread between minimum and maximum article counts for older articles, as well as that most of these articles are read more often than the more recent articles from 2001-07-01 and later.

What is not shown is that a number of the older articles were dated from 1997 to 2000. These articles total 10 490, or nearly 12%. Another curious point is that 449 157 of the read events was for a single group.

5.3 Discussion

The coarse comparison of Nextra with Ifi in section 5.2.3 on page 68 shows that there are no read events for a very large portion of the groups that also see XOVER commands. This formed an initial suspicion that a majority of the groups may not be read at all within the time of measurement, and indicates that group caching may help significantly. The more detailed numbers from Ifi seem to confirm that suspicion, since the read events are for only 14% (1 548) of the available 11 257 groups according to the Newsgroups header, or only 10% (1 099) groups according to the use of the GROUP command.

The data from Ifi was collected during midsummer break, which is a period with low usage. I cannot assume that this is directly transferable to medium or high usage periods, but the data are valid nonetheless; low usage periods can be expected for most reading servers, and I consider it a good thing if they adapt the available groups accordingly with group caching.

It is a problem that group caching may lead to a great number of extra, unread articles per group compared to other strategies. This causes unnecessary bandwidth usage when fetching the articles in a group, and does not preserve storage either. In addition, fetching of a new group can be initiated

very often. Table 5.9 on page 67 shows that there are more than 41 thousand group changes, while there were far fewer groups than that available. If group fetching is initiated often and people read more groups at other times of year than during summer, the server is likely to use a lot of bandwidth at a time for fetching various newsgroups. This can adversely effect user perceived performance, but my data does not support a more thorough examination of this point.

Considering that there were 3.95 million available articles in Ifi's news spool, and a total of 1.26 million incoming articles, a number of 88 829 unique articles requested is very small (section 5.2.2 on page 67). This is also a possible effect of vacation time. However, that there are only 2% read articles, and that these are in 10% of the groups should be a clear indication that caching entire groups is far from as effective as more fine grained methods.

Figure 5.9 on page 78 visualizes the distribution of unique articles with Ifi's number of associated read events for the three time periods. This distribution is interesting. For two large ranges of articles totalling nearly 80% of the span, old articles seem to have almost exactly the same amount of read events associated with them. Apart from these two ranges, the number of read events varies a lot. More recent articles have in general fewer read events. Since the old articles read are about 56% of the read articles, it is pretty clear that Ifi's users want to read older articles. But still, a full 35% of the articles were from the week I collected data for, so I think it is fair to say that new articles are interesting as well.

This pattern seems to be good for single article caching combined with a long expire time. From what I can see from these data, single article caching could have a cache hit rate of nearly 90% for articles over all, which is very good. However, if we disregard articles older than 2001-07-01, the hit rate drops, because 61% of the newer articles and 59% of those from the middle period are read only once. For Ifi this summer, a short expiry time would have been a disadvantage in hit rate and therefore user perceived performance. An advantage is that single articles will not be fetched together in the same way that a group fetch would, meaning that bandwidth usage would be less intense. It is possible that this large number of older articles being read is a side effect of vacation time and people catching up with old, unread articles. If this is the case, a short expiry time may not be disadvantageous to hit rate after all.

I mentioned thread based caching in connection with table 5.18 on page 75. The penalty in form of additional articles fetched is not very high; for the articles with read events at Ifi, only 20% extra articles would be fetched on a whole. Compared to group caching, this seems like a small gain on its own, though. Also, fewer articles will be fetched at a time, which should improve on bandwidth usage and response time to the user. Compared to single article caching, the differences in response time and bandwidth usage should be negligible, but disk usage will be greater.

Caching by similar subject is also worth considering in terms of hit rate. There are about as many different subjects as there are different threads, if one does substring matching. The possible problem for subject caching is that it requires additional processing power, while the competition from single article caching, thread caching, and group caching do not. This has an adverse effect on user perceived performance, perhaps to a stronger degree than group caching, since the server will have to compare subjects for every single cache miss. The comparative scarcity of different article authors, however, is probably a bit better, if the numbers from Ifi are anything to go by, but comparison of text strings must be done anyway.

Time based caching for these articles seems futile; as mentioned at the end of section 5.2.2 on page 67 near table 5.23 on page 79, almost 12% of the articles that were read were from between 1997 and 2000. The newer articles are unfortunately not so nicely distributed, and the data here is not good enough to tell what alternative caching strategies would help. I think this kind of filtering is harmful to the principle of caching as little as possible.

However, the NSP can choose to introduce penalties for requests for older articles. One such penalty can be that articles older than a certain point in time, for instance one month back, will never be cached, only fetched from an upstream peer if available. This means that there is a real risk that such an article will not be available at all, and if it is, it will take a longer time to get it for the proxy reading server. Another way is to charge customers per read event they cause in the proxy or per byte downloaded[5], so that those who use filtering methods causing read events pay more, perhaps enough or even more than enough to finance the extra storage or bandwidth needed to ensure this availability.

[5]NSPs like Giganews and NewsReader.com already charge for the capacity their users want

Chapter 6

Conclusion

I have presented the history of Usenet from a growth perspective, and shown that there are technical problems with the its continued growth. Smaller sites cannot afford to offer their users all the newsgroups they might want to read, and the problem seems to be growing.

While other solutions than caching — such as filtering — greatly reduce the size of a full newsfeed, they are rigid and do not adapt the incoming flow depending on usage, as caching will.

The world wide web has used various caching methods for years, and a lot of work and research has been done to optimize caching for the web. However, nobody has worked with solutions for news.

My proposed advanced caching methods for Usenet will help the smaller sites to appear to offer a greater amount of newsgroups and articles, but does not address the problem of the seemingly exponential growth. However, even a linear reduction in newsfeed size will buy the news administrators time to postpone the next hardware upgrade, which means they will save money. One small weakness is that I do nothing to help the backbone Usenet sites, which are the ones who carry the bulk of Usenet traffic today.

The following section of this chapter will — as far as possible — give answers to the research questions posed in section 1.6. The last section looks at the unresolved issues and proposes future research and work for improvement of Usenet.

6.1 Answers to Research Questions

In section 1.6, I asked a few research questions. Unfortunately, the data I have collected is not good enough that I can draw any definitive conclusions about what works best in which situation. Below, I present the answers I can give based on the discussion in section 5.3 of findings and results.

6.1.1 Which strategy or strategies are better for bandwidth?

As mentioned at the end of section 1.6, bandwidth benefits most from having no cache misses. While no cache misses is unrealistic if we are going to have a caching proxy, there are a few strategies that should present better cache hit rates than others.

According to my discussion in chapter 4, these are: Single article caching, protocol command caching, and thread caching. My findings in chapter 5 show that this may be correct, but the data does not support any definite conclusion on the subject.

6.1.2 Which strategy or strategies are better for disk usage?

Not having anything stored on disk is realistic in the case of proxies, as opposed to not having cache misses. Strategies that allow all or most data to be cached in main memory should be better here.

The ones I claimed to be better in chapter 4 were: Single article caching, protocol command caching, and thread caching. Disk usage seemed to be slightly higher for thread caching than the other three. However, this depends highly on usage patterns, and the data I collected is not of high enough quality to back up a solid conclusion on the subject.

6.1.3 Which strategy or strategies are better for user perceived performance?

In section 1.6, I made the obvious claim that users would see the best performance if articles were already stored locally when requested, and if system resource usage was low.

The strategies in chapter 4 I thought would give the best performance were, ranked from highest to lowest: Statistical caching, prefetching groups or hierarchies, group caching, and thread caching.

My findings were that this is very dependent on expiry time for the cache, and probably usage patterns as well. Group caching seemed to be a better basis than single article caching and thread caching, but again the data does not support a definitive conclusion.

6.1.4 Will caching proxies (as suggested) be a general improvement to Usenet ?

Throughout my presentation of the problems, as well as my discussion in chapter 4, I claimed that it would. While the findings do not support conclusions about which caching strategies are better, there can be no doubt that caching would be a general improvement for Usenet, as evidenced by the amount of unread groups and articles at Nextra and Ifi, even after filtering in the case of Nextra.

Considering that it may be possible to gain up to 50% hit rate from simple article caching alone at Ifi, and the claims from DNEWS (section 3.2.1) of 90% less bandwidth usage for binary groups, caching is not unrealistic for providing a reading service, but it remains unproven whether it may work for other kinds of news servers. It is also impossible to say how well caching will work, based on the data I have presented.

As for the availability of data when there is a cache miss, my test data cannot tell me directly which strategy would be better, of those I mentioned in section 4.4. I think highly of the reverse path lookup strategy (section 4.4.3), but there are indications in figure 5.9 on page 78 that for reading patterns similar to the ones displayed at Ifi, a cache that expires articles after just one or two weeks will have to fetch these articles again later. For these kinds of sites, it would perhaps be better if older articles were available from injecting server repositories or central group/hierarchy repositories, rather than asking the regular upstream peer to provide them.

6.2 Future Work

As a result of the work with this thesis, I have learned that there is more work that can be done with Usenet, both in practice and in an academic setting. I also have some ideas on how to build on the work of this thesis, and perhaps come to better conclusion based on better data.

To build a more complete model of how to decide what caching strategies are best, a news usage survey would help. If this was combined with more solid statistical software packages for existing news servers than the minor patches I wrote for INN, I think it should be possible to examine usage patterns closely enough to determine which caching strategies would work best. I would also like to see tests and evaluations of how the different strategies can be combined for optimum performance. The methods I have been using for evaluating the strategies suggested can be developed further for use in practical tests of thread based caching and statistical caching. A real life implementation of the best strategies would be a natural next step.

It would also be interesting to look at the entire problem from a different perspective. In the field of distributed operating systems, there is a term called *Distributed Shared Memory*. Briefly explained, this means that what programs see as memory local to one computer in fact is distributed over several. This is similar to how the different article retrieval storage strategies mentioned at the end of section 1.5.2 would seem to a user, if working properly, but I have not looked at it from this angle. This shared memory approach could be combined with Curt Welch's idea of header-only feeds mentioned in section 3.3.4, where he proposes a new "Body" header that could e.g. contain an URI (Universal Resource Identifier, defined in RFC 1630) pointing to one or several places where the body of the article can be fetched from.

Bibliography

[Assange et al., 2001] Assange, J., Bowker, L., and nntpcache crew, T. (2001). What is NNTPCache? http://www.nntpcache.org/about.html.

[Barber, 2000] Barber, S. (2000). RFC 2980: Common NNTP extensions. RFC.

[Barber, 2001] Barber, S. (2001). Network news transport protocol. Internet Draft.

[Bumgarner, 1995] Bumgarner, L. S. (1995). USENET — The Great Renaming — 1985–1988. http://www.vrx.net/usenet/history/rename.html.

[Cidera Inc., 2001] Cidera Inc. (2001). Cidera usenet news service. http://www.cidera.com/services/usenet_news/index.shtml.

[Collyer, 1992] Collyer, G. (1992). newsoverview - netnews overview files. newsoverview(5) man page.

[Crocker, 1982] Crocker, D. H. (1982). RFC 822: Standard for the Format of ARPA Internet Text Messages. RFC.

[Danzig, 1998] Danzig, P. (1998). Netcache architecture and deployment. *Computer Networks and ISDN Systems*, 30:2081–2091.

[Delany and Herbert, 1993] Delany, M. and Herbert, A. (1993). gup - a group update program. ftp.mira.net.au:/unix/news/gup-0.4.tar.gz.

[Freed and Borenstein, 1996a] Freed, N. and Borenstein, N. (1996a). RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC.

[Freed and Borenstein, 1996b] Freed, N. and Borenstein, N. (1996b). RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. RFC.

[Freed and Borenstein, 1996c] Freed, N. and Borenstein, N. (1996c). RFC 2049: Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples. RFC.

[Freed et al., 1996] Freed, N., Klensin, J., and Postel, J. (1996). RFC 2048: Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures. RFC.

[Freenix, 2001] Freenix (2001). Top 1000 Usenet sites. http://www.top1000.org.

[Hardy, 1993] Hardy, H. E. (1993). The Usenet System. *ITCA Yearbook*.

[Hauben and Hauben, 1995] Hauben, R. and Hauben, M. (1995). On the Early Days of Usenet: The Roots of the Cooperative Online Culture. http://www.columbia.edu/~rh120/ch106.x10.

[Horton and Adams, 1987] Horton, M. and Adams, R. (1987). RFC 1036: Standard for USENET Messages. RFC.

[Ingvoldstad, 1998] Ingvoldstad, J. (1998). Usenet news som plattform for effektiv gruppekommunikasjon. Essay written in Norwegian as a part of studies at Ifi.

[Inktomi Corporation, 2000] Inktomi Corporation (2000). NNTP Caching for Usenet Services. http://www.inktomi.com/products/network/traffic/tech/nntp/index.html.

[Kantor and Lapsley, 1986] Kantor, B. and Lapsley, P. (1986). RFC 977: Network News Transfer Protocol — A Proposed standard for the Stream-Based Transmission of News. RFC.

[Kondou, 2001] Kondou, K. (2001). Daily Usenet Article statistics on newsfeed.mesh.ad.jp. http://newsfeed.mesh.ad.jp/flow/.

[Konstan et al., 1997] Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Reidl, J. (1997). GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM*, 40(3):77–87.

[Krasel, 2001] Krasel, C. (2001). Leafnode, an NNTP server for small sites. http://www.leafnode.org.

[Krten, 1996] Krten, R. (1996). Improving Usenet news performance. *Dr Dobb's Journal*, 21(5):66, ++.

[Kurcewicz et al., 1998] Kurcewicz, M., Sylwestrzak, W., and Wierzbicki, A. (1998). A distributed WWW cache. *Computer Networks and ISDN Systems*, 30:2261–2267.

[Lindsey, 2001] Lindsey, C. H. (2001). News article format. Internet Draft.

[Moore, 1996] Moore, K. (1996). RFC 2047: Multipurpose Internet Mail Extensions (MIME) Part Three: Message Header Extensions for Non-ASCII Text. RFC.

[NetWin Ltd., 2001] NetWin Ltd. (2001). Dnews usenet news server software. http://netwinsite.com/dnews.htm.

[Nixon, 2000] Nixon, J. (2000). Newsfeed Size — How big is Usenet, anyway? http://newsfeed-east.supernews.com/feed-size/.

[Phifer, 2001] Phifer, L. (2001). Cidera's internet broadcast backbone. ISP Planet. http://www.isp-planet.com/technology/2001/cidera.html.

[Pratt and Bostian, 1986] Pratt, T. and Bostian, C. W. (1986). *Satellite Communications*. John Wiley & Sons.

[QNX, 2001] QNX (2001). QNX 4 Realtime OS. http://www.qnx.com/products/os/qnxrtos.html.

[Rose, 1993] Rose, M. T. (1993). *The Internet Message*. Prentice-Hall.

[Salz, 1991] Salz, R. (1991). Seeking beta-testers for a new NNTP transfer system. article in news.software.nntp. Message-ID: <3632@litchi.bbn.com>.

[Salzenberg et al., 1998] Salzenberg, C., Spafford, G., and Moraes, M. (1998). What is usenet? article in news.answers, http://www.faqs.org/faqs/usenet/what-is/part1/.

[Sato, 2001] Sato, Y. (2001). DeleGate Home Page. http://www.delegate.org/delegate/.

[Spafford, 1990] Spafford, G. (1990). Re: The List again :-). http://communication.ucsd.edu/bjones/Usenet.Hist/Nethist/0014.html.

[Spafford and Moraes, 1998] Spafford, G. and Moraes, M. (1998). Usenet software: History and sources. article in news.answers, http://www.faqs.org/faqs/usenet/software/part1/.

[Spencer, 1994] Spencer, H. (1994). "Son-of-RFC1036": News Article Format and Transmission. "Internet Draft-to-be".

[Spencer and Lawrence, 1998] Spencer, H. and Lawrence, D. (1998). *Managing Usenet*. O'Reilly & Associates. Out of print.

[Tanenbaum, 1996] Tanenbaum, A. S. (1996). *Computer Networks*. Prentice-Hall International, 3rd edition.

[Templeton, 1993] Templeton, B. (1993). The dynamic usenet feeding system. ftp.clarinet.com:/sources/dynafeed.tar.Z. Looking Glass Software Limited.

[Toigo, 2000] Toigo, J. W. (2000). Enterprise storage: Hitting the wall at 150 gb/in2. http://www.esj.com/fullarticle.asp?ID=102600101754AM.

[Udell, 1998] Udell, J. (1998). *Practical Internet Groupware*. O'Reilly & Associates. Out of print.

[Van Hees, 1993] Van Hees, K. (1993). VM NNTP: A TCP/IP Server Application For News. Computer Networks for Research in Europe, a supplement to Computer Networks and ISDN Systems.

[Welch, 1999a] Welch, C. (1999a). Header-only feeds. article in news.software.nntp. Message-ID: <19991216152219.196$Pi_-_@newsreader.com>.

[Welch, 1999b] Welch, C. (1999b). Re: Header-only feeds. article in news.software.nntp. Message-ID: <19991216223728.072$QK@newsreader.com>.

[Zakon, 2001] Zakon, R. H. (2001). Hobbes' internet timeline. http://www.zakon.org/robert/internet/timeline/. version 5.3.

Appendix A

Sources

This appendix contains source material for discussions in the thesis. This material is likely not to be available from other sources.

A.1 Changes to INN

Included below are the patches for significant changes in INN at Ifi.

The following data was generated from the original source files and the patched source files with this command:

```
diff -rub inn-work.orig inn-work.jani
```

```
diff −rub /local/src/news/inn/inn−work/include/config.h.in inn−work/include/conf...⇒
...ig.h.in
−−− /local/src/news/inn/inn−work/include/config.h.in Tue Oct 27 05:24:54 1998
+++ inn−work/include/config.h.in Sat Jul 7 00:04:39 2001
@@ −6,6 +6,14 @@

 #ifndef __CONFIG_H__
 #define __CONFIG_H__
+
+/* Define that you are Jan or Lars, for thesis work */
+#define JAN_LARS
+
+#ifdef JAN_LARS
+/* Extra logging information for thesis work */
+#define L_JAN_LARS LOG_DEBUG
+#endif

 /* Change to fork() if your vfork() is broken */
 #define FORK() vfork()
```

```
diff -rub /local/src/news/inn/inn-work/nnrpd/Makefile inn-work/nnrpd/Makefile
--- /local/src/news/inn/inn-work/nnrpd/Makefile Tue Oct 27 05:24:58 1998
+++ inn-work/nnrpd/Makefile Sat Jul 7 00:51:17 2001
@@ -14,7 +14,8 @@
        article.o group.o commands.o misc.o newnews.o \
        perl.o post.o loadave.o track.o udp.o

-ALL     = nnrpd actived
+# ALL   = nnrpd actived
+ALL     = nnrpd

 all:           $(ALL)

diff -rub /local/src/news/inn/inn-work/nnrpd/article.c inn-work/nnrpd/article.c
--- /local/src/news/inn/inn-work/nnrpd/article.c Tue Oct 27 05:24:59 1998
+++ inn-work/nnrpd/article.c Sat Jul 7 01:33:18 2001
@@ -834,6 +834,9 @@
     ARTNUM          art;
     char            *msgid;
     ARTNUM          tart;
+#ifdef JAN_LARS
+    char            *jlheader;
+#endif /* JAN_LARS */

     /* Find what to send; get permissions. */
     ok = PERMcanread;
@@ -922,6 +925,53 @@
         return;
     }
     Reply("%d %s %.512s %s\r\n", what->ReplyCode, buff, msgid, what->Item);
+#ifdef JAN_LARS
+    /* Setup timing to see how much extra time this takes */
+    JLSysLog(1, "Header dump");
+
+    if (msgid != NULL) {
+        JLSysLog(0, "Message-ID: %s", msgid);
+    } else {
+        JLSysLog(0, "Message-ID not found");
+    }
+    /* QIOrewind(ARTqp); */
+    if ((jlheader = GetHeader("From", FALSE)) != NULL) {
+        JLSysLog(0, "From: %s", jlheader);
+    } else {
+        JLSysLog(0, "From not found");
+    }
+    /* QIOrewind(ARTqp); */
+    if ((jlheader = GetHeader("Subject", FALSE)) != NULL) {
+        JLSysLog(0, "Subject: %s", jlheader);
+    } else {
```

```
+         JLSysLog(0, "Subject not found");
+     }
+     /* QIOrewind(ARTqp); */
+     if ((jlheader = GetHeader("References", FALSE)) != NULL) {
+         JLSysLog(0, "References: %s", jlheader);
+     } else {
+         JLSysLog(0, "References not found");
+     }
+     /* QIOrewind(ARTqp); */
+     if ((jlheader = GetHeader("Date", FALSE)) != NULL) {
+         JLSysLog(0, "Date: %s", jlheader);
+     } else {
+         JLSysLog(0, "Date not found");
+     }
+     /* QIOrewind(ARTqp); */
+     if ((jlheader = GetHeader("Newsgroups", FALSE)) != NULL) {
+         JLSysLog(0, "Newsgroups: %s", jlheader);
+     } else {
+         JLSysLog(0, "Newsgroups not found");
+     }
+     /* QIOrewind(ARTqp); */
+     if ((jlheader = GetHeader("Followup-To", FALSE)) != NULL) {
+         JLSysLog(0, "Followup-To: %s", jlheader);
+     } else {
+         JLSysLog(0, "Followup-To not found");
+     }
+     JLSysLog(1, "End of header dump");
+#endif /* JAN_LARS */
      if (what->Type != STstat) {
          if (ARTmem)
              ARTsendmmap(what->Type);
diff -rub /local/src/news/inn/inn-work/nnrpd/nnrpd.c inn-work/nnrpd/nnrpd.c
--- /local/src/news/inn/inn-work/nnrpd/nnrpd.c          Tue Oct 27 05:25:01 1998
+++ inn-work/nnrpd/nnrpd.c              Sat Jul  7 01:51:02 2001
@@ -614,6 +614,22 @@
 }


+#ifdef JAN_LARS
+#if defined(STDC_HEADERS) || defined(HAVE_STDARG_H)
+# include <stdarg.h>
+# define JL_VA_PARAM(type1, param1, type2, param2) (type1 param1, type2 par...⇒
...am2, ...)
+# define VA_PARAM(type, param)  (type param, ...)
+# define VA_START(param)            (va_start(args, param))
+#else
+# ifdef HAVE_VARARGS_H
+#  include <varargs.h>
+#  define JL_VA_PARAM(type1, param1, type2, param2)(type1 param1, param2, v...⇒
```

```
...a_alist) type2 param2; va_dcl
+#   define VA_PARAM(type, param) (param, va_alist) type param; va_dcl
+#   define VA_START(param)             (va_start(args))
+# endif
+#endif
+#endif /* JAN_LARS */
+
 #if       !defined(VAR_NONE)

 #if       !defined(VAR_NONE)
@@ -785,6 +801,11 @@
       struct group              *grp;
       GID_T                     shadowgid;
 #endif /* HAVE_GETSPNAM */
+#ifdef JAN_LARS
+      time_t                    JLlogtime;
+      char                      JLlogname[SMBUF];
+      char                      JLb;
+#endif /* JAN_LARS */

 #if       !defined(HPUX)
       /* Save start and extent of argv for TITLEset. */
@@ -1073,6 +1094,14 @@
       /* Exponential posting backoff */
       (void)InitBackoffConstants();

+#ifdef JAN_LARS
+      sprintf(JLlogname,"/local/lib/news/log/jani/news.debug_jani.%d.log",...⇒
time(NULL));
+      if((JLlogfile=fopen(JLlogname,"a"))==NULL) {
+          syslog(L_NOTICE,"Error opening jani's debug log '%s'",JLlogname);
+      }
+      JLSysLog(0, "Starting");
+#endif /* JAN_LARS */
+
       /* Main dispatch loop. */
       for (timeout = INITIAL_TIMEOUT, av = NULL; ;
                                 timeout = innconf->clienttimeout) {
@@ -1153,11 +1182,104 @@
                 continue;
             }
             TITLEset(av[0]);
+#ifdef JAN_LARS
+            /* Log exactly what command the user made */
+            JLb = buff[11];
+            buff[11] = '\0';
+            if (strcmp(buff, "LIST ACTIVE") != 0) {
+                buff[11] = JLb;
+                JLSysLog(0, "%s", buff);
```

```
+           } else {
+                buff[11] = JLb;
+           }
+#endif /* JAN_LARS */
            (*cp->Function)(ac, av);
        }

        Reply("%s\r\n", NNTP_GOODBYE_ACK);
+#ifdef JAN_LARS
+       JLSysLog(0, "Stopping");
+       if(JLlogfile!=NULL) {
+           fclose(JLlogfile);
+       }
+#endif /* JAN_LARS */

        ExitWithStats(0);
        /* NOTREACHED */
    }
+
+#ifdef JAN_LARS
+void
+JLSysLog JL_VA_PARAM(int, timestamp, const char *, fmt) {
+       va_list                 args;
+       char                    tmpfmt[SMBUF];
+       char                    *logmess;
+       static TIMEINFO         Session;
+       TIMEINFO                Now;
+       int                             size;
+       int                             tmp;
+
+       VA_START (fmt);
+       /* We're only interested in getting the session time at the first invocatio...⇒
...n. */
+       if (!Session.time) {
+           if (GetTimeInfo(&Session) < 0) {
+                syslog(L_FATAL, "can't gettimeinfo %m");
+                exit(1);
+           }
+       }
+       if (timestamp) {
+           if (GetTimeInfo(&Now) < 0) {
+                syslog(L_FATAL, "can't gettimeinfo %m");
+                exit(1);
+           }
+           sprintf(tmpfmt, "[JLsessID %d.%06d, %d.%06d] %s", Session.time, Session....⇒
...usec, Now.time, Now.usec, fmt);
+       } else {
+           sprintf(tmpfmt, "[JLsessID %d.%06d] %s", Session.time, Session.usec, fmt...⇒
...);
```

```
+       }
+       va_end (args);
+
+       /* Assume that most output is shorter than SMBUF, and take the
+          penalty for e.g. longer headers like References */
+       size = SMBUF;
+       if ((logmess = malloc (size)) == NULL) {
+           /* Ick.  Do the INN thing for malloc failure.  For now, just
+              print a debug message and return and pretend that nothing
+              serious is wrong. */
+           syslog(L_JAN_LARS, tmpfmt,
+                  "Failed to allocate memory for log message.");
+           return;
+       }
+       while (1) {
+           VA_START(tmpfmt);
+           tmp = vsnprintf(logmess, size, tmpfmt, args);
+           va_end(args);
+
+           if (tmp > -1 && tmp < size) {
+/*             printf("%s\n",logmess); */
+/*             syslog(L_JAN_LARS, logmess); */
+               if(JLlogfile!=NULL) {
+                   fprintf(JLlogfile,"%s\n",logmess);
+               }
+               return;
+           }
+           if (tmp > -1) {
+               /* If vsnprintf() returned the size of the string that
+                  would be written excluding the trailing '\0', which is
+                  correct according to C99, use that for the new size and
+                  save lots of time */
+               size = tmp + 1;
+           } else {
+               /* Bleh, old vsnprintf(), let's try to be quick about it
+                  anyway */
+               size *= 2;
+           }
+           if ((logmess = realloc (logmess, size)) == NULL) {
+               /* Ick (again).  Do the INN thing for memory alloc
+                  failure.  For now, just print and return. */
+               syslog(L_JAN_LARS, tmpfmt,
+                      "Failed to allocate memory for log message.");
+               return;
+           }
+       }
+}
+#endif /* JAN_LARS */
diff -rub /local/src/news/inn/inn-work/nnrpd/nnrpd.h inn-work/nnrpd/nnrpd.h
```

```
--- /local/src/news/inn/inn-work/nnrpd/nnrpd.h          Tue Oct 27 05:25:01 1998
+++ inn-work/nnrpd/nnrpd.h                  Sat Jul  7 01:28:59 2001
@@ -48,6 +48,10 @@
 #define Reply                   printf
 #endif  /* defined(VAR_NONE) */

+#ifdef JAN_LARS
+void                    JLSysLog(int timestamp, const char *fmt, ...);
+FILE                    *JLlogfile;
+#endif /* JAN_LARS */

 /*
 **  A group entry.
```

A.2 Man Page: newsoverview(5)

This is documentation for the format of news overview files.

```
NEWSOVERVIEW(5)                                        NEWSOVERVIEW(5)


NAME
     newsoverview - netnews overview files

SYNOPSIS
     /usr/spool/news/group/.overview

DESCRIPTION
     Each  newsgroup  directory  contains  a  file  named
     '.overview', containing one-line summaries of articles  in
     that group.  Fields are separated by tabs, and any tabs or
     newlines  in  the  original  articles  headers  have  been
     replaced  with  spaces.  The fields are, in order: article
     number (file name),  subject,  author,  date,  message-id,
     references,  byte  count, line count, and optionally other
     headers, as arranged locally (none  are  supplied  by  the
     database  maintenance  software,  as  shipped).  The line-
     count and references field may be empty.  If the  optional
     other  headers are present, they include their header key-
     word and colon; if they are absent entirely, the tab after
     the line-count field may also be absent.

     The file is maintained in numerical order, by article num-
     ber.

     At the time of  writing  (late  1992),  the  lines  in  an
     overview  file  are typically 150-300 characters long, and
     an overview file for  a  typically  busy  group  is  often
     30,000 bytes to 60,000 bytes long, with notable exceptions
     exceeding 500,000 bytes.

EXAMPLES
     A few lines from one overview file, with tabs displayed as
     '|' and lines continued after '\'.

       8870|strange message id's: <something>QUIT|schmitz@scd.hp.com (John Schmitz)\
       |18 Sep 1992 19:57:16 GMT|<19dcasINNce@hpscdf.scd.hp.com>||821|
       8871|Re: BNF rule for newsgroup names?|kris@tpki.toppoint.de (Kristian Koehntopp)\
       |Fri, 18 Sep 1992 17:51:09 GMT|<1992Sep18.175109.21999@tpki.toppoint.de>\
       |<43GQBM4D@cc.swarthmore.edu> <BuLB9y.7Dp@world.std.com>\
        <BuLG5A.xH@cs.psu.edu> <339@blars.UUCP>|926||Supersedes: <1234@foovax>

FILES
     /usr/spool/news/group/.overview

SEE ALSO
     newsdb(5)

HISTORY
     Written by Geoff Collyer as part of the C News project.

BUGS
     The  contents  of  the  line-count  field  should  not  be
     believed and are really pretty worthless yet popular.
```

A.3 Selected Newsgroups From Nextra

Below are the 309 groups that were selected from an alphabetical list of newsgroups available from Nextra's reading servers.

```
alt.0d
alt.adoption
alt.amateur-comp
alt.animals.dogs.collies.open-forum
alt.apocalypse
alt.arts.origami
alt.autos.audi
alt.autos.volvo
alt.bbs.first-class
alt.binaries.e-book.palm
alt.binaries.images.fun
alt.binaries.multimedia.utilities
alt.binaries.pictures.animals
alt.binaries.pictures.drag-racing
alt.binaries.pictures.erotica.d.moderated
alt.binaries.pictures.erotica.fetish.latex
alt.binaries.pictures.erotica.male.bodybuilder.moderated
alt.binaries.pictures.fishing
alt.binaries.pictures.movie-posters
alt.binaries.pictures.strippers
alt.binaries.radio-control
alt.binaries.sounds.midi.classical
alt.binaries.starwars
alt.books.david-weber
alt.building.architecture
alt.cars.lotus
alt.cellular.umts
alt.chinchilla
alt.collecting.pens-pencils
alt.comp.freeware.gdp
alt.comp.periphs.mainboard
alt.comp.periphs.videocards.nvidia
alt.consciousness.mysticism
alt.crime.bail-enforce
alt.current-affairs.muslims
alt.disability.blind.social
alt.drugs.ecstasy
alt.emergency.services.dispatcher
alt.ezines.rad
alt.fan.countries.greece
alt.fan.elton-john
alt.fan.jen-aniston
alt.fan.mailer-daemon
alt.fan.pornstar.darrian
alt.fan.starwars
alt.fashion.crossdressing
alt.flame.girlfriend
alt.food.wine
alt.fuckin.rowin.mate
alt.games.black+white
alt.games.diablo2
alt.games.fucking
alt.games.metal-gear-solid
alt.games.nintendo.pokemon
alt.games.rpg.ufa
alt.games.upcoming-3d
```

```
alt.games.video.xbox
alt.grad-student.tenured
alt.ham-radio.am
alt.help.businesscalc
alt.horology
alt.hvac
alt.inner.circle
alt.irc.efnet
alt.irc.webnet
alt.ketchup
alt.legend.king-arthur
alt.lycra.pictures
alt.marshmellow.peeps
alt.mens-rights
alt.motd
alt.movies.kubrick
alt.music.4-track
alt.music.blues
alt.music.deftones.moderated
alt.music.gogos
alt.music.lyrics
alt.music.nin
alt.music.portishead
alt.music.sex-pistols
alt.music.tool
alt.necronomicon
alt.norway.fishhead
alt.org.team-os2
alt.paranormal
alt.personals.intercultural
alt.php.sql
alt.politics.usa.constitution.gun-rights
alt.psst.hoy
alt.radio.pirate
alt.religio.konfuceo
alt.revisionism
alt.rv
alt.sci.time-travel
alt.sex.abstinance
alt.sex.cu-seeme
alt.sex.fetish.motorcycles
alt.sex.fetish.white-mommas
alt.sex.leri
alt.sex.prom
alt.sex.strip-clubs
alt.sex.weight-gain
alt.skinheads
alt.soft-sys.corel.draw
alt.sport.horse-racing.systems
alt.sports.hockey.nhl.phila-flyers
alt.stop.spamming
alt.support.crossdressing
alt.support.mcs
alt.swedish.chef.bork.bork.bork
alt.tasteless
alt.test
alt.transgendered
alt.tv.animaniacs.pinky-brain
alt.tv.commercials
alt.tv.frasier
alt.tv.law-and-order
alt.tv.networks.cbc
```

```
alt.tv.real-world
alt.tv.simpsons
alt.tv.tiny-toon
alt.usa-sucks
alt.video.letterbox
alt.yoga
aus.radio.scanner
ba.jobs
bit.listserv.techwr-l
bofh.config
bofh.jobfh.offered
bofh.sysops
borland.public.cppbuilder.activex
borland.public.datagateway
borland.public.jbuilder.announce
borland.public.jbuilder.jbcl
cern.delphi.bg
comp.ai.genetic
comp.archives.msdos.d
comp.data.administration
comp.databases.sybase
comp.editors
comp.graphics.algorithms
comp.graphics.packages.3dstudio
comp.infosystems.www.authoring.cgi
comp.lang.basic.powerbasic
comp.lang.forth
comp.lang.ml
comp.lang.perl.tk
comp.mail.headers
comp.os.cpm
comp.os.ms-windows.ce
comp.os.ms-windows.programmer.networks
comp.os.msdos.programmer
comp.os.os2.setup.storage
comp.protocols.dns.bind
comp.publish.prepress
comp.society
comp.std.c++
comp.sys.amiga.applications
comp.sys.apollo
comp.sys.cdc
comp.sys.ibm.pc.games.naval
comp.sys.intergraph
comp.sys.mac.hardware.storage
comp.sys.mentor
comp.sys.pen
comp.sys.sgi.bugs
comp.sys.unisys
comp.unix.amiga
comp.unix.tru64
cz.comp.linux.suse
de.alt.flame
de.alt.ufo
de.comp.sys.amiga.comm
de.rec.musik.rock+pop
dk.bolig
dk.edb.internet.software.mail+news
dk.edb.os2
dk.edb.system.beos
dk.fritid.dyr.hest
dk.helbred.behandling.alternativ
```

```
dk.marked.privat.bilstereo
dk.opslag.stillinger
dk.teknik.telefoni.isdn
england.jobs.offered
europa.viages.tourismo
fido7.kharkov.pickup
fido7.ru.anekdot
fido7.ru.sex
fido7.ua.os2.crack
fr.rec.plongee
free.autos.daihatsu
free.ebooks-2000
free.inter-vieri
free.it.auto.lancia-delta
free.it.radio.scanners
free.napster
free.ti.amici.nt.amatore
free.uk.music.classical
free.uk.talk.orkney-shetland
gnu.bash.bug
gnu.gdb.bug
hun.lists.hix.auto
hyssing.net.adsl
it.binari.x.hentai
linux.debian.devel
linux.dev.kernel
linux.postgres
linux.wine.users
microsoft.public.access.security
microsoft.public.biztalkserver.setup
microsoft.public.dotnet.csharp.general
microsoft.public.es.pocketpc
microsoft.public.exchange2000.setup.installation
microsoft.public.games.zone.asherons_call
microsoft.public.internetexplorer.win95
microsoft.public.nordic.ie40
microsoft.public.officedev
microsoft.public.project.vba
microsoft.public.sqlserver.datamining
microsoft.public.vb.bugs
microsoft.public.vb.winapi
microsoft.public.vstudio.general
microsoft.public.win2000.enable
microsoft.public.win2000.setup
microsoft.public.win95.msdosapps
microsoft.public.windows.inetexplorer.ie55.outlookexpress
microsoft.public.windowsme.software
microsoft.public.windowsnt.terminalserver.client
microsoft.public.xsl
misc.invest.technical
misc.wanted
muc.lists.freebsd.ports
net.config
net.sexuality.stories
netscape.public.mozilla.os2
news.groups
nl.naturisme
no.alt.frustrasjoner
no.alt.sjokolade
no.annonser.it.telekom
no.fag.medisin.diverse
no.fritid.jakt
```

```
no.it.maskinvare.diverse
no.it.os.unix.linux.nettverk
no.it.telekom.diverse
no.kultur.humor
no.samfunn.helse.funksjonshemming.diverse
no.sport.fotball
nordunet.announce
norge.oppland
novell.lanalyzerforwindows
novell.support.os.server.netware5x
online.xdsl
opera.magic
pl.comp.grafika.grafika3d
rec.arts.anime.creative
rec.arts.comics.reviews
rec.arts.movies.lists+surveys
rec.arts.sf.movies
rec.arts.theatre.musicals
rec.autos.makers.vw.watercooled
rec.aviation.ifr
rec.bicycles.misc
rec.collecting.stamps.discuss
rec.food.drink.beer
rec.games.chinese-chess
rec.games.frp.misc
rec.games.roguelike.moria
rec.games.video.sega
rec.mag.dargon
rec.music.artists.ani-difranco
rec.music.country.old-time
rec.music.makers.synth
rec.parks.theme
rec.photo.marketplace.digital
rec.radio.amateur.space
rec.sport.archery
rec.sport.volleyball
rec.video.desktop
redhat.kernel.general
sci.astro.amateur
sci.electronics.equipment
sci.environment.waste
sci.med.aids
sci.physics.relativity
se.dator.programmering.diverse
se.politik.forsvar
sfnet.harrastus.sukututkimus
soc.culture.asian.american
soc.culture.haiti
soc.culture.thai
soc.genealogy.marketplace
soc.history.science
soc.religion.shamanism
south-wales.misc
staroffice.com.support.stardesktop
swnet.fritid.jakt
swnet.pryltorg
talk.religion.buddhism
uk.comp.os.win2000
uk.games.video.playstation.forsale
uk.media.tv.sf.startrek
uk.railway
uk.rec.fishing.sea
```

```
uk.rec.subterranea
us.config
z-netz.alt.sat-tv
z-netz.rechner.atari.8-bit
z-netz.rechner.ibm.programmieren
```

A.4 Answers to Questions on news.software.nntp

The three sites where news administrators answered have been labeled "Site A", "Site B", and "Site C". "Site A" is a medium sized site, "Site B" is fairly large, while "Site C" is huge. The questions are included below as they were asked, but the answers have been anonymized and summarized. The questions were asked on **news.software.nntp** 2001-04-30, and the third answer came 2001-05-28, the day after I posted a summary of the first two to the group.

- **How many articles are you offered and how many do you accept per day, in average for the past month or so?**

  Sites A and B estimated the number of offered articles as three and eight times as many as the accepted articles. Site C answered that this depends on the number of feeds.

  Accepted articles were around 1.2 to 1.3 million per day for all three sites.

- **What is the total volume of the offered (that may be tricky :) and accepted articles in the same period?**

  None of the sites are able to measure the volume of articles that were not accepted. The volume of accepted articles per day was approximately 140 GB for Site A, 190 GB for Site B, and between 220 and 250 GB for Site C.

- **How many peers serve you, and how many do you serve?**

  Site A has 5 peers, Site B has approximately 30, and Site C answered that they have enough peers.

- **How many users do you serve?**

  Site A serves less than 2 000 users, Site B serves less than 10 000, and Site C does not know how many end users there are, because many of their clients are corporate accounts.

- **Does the size of a "full" feed (assume an excess of 300 GB/day, tripling every 13 months) pose a problem for your servers' and network's capacity?**

  Simply put, yes, at least from a business and cost perspective.

- **If so, do you have a plan to deal with it, and what would that be?**

  Sites A and B answered that they would be upgrading when economically viable, moving traffic from expensive commodity network connections to zero-cost links, growing the business. Site C did not answer.

- **What is the internal news network and server architecture?**

  Site A and B use a dedicated feeding server with standard NNTP newsfeeds. On the reader side, A used a traditional news spool, while B used cyclical spools. Site C has multiples of everything as well as custom software on the reading side.

- **Do you use any kind of load balancing, on any level, and if so, what kind?**

  The different sites varied wildly from no load balancing (Site A) to a fairly advanced setup (Site C) using automated switching within a server farm.

- **Do you try to carry "all" groups/hierarchies, or do you have some specific limitations to which groups/hierarchies you don't want, and which groups/hierarchies you want?**

  For the feeders, all try to allow everything in transit, but Site B has a limit on binaries to some peers because of bandwidth costs to those, and Site C weeds out unused hierarchies or hierarchies that are not propagated.

- **Is there a grey area of groups that you may or may not want, which really depends on what your users or downstream peers request?**

  This question was partially misunderstood, because of the way the question was phrased. One relevant answer was that legal issues may have a bearing on what is being carried. After a clarification on my side in a summary posted to the newsgroup, Site C responded that they had a partially automatic setup for creation and removal of newsgroups based on PGP signed control messages where possible, in hierarchies without formal procedures wholly automatically, and otherwise on a manual request basis.

- **Do you use a header-only feed, and if so, how do you provide the bodies to the readers?**

  Site A had the front end reading servers receive header-only, and read the articles directly from the news spool of the feeding server based on data from the headers. Site C did not answer the question.

- **If not, why aren't you using a header-only feed?**

  Site B answered that the network architecture was good enough that it would be pointless. Site C did not answer the question.

- **What do you consider the pros and cons of a header-only feed?**

  Site A thought there were no obvious cons the way it is implemented there, and that it forms a basis for a simple and robust distributed architecture, where the front end knows whether a given article has expired or not. Site C did not answer the question.

- **If your reader doesn't have an article available, and it is requested by a user agent, does it attempt to get it anyway, and if so, how? Does it try to fetch that specific article, or that article plus several others by References, newsgroup, or other grouping methods?**

  At Site A, the reading server always knows what it has and what it does not have. At Site B, the reading server will not attempt to get the article anyway. Site C did not answer the question.

- **Can your feeder or injector accept automated requests for specific articles, or grouped articles as mentioned above, for immediate feeding to a downstream peer?**

  Essentially, no, unless NNTP reader commands are used for specific articles. Site C did not answer the question.

- **Do you try to limit the traffic (including control messages) by means of filtering, and if so, what kind?**

  Site A limits article size between the feeding server and the reading server. Site B runs Cleanfeed in paranoia mode, no measurements made, plus anti-spam feature enabled in the feeder. Site C also performs extensive filtering on spam misplaced or excessive numbers of binaries, abusive cross-posting, etc.

- **How much does that filtering reduce accepted volume and amount of articles (including control messages)?**

  Site A sees a reduction in volume by 50% and article count by 2%. Site B did not answer the question. Site C sees a reduction in volume from 220-250 GB/day to 190-220 GB/day.

- **How do you store your articles? Cyclic spools, traditional one-file-per-article, combination, or other means?**

  Site A concatenates articles into large files with NNTP wire format. Site B uses a traditional news spool. Site C uses a custom designed cyclic news spool. This is all on the reading servers; the feeding servers generally try to avoid storing articles other than temporarily.

Appendix B

News Articles

The purpose of this appendix is to provide some news articles that may not be readily available in the future.

B.1 Rich Salz Announcing INN Testing

```
Path: papaya.bbn.com!rsalz
From: rsalz@bbn.com (Rich Salz)
Newsgroups: news.software.nntp,news.admin,comp.org.usenix
Subject: Seeking beta-testers for a new NNTP transfer system
Message-ID: <3632@litchi.bbn.com>
Date: 18 Jun 91 15:47:21 GMT
Followup-To: poster
Organization: Bolt, Beranek and Newman, Inc.
Lines: 72
Xref: papaya.bbn.com news.software.nntp:1550 news.admin:15565 comp.org.usenix:418

InterNetNews, or INN, is a news transport system.  The core part of the
package is a single long-running daemon that handles all incoming NNTP
connections.  It files the articles and arranges for them to be forwarded
to downstream sites.  Because it is long-running, it can be directed to
spawn other long-running processes, telling them exactly when an article
should be sent to a feed.  This can replace the "watch the logfile" mode
of nntplink, for example, with a much cleaner mechanism:  read the
batchfile on standard input.

InterNetNews assumes that memory is cheap and fast while disks are slow.
No temporary files are used while incoming articles are being received,
and once processed the entire article is written out using a single
writev(2) call (this includes updating the Path and Xref headers).  The
active file is kept in memory (a compile-time option can be set to use
mmap(2)), and the newsfeeds file is parsed once to build a complete matrix
of which sites receive which newsgroups.

InterNetNews uses many features of standard BSD sockets including
non-blocking I/O and Unix-domain stream and datagram sockets.  It is
highly doubtful that the official version will ever provide support for
TLI, DECNET, or other facilities.

INN is fast.  Not many hard numbers are available (that is one requirement
of being a beta-site), but some preliminary tests show it to be at least
twice as fast as the current standard NNTP/C News combination.  For
```

example, Jim Thompson at Sun has had 20 nntpxmits feeding into a 4/490, and was getting over 14 articles per second, with the CPU 11% utilized. I was getting 10 articles/second feeding into a DECstations 3100, with the program (running profiled!) 50% idle and the load average under .7. (It is a scary thing to see several articles filed with the same timestamp.)

The sys file format is somewhat different, and has been renamed. The arcane "foo.all" syntax is gone, replaced with a set of order-dependant shell patterns. For example, instead of "comp,comp.sys.sun,!comp.sys" you would write "comp.*,!comp.sys.*,comp.sys.sun"; to not get any groups related to binaries or pictures, you write "!*pictures*,!*binaries*".

There are other incompatibilities as well. For example, ihave/sendme control messages are not supported. Also the philosophy is that that invalid articles are dropped, rather than filed into "junk." (A log message is written with the reason, and also sent back to the upstream feed as part of the NNTP reject reply.) The active file is taken to be the definitive list of groups that an article wants to recieve, and if none of an article's newsgroups are mentioned in the active file, then the article is invalid, logged, and dropped.

The history and log files are intended to be compatible with those created by C News. I want to thank Henry and Geoff for their kind permission to use DBZ and SUBST. You will need to be running C News expire or a B2.11 expire that has been modified to use DBZ.

The InterNetNews daemon does not implement all NNTP commands. If sites within your campus are going to post or read news via NNTP, you will need the standard NNTP distribution. The daemon will spawn the standard nntpd if any site not mentioned in its "hosts.nntp" file connects to the TCP port. InterNetNews includes a replacement for the "mini-inews" that comes with the standard NNTP distribution. This can be used on any machine that posts news and connects to an NNTP server somewhere; its use is not limited to INN. At some point I hope to have a replacement nntpd optimized for newsreaders, and an NNTP transmission program. These will remove the need for any external software beyond the C News expire program.

If you would like to beta-test this version, please FTP the file pub/usenet/INN.BETA from cronus.bbn.com for directions. It will be a fairly tightly-screened beta: DO NOT ASK ME FOR COPIES! Once the system is stable, it will be freely redistributable. I hope to have the official release by August 7, so that schools can bring the system up before the semester starts.

/rich $alz

--

Please send comp.sources.unix-related mail to rsalz@uunet.uu.net.
Use a domain-based address or give alternate paths, or you may lose out.

B.2 Curt Welch on Limiting Feed Size

B.2.1 Header-Only Feeds

```
Path: nntp.uio.no!uio.no!news-spur1.maxwell.syr.edu!news.maxwell.syr.edu
        !newspeer1.nac.net!netnews.com!news-xfer.newsread.com!netaxs.com
        !newsread.com!feed.newsreader.com!news2.newsreader.com
        !flame-test.newsreader.com!not-for-mail
Subject: Header-Only feeds (was: A nice side benefit of WebReader)
From: curt@kcwc.com (Curt Welch)
Date: 16 Dec 1999 20:22:19 GMT
Organization: NewsReader.Com
Message-ID: <19991216152219.196$Pi_-_@newsreader.com>
Newsgroups: comp.lang.java.softwaretools,news.software.readers,
        alt.usenet.offline-reader,news.software.nntp
Followup-To: news.software.readers,news.software.nntp
References: <38569b7a$0$224@nntp1.ba.best.com>
        <6aJWOE2g9YI1qQ46VtGbCuiICWXm@4ax.com>
        <3856bc3d$0$229@nntp1.ba.best.com>
X-User: Curt@NewsReader.Com
X-Face: "p}G*1KH{+F7EYGKLb>ogDguabZ+%,?^epeFB!nzu‘)‘$=QcvL1KF6<0GH!Tbc!Sqo[|tV5
        %IW48mQf3K=Ci&gZ7]]aazx@]Y-nq!r5{yH/#,?@lDdUDvOfByB2hVW0.@OM%{l/{cT’{w
X-Url: http://CurtWelch.Com/
Lines: 213

uomini@fractals.com (Robert Uomini) wrote:
> Sorry about the munged message. Here it is in its entirety:
>
> One nice benefit of the transparent HTML technology, even if you
> don’t post HTML articles, is that it can potentially save significant
> disk space on news servers worldwide. Here’s how:
>
> When you post an article which makes use of the technology, only the
> NNTP header

The article header has nothing to do with NNTP.  You are confusing
two different standards.

> (including the URL of the article) is sent to Usenet. So,
> if a news server knows that all the clients it serves understand
> transparent HTML, there’s no need to send a synopsis of the article;
> the body of the article becomes redundant. This technique works even
> if the article consists of plain text, as long as it’s wrapped in the
> standard <HTML><BODY>...</BODY></HTML> tags.

No, you can’t wrap plan text in that and have it come out correctly.
You have to include <PRE> tags and do escape processing on
the <, > and & characters to make it display correctly.

But if it’s plain text the best way is to just make the server know
that and give it a type of text/plain instead of text/html and then
it works fine. (at least it works fine for web browsers, don’t know
if your webreader can support text/plain correctly).

> Think of the disk space which can be saved...

I wonder if this idea might actually be useful.  Not for the stupid
HTML support that webreader seems to tout, but just as the basis for
restructuring how Usenet works to deal with the bandwidth issue.  I’ve
been thinking about this as I’ve read this thread and thought it might
be worth while to discusse it.  So here are some of my thoughs on the
issue (mostly created as I wrote this).
```

I've added news.software.nntp to the newsgroups and set followup to n.s.n and n.s.r because I think to work, this has to be mostly done at the server level and not the newsreader level.

What I'm thinking about is creating a standard to allow header-only feeds to become the norm. I use this internally on my servers where my front-end servers get header-only feeds and are able to process all the NNTP commands locally but go to the back-end servers for the actual articles. And lots of other sites have done this with Diablo as well (I use my own custom servers instead of Diablo).

It's a great technique because a header-only server can keep up with a full feed with with only a small fraction of the CPU power, disk space, and bandwidth, yet it offloads a large percentage of the user load from the back-end servers.

So maybe it would make sense to try and transform Usenet so everyone was passing around header-only versions of the articles. The full version of the article would remain only on central servers, normally the server it was posted on.

I think the only protocol change that needs to be made is a new article header which would both indicate that the article was a "header-only" article, and tell where to go to get the article.

I'm thinking Body: would be a good candidate for this. I'm not sure what syntax would be best for the content of this header however. A URL could work, but it strikes me that you don't want to allow the full range of URLs. I'm thinking we would just use NNTP to get the articles from the remote server like the way DIABLO works. So all you need is a host name and an optional username/password and port. Maybe the URL syntax is the way to go but have a well defined standard for what's allowed and what's not allowed.

A header only feed at todays news volume levels is only around 1GB/day and 100Kbps of bandwidth. That's two orders of magnatude smaller than a full feed.

Only the largest ISPs seem to be able to run a news server these days and even some the largest seem to be cutting back their feeds because they can't justify the cost of a full feed. Just about all ISPs would be able to justify running their own header-only server.

So how does the net convert to this? I'm thinking it would work something like this. We do it all at the server level. We build servers that have the fetures needed to support the function. For any article posted localy, the server adds the Body: header and is willing to accept connections from anywhere for people to read those articles. You might even set up a seperate server to hold just the articles posted locally and the Body: headers would point everyone to that server (and of course your main server(s) would have to feed all local articles there as well).

These "new" servers would be able to accept both normal articles and header-only articles with Body: headers. When a user requests an article that the server only has the header for, the server makes a connection to the remote site and gets the article and sends it to the user. But it also caches the article for other users --- i.e., it acts like a caching server.

When feeding, you could configure these "new" servers do do a few different things. For compatibility with current servers, they must have

the option of doing full article feeds. They would send the entire article even it was posted localy. But they would still add Body: headers to local articles they were willing to serve to the world.

For downstream feeds that were both able and willing to take the header-only articles, only the header for the local articles would be passed. And if the server had articles with Body: headers, only the header of those articles would be feed, even if the entire article happened to be on the server.

The third feed option is a complete header-only feed. Any article which already had the Body: header would be feed as is. But articles without the Body: header would have one added and the feeding server would then be acting as the "body server" for all those articles. This of course would only be done on a limited basis because no one would be willing to server all of Usenet to the world for free.

And since the servers are able to deal with a mix of full articles and header only articles, there's always the options of controling what's feed as a full article and what's feed as header-only. For example, if you have the bandwidth, you can accept full articles for all the small articles, but get header-only copies of articles over a certain size.

It seems to me that any ISP willing to outsource their news might prefer to run a header-only feed from the outsourcer. It's a small simple box to run (compared to a normal news server), gives their users fast xover response, and acts as an article cache so articles read by their users only need to tranist their external links once. And since they are paying for the service, the outsourcer in this case will be willing to act as the master server for all articles without Body: headers. This is very similar to running one of the caching servers that already exist today, but then we have the option of making that local server act as the store house for the locally posted articles.

The problem with this of course is why would any usenet site be willing to act as an article server to the world for their local articles. Under the current system, when a user posts an article, you only have to send it out to your feeds once. But if you act as a server, you will most likely have to send it out a lot more. Maybe even thousands of times.

In this system, you are serving the article not to every user that reads it, but to every usenet server where at least one user reads it. And in theory, you would only send it to each of those servers at most once.

And some sites could set up caching servers to be used by groups of servers. Say for example an ISP that sells/provides news feeds could act as a central caching server for all their down-stream feeds. They could send out header only feeds to their down-stream customers where the Body: lines were all modified to point back to their server. When their server receives a request for the article, it once again sees that it's only got the header and then chains back to the orignal site.

People would be willing to set up their servers as "body-servers" to the world if overall it meant they would be able to reduce their bandwith usage. So the question is, would it reduce the overall volume if we all switched to this or not? I'm not sure.

But, a very good side effect of this system is that sites now become more responsiple for what there user's post. Not as much in terms of responsibility for the content, but more so in terms of responsibility of the bandwidth costs of distributing the post. Part of the problem with the current Usenet design is that costs don't get allocated correctly. People

pay to read news, not to post it. But posting is what costs the most money.

So why don't we charge users to post? Because it's not "our" money they spend when they do that. A single post costs hundreds of dollars in bandwidth usage to distribute throughout the world, but it costs the local site almost nothing. So we don't charge. So the user doesn't get charged based on what they really "cost" usenet.

But in this new header-only system, we pay very little for the incomming feed, and pay a lot for the outgoing articles posted by our users. So now we are motivated to control how much they post, or charge more based on the volume they post. And that's the way it should work, unlike what we have today. They people who post should carry a substantial portion of the cost instead of all the readers.

I'd think that retention time would be overall much better with this new system as well. Or overall, we would need a lot less disk space to get the same amount of retention. The retention for the headers would take 1/100 of what it takes now. And local posts generally take a lot less space so they could be retained for much longer periods.

If posting ends up costing the person doing the post more, overall volume should go down, and quality should go up. Spam should go down as well because it will become very expensive to bulk post hundreds or thousand articles.

At first, when both the old a new systems are still in use, spammers will obviously prefer to spam in the old system (i.e. send out full articles). But if this new system works, then that would just motivate the net to force a conversion to the "new" system.

One problem I see is that when you read news with this new system, you end up getting news from lots of different places. And like the web, you will end up getting errors from some percentage of those - or get very slow performance etc. You need lots of servers to be working correctly, not just your local server. If it becomes too hard to read news, say because every third article is not available, then this just won't work.

Then there's the issue that when you post an article, the header will propigate much faster than it has in the past. So then, you get lots of people trying to read the article in a very short period of time. So the problem isn't that you have to serve the article to 1000 other sites, but that you have to do most of that within 5 minutes of posting the article? It's hard to know how much of a problem that would really be without actually trying it.

What other problems are there with this idea?

--
Curt Welch   http://CurtWelch.Com/
curt@kcwc.com   Webmaster for http://NewsReader.Com/

B.2.2 Text Feeds vs Header-Only Feeds in Size

```
Path: nntp.uio.no!uio.no!news-spur1.maxwell.syr.edu!news.maxwell.syr.edu!
        newsfeed.usit.net!feed.newsreader.com!news2.newsreader.com!
        flame-test.newsreader.com!not-for-mail
Subject: Re: Header-Only feeds (was: A nice side benefit of WebReader)
From: curt@kcwc.com (Curt Welch)
Date: 17 Dec 1999 03:37:28 GMT
Organization: NewsReader.Com
Message-ID: <19991216223728.072$QK@newsreader.com>
Newsgroups: news.software.readers,news.software.nntp
References: <38569b7a$0$224@nntp1.ba.best.com>
        <6aJWOE2g9YI1qQ46VtGbCuiICWXm@4ax.com>
        <3856bc3d$0$229@nntp1.ba.best.com>
        <19991216152219.196$Pi_-_@newsreader.com>
        <brad-1F6669.00533517121999@news.skynet.be>
X-User: Curt@NewsReader.Com
X-Face: "p}G*1KH{+F7EYGKLb>ogDguabZ+%,?^epeFB!nzu‘)’$=QcvL1KF6<0GH!Tbc!Sqo[|tV5
        %IW48mQf3K=Ci&gZ7]]aazx@]Y-nq!r5{yH/#,?@lDdUDvOfByB2hVW0.@OM%{l/{cT’{w
X-Url: http://CurtWelch.Com/
Lines: 55
Xref: nntp.uio.no news.software.readers:69158 news.software.nntp:81319

Brad Knowles <brad@shub-internet.org> wrote:
> In article <19991216152219.196$Pi_-_@newsreader.com>, curt@kcwc.com
> (Curt Welch) wrote:

>     How about something like
> <nntp://news.your.dom.ain[:port]?user=name?pass=snark/message-id>?  Or
> does the URL syntax allow for userids and passwords already?

I keep getting confused on the syntax of the news: url.  But I think
it’s like the other ones so it’s something like:

  news:[//[user[.password]@]host[:port]/]message-id|newsgroup-name

With the < > missing from the Message-ID which BTW, sucks.

or is it user:password?  I forget.

People seldom use the host name in news: url’s so we don’t see them much.

> And where would this syntax be defined?

I tend to go to http://www.w3.org/ to start looking for that type of stuff.
There’s an RFC that defines urls isn’t there?

> > A header only feed at todays news volume levels is only around 1GB/day
> > and 100Kbps of bandwidth.  That’s two orders of magnatude smaller than
> > a full feed.
>
>     Last I checked, a text-only feed was about 1GB/day, or roughly
> 100Kbps bandwidth.  Are you saying that the headers of the binary
> articles are roughly as large as the bodies of the non-binary articles?

Last I checked I though it was closer to 2GB/day, but..

A header only feed of all the articles I’m sure is smaller than a full body
feed of only the text articles.  And adding the headers of the binary files
to a text only feed would make such a small difference that no one would
notice.
```

The binary files make up around 90% of the volume, but only 10% or so of articles. So adding binary headers to a text only feed will probably only increase the feed size by 5%. The day to day variance is around that so it would be hard to notice the difference.

And if you took the bodies off of a text only feed, making it a header only feed, the volume would only drop by maybe 30% (just a guess).

My 1GB/day number was just a rough estimate (say +- 20%). So to that same degree of accuracy, a text only feed, or a full header only feed or a text only feed with binary headers is all about the same size. For all these, we are still talking close to 1/100 the size of a full feed.

--
Curt Welch http://CurtWelch.Com/
curt@kcwc.com Webmaster for http://NewsReader.Com/

B.3 Katsuhiro Kondou Quitting

```
Path: nntp.uio.no!uio.no!news-spur1.maxwell.syr.edu!news.maxwell.syr.edu
        !nntp-relay.ihug.net!ihug.co.nz!usenet.net.nz!news.darkmere.gen.nz
        !not-for-mail
From: Simon Lyall <simon@darkmere.gen.nz>
Newsgroups: news.software.nntp
Subject: mesh.ad.jp stats page
Date: 18 Jun 2001 06:13:43 GMT
Organization: Darkmere Private Access Internet, Auckland, NZ.
Lines: 16
Message-ID: <9gk66n$olv$1@red.darkmere.gen.nz>
NNTP-Posting-Host: green.darkmere.gen.nz
X-Trace: red.darkmere.gen.nz 992844823 25279 203.109.158.194 (18 Jun 2001
        06:13:43 GMT)
User-Agent: tin/1.4.1-19991201 ("Polish") (UNIX) (Linux/2.2.17 (i586))
Xref: nntp.uio.no news.software.nntp:92777

I've just been informed by Katsuhiro Kondou that he will soon no longer be
the newsadmin at mesh.ad.jp and the news operation there will been losing
most peers.

Are there any other stats pages that are graphing a near full feed?

--
Simon J. Lyall | Very Busy | Web: http://www.darkmere.gen.nz/
"Inside me Im Screaming, Nobody pays any attention."  |  eMT.

"The slippery KY-covered gerbil seemed, to Patrick, to be eyeing him up
unpleasantly. The gerbil, obviously a veteran of the process, even
looked like he enjoyed it. "

       -   "The Heimlich Manouvre" staring P Dunford & Doctor von Heimlich.
```

Index