International Journal of Management, IT & Engineering
(ISSN: 2249-0558)

CONTENTS

Editorial Board

Editorial Advisory Board

Title

DIFFERENT ISSUE FOR HANDLING DIFFERENT CACHE STRATEGIES ON USENET

Author(s)

Harish Rohil
A.P. (C.S. & App. deptt.)/ CDLU, Sirsa

Jitender Yadav
M.Tech. (C.S. & App. deptt.)/ C.D.L.U, Sirsa

ABSTRACT:

Usenet is the name of a worldwide network of servers for group communication between people from 1979 and onwards, it has seen a fast growth in the amount of data transported, which has been a break on bandwidth and storage. This paper describes the history of Cache Strategies used for Usenet in a growth viewpoint and its different issues. Usenet volume has been growing exponentially for many years; this growth increasing demands on the cache strategies of a net news server for disk space. We wanted to find a best strategy that could easily handle to meet future growth of system administration effort. In this paper we present a high performance cache server for Usenet News that helps to conserve network bandwidth and disk storage. After a systematic comparison of existing news database formats we designed NEWSCACHE to remedy Usenet News bottlenecks. We present an experimental comparison of different cache strategies as well as an evaluation of the use of NEWSCACHE as a news server.

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 a notable 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 large sums of money on hardware components relieves the situation, but it does not solve it.

This paper is to find a way to describe the different cache strategies and its issues and describe NEWSCACHE for the today problem of news overload handling on Usenet. We have introduced the idea of advanced caching methods as a common enhancement for parts of the Usenet distribution network that has been done to relieve network bandwidth and storage capacity.

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 paper, firstly provide an introduction to Usenet, followed by Usenet's brief history from the viewpoint of growth, as well as suggestions for dealing with the volume of Usenet data traffic with the help of describe the different cache strategies and compare these cache strategies and development of Usenet cache technology of NEWSCACHE of existing database format of articles that stores on the news server.

The world-wide set of cooperating news servers makes up the distribution infrastructure of the News system. Articles are distributed among news servers using the Network News Transfer Protocol (NNTP) which is defined in RFC977. In recent years several extensions have been applied to NNTP. These are currently available as an Internet draft which will totally supersede RFC977 within the next some year. News readers provide the user interface for reading News and interact with their news server using the Network News Reader Protocol (NNRP). NNRP is actually the subset of NNTP that is used by news readers. Since the news reader stores data specific to each news server, such as article numbers to keep track of read articles, the user must always connect to the same news server to get a consistent view. The number of newsgroups (currently about 55000) and users is growing steadily. At the infrastructure level this implies growing amounts of data that need to be distributed and pushes existing News infrastructure to its limits. A news server maintains a set of databases and log files to store and monitor the news spool which are central to the performance of the system as a whole. These databases are explained along with a comparison of the organization of these databases as implemented in various well-known news servers and NEWSCACHE itself.

We present the design of NEWSCACHE in last. Firstly, we explain the different cache strategy used for used for a caching news server. Besides caching, our NEWSCACHE provides additional and new features which are described and evaluates our design.

Most of the caching strategies depend upon the use of an upstream peer that can provide articles on request.

1. Simple Caching Strategies:

Simple caching strategies do not require metadata to function properly.

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.

We 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 time span, the comparative overhead for fetching single articles may be relatively low. We 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.

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, we expect it to perform poorly, because of the lower level of caching compared to single article caching.

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, 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 we 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.

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, we think this method has a higher degree of bandwidth and disk usage, as well as bad user perceived performance.

2 Metadata Dependent Caching Strategies:

The following strategies require a feed of metadata from their upstream peer. As a minimum, 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.

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.

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. 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.

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.

This strategy has the greatest potential of all single strategies mentioned here. We expect thread caching to have a high hit rate and user perceived performance, as well as decent bandwidth and disk usage.

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: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 follow up 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. 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. 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.

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.

To measure, 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, we do not think it will be good for disk usage. Bandwidth usage is also likely to suffer, unless the hit rate is very high.

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. This method is too coarse grained to be of good use.

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, we do not see the point in pulling the articles, when today's method seems to be working adequately for that purpose. The prefetching of groups and hierarchies to be bad for bandwidth and disk usage, while very good for user perceived performance.

2.7 General header caching

Caching only article headers is an approach that will reduce the volume distributed by a great deal. 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 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 (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.

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.

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.

3.2 News reader controlled caching/prefetching

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.

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 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.

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.

4.2 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.

In this section we explain my purposed design of NEWSCACHE is a cache server implementation intended to access the News infrastructure. Since it only uses NNRP, it fits seamlessly into the existing infrastructure without requiring modifications to existing software. Each requested article is stored locally by NEWSCACHE to satisfy successive requests without having to contact the news server again. This eliminates additional transfers, thus conserving bandwidth; decreases load on the news server; and reduces disk space requirements since only articles that actually are accessed need to be stored.

5. News Database

Each news server maintains a set of databases that store articles and newsgroups as well as meta-information. We have also included NNTPCACHE's organization of the news database another cache server for News that was developed in parallel with NEWSCACHE.

5.1 Active Database

The active database stores a list of all the newsgroups available on the news server along with the number of the first article (low watermark), the number of the last article (high watermark), a posting flag, which indicates the type of the newsgroup, and the creation times of a newsgroup. Articles are numbered sequentially within a newsgroup. Articles posted to different newsgroups, will have several article numbers (one for each newsgroup). The article number is site-specific, depending on the arrival order and must never be reused within a newsgroup. NEWSCACHE, however, stores all this information in one memory mapped database including the number of articles available in each newsgroup and a timestamp when a newsgroup has been requested the last time from the news server.

An advantage of storing the number of articles within the active database is that articles need not be counted whenever a newsgroup is selected while still being able to provide an accurate count of the articles present in the newsgroup.

5.2 Article and Newsgroup Database

The article and newsgroup database stores all the news articles and maps the articles to the newsgroups which they have been posted to. Traditionally the newsgroup hierarchy is mapped onto a directory hierarchy and the articles are stored in separate files in their newsgroup directory.

If an article is posted to several groups hard links are used to prevent multiple storing of an article (c-news, NNTP reference implementation, NNTPCACHE). INN uses the same format but is able to use soft links which allows distribution of the News spool over multiple file systems.

5.3 Overview Database

The overview database stores a short summary for each article. Using the overview database, a news reader can provide a faster overview of the articles available in a newsgroup. INN stores the overview database for each newsgroup as a single plain text file. Whenever an article is posted to a newsgroup. If an article gets deleted or is expired in the newsgroup, the whole file has to be rewritten. NNTPCACHE takes the same approach except that the overview records for one group are split up into several files with 512 overview records per file. This is necessary since otherwise the whole overview database would have to be rewritten when a new record has to be inserted at the beginning of the file. This occurs frequently because NNTPCACHE has no control of when a client requests an overview record. Some news readers even request the overview records in reverse order, to be able to present the newest articles first to the user while older records are being retrieved (e.g., netscape). NNTPCACHE generates overview records on the fly if an article is not available and stores them in the overview database.

5.4 History Database

The history database stores meta information about articles and newsgroups. It stores the arrival time and expiration of an article along with its message identifier. This allows the news server to identify whether an article is offered again by another news server but has already been expired on the local server. It stores the creation and deletion time of newsgroups. The creation time is necessary to be able to inform news readers of newly created newsgroups.

6. Design of NewsCache:

NEWSCACHE has been designed to be easily extendible and reusable. For this purpose we design a news server class library that provides an interface to different kinds of news databases. The hierarchy is shown in Figure 1.1.

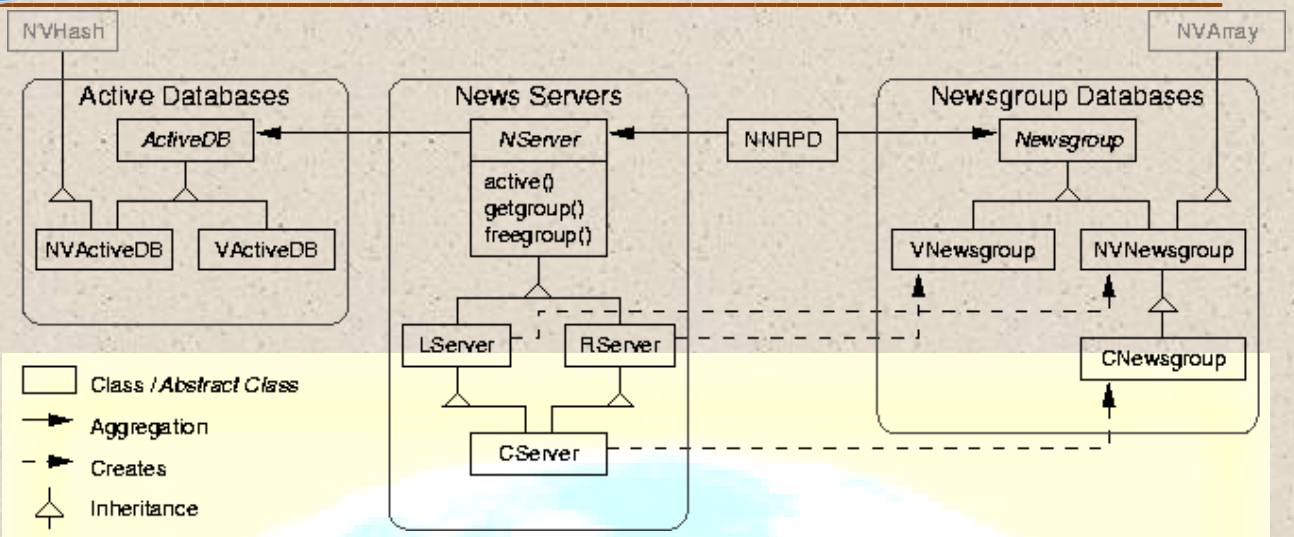

Figure 1.1: NEWSCACHE's class hierarchy

The abstract news server class (NServer) defines the interface that has to be implemented by all the news server classes and provides a factory method (getgroup()) that lets the user create the news server's news database. The local server class (LServer) implements an interface to a local news database. This class can serve as a base class for the implementation of a news server trimmed to the user's requirements. The remote server class (RServer) implements an interface to access a database provided by a news server. It also allows for multiplexing between different news servers on a per newsgroup basis. The RServer class can be used as the communication interface to a news server for a new news reader. The cache server class (CServer) inherits the functionality from the LServer and the RServer classes. As a result, CServer provides an interface to the database of a news server with the ability to cache messages in the news database provided by the LServer class. For the databases used by the LServer and CServer classes, we design a Non Volatile Container class library. Instead of using the heap for memory allocation, the class library provides its own functions for allocating and freeing memory by using memory mapped files. Using this memory allocation model we implemented a set of containers. NVcontainer provides all the methods necessary for allocating and freeing memory from the mapped file. This functionality is inherited by all the subclasses of NVcontainer. At the moment we provide a list container (NVList), an externally linked hash table (NVHash), and an array (NVArray). Figure 1.2 shows the hierarchy of this library.

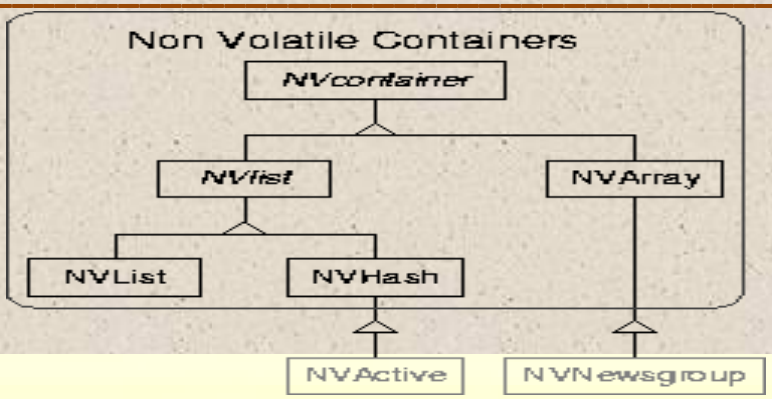

Figure 1.2: Inheritance hierarchy of the Non Volatile Container Class library.

Compared to other strategy, we provide a sophisticated newsgroup database. Only big articles (articles bigger than 16KB at the moment) are stored using a separate file. All other information, be it a small article or an overview record, are stored in the same database. This reduces the number of files and keeps related information together and thus improves caching behavior. This performance penalty (about 20%) is outweighed by the disk space reduction, since the overview database is up to 20% of the size of the news database.

CONCLUSION:

We have presented the Usenet technical problems with the it's continued growth. 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. So some 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.

Despite the fact that consensus exists that caching must be applied to News in the presence of overloaded networks, only few approaches exist to attack this problem. NEWSCACHE, however, attacks this at the access infrastructure while still being compatible to existing news software. NEWSCACHE can replace existing leaf node news servers thus reducing network bandwidth consumptions and reducing hardware requirements for the provision of Usenet News since only a fraction of the full News spool has to be stored while still providing a full feed to news users.

NEWSCACHE can be used to speed up retrieval of News in environments where only a slow link to the news server exists. Another advantage of NEWSCACHE is that it offers news reading functionality only (postings are directly forwarded to the upstream news server) and needs not allocate resources and computing power for news distribution. This drastically cuts down on I/O load. The news database is based on memory mapped files using our own memory management. This approach allows us to manipulate persistent complex data structures as if they were stored on the heap. Other news servers such as INN might also benefit from organization.

Additionally, NEWSCACHE makes the life of Usenet administrators easier by providing the following new features without forcing the administrator to install a news server with a full newsfeed: provision of local newsgroups, transparent merging of multiple news servers into one virtual news server, and providing a virtual full feed over slow links where a full feed would not be possible. These factors should make NEWSCACHE popular in the future. An increasing number of people already use NEWSCACHE including Internet service providers and NEWSCACHE is included in the Debian Linux distribution.

REFERENCES:

- [Barber, 2001] Barber, S. (2001). Network news transport protocol. Internet Draft.
- Sandvine Inc., "2009 global broadband phenomena," http://www.sandvine.com/news/global_broadband_trends.asp, 2009.
- [Bumgarner, 1995] Bumgarner, L. S. (1995). USENET — The Great Renaming — 1985–1988. http://www.vrx.net/usenet/history/rename.html.
- [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
- [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/.

- [Nixon, 2000] Nixon, J. (2000). Newsfeed Size — How big is Usenet, anyway? http://newsfeed-east.supernews.com/feed-size/.
- [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/.
- [Spafford, 1990] Spafford, G. (1990). Re: The List again] http://communication.ucsd.edu/bjones/Usenet.Hist/Nethist/0014.html.
- [usenet grownt 1997 to 2010] usenet service site . http://www. news-Service.com.
- Brian Kantor and Phil Lapsley. Network News Transfer Protocol - A proposed standard for the stream-based transmission of news. RFC977, February 1986.
- M. Odlyzko, "Internet traffic growth: Sources and implications," http: //www.dtc.umn.edu/mints/home.php, 2003.
- Global Internet Geography, "TeleGeography research," http://www.telegeography.com/product-info/gb/download/executive-summary.pdf,2009.
- H. Schulze and K. Mochalski, "ipoque internet study 2008/2009," http: //www.ipoque.com/resources/internet-studies/ (need to register), 2009.
- V. Paxson, "Bro: A system for detecting network intruders in real-time," Computer Networks, vol. 31, no. 23–24, 1999.
- R. Pang, V. Paxson, R. Sommer, and L. Peterson, "binpac: A yacc for writing application protocol parsers," in Proc. ACM Internet Measurement Conference, 2006, pp. 289–300.
- Thomas Gschwind. A Cache Server for News. Master's thesis, Technische Universität Wien, April 1997. http//ww.infosys.tuwien.ac.at/NewsCache/.
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, October 1994.