Drinking from the Firehose: Multicast USENET News

Paper Author: Kurt Lidl
Code Author: Josh Osborne
Code Author: Joseph Malcolm

News transport and spooling systems of the last several years have concentrated on decreasing the resource load on news servers. One beneficial side effect has been the average decrease in time that a news system spends on a given article. This paper describes a novel USENET news transport protocol, which we call *Muse*. The two major motivations behind *Muse* are to reduce the average propagation delays of articles on USENET and to further reduce the resource load on a centralized news server. *Muse* runs on top of the experimental Internet multicast backbone, commonly referred to as the *Mbone*. Major design and implementation issues are discussed. Security concerns of multicast news are discussed and our solution is examined. The problems of scaling news distribution to thousands of hosts are also addressed.

1. Introduction

USENET is a distributed messaging system, which is implemented as a software system layer over a variety of networks and connections. The basic message unit is the *article*. Articles are logically grouped into *newsgroups*. The format of each article is described in [Adams87].

One of the primary concepts of the USENET "network" is that of a *news-neighbor*. A site's news-neighbors are defined as those sites that have bidirectional transfers of articles with a given site. Each news site has a unique identifier, which is known as that site's *news-name*. All traditional news systems know the *news-name* for all their news-neighbors. Each USENET site that generates an article passes that article to its news-neighbors, based on several control criteria. These criteria are based on which newsgroup an article has been posted to, what the distribution of the article is and whether or not the news-neighbor has seen the article before.

The *Path:* header of each article is one of the mechanisms used to prevent message loops from occurring between news sites. The loop prevention logic embodied by the *Path:* header is executed on the news system that is sending an article. Before sending an article to its news neighbors, each site prepends its *news-name* to the *Path:* header. News systems are required to check the *Path:* header against the news-name of the candidate receiving site. If the news-name of the receiving site under consideration is already in the *Path:* header, the article has already passed through the site and does not need to be queued for transit there. Thus, the sending news site can safely not transfer the message to that receiving site.

The second form of duplication suppression is implemented by the receiving news site, in the form of *message-ID* checking. Each news site keeps a database of the *message-IDs* of all the articles that have been received by that system. These records are normally kept for two to three weeks and then purged from the database. When a news site receives an article or is offered an article via NNTP, it checks for the existence of the *message-ID* in the database. If the site finds the *message-ID* already in the database, the news system has already received the message and can safely ignore the duplicate copy of the article. For large news systems with many news-neighbors, connected via high speed communications lines, the resources spent doing duplicate suppression can be very significant. Doing the *message-IDs* check is the equivalent of a database search operation, which must be performed at peak rates of over two hundred times a second, on busy news servers.

Traditionally, there have been two major news transfer methods – UUCP spooling and NNTP. NNTP is the newer of the transfer protocols and features a SMTP-like dialogue for each article that is sent. It is worthwhile to note that NNTP requires a TCP connection between the sender and the receiver of the news. UUCP spooling consists of writing multiple news articles to a single batchfile and then using *uucp* to copy this file to the receiving system, where the news is processed. UUCP does not require a TCP connection and can be configured to run over a wide variety of connections, including dialup modems, X.25 networks, directly connected serial ports and TCP/IP networks.

2. Motivations behind Multicast USENET News

The major design goal for *Muse* is to significantly reduce the average article propagation delay through USENET. This decrease in propagation time is beneficial to the users of USENET, as it changes the perception of the network making it seem that it is more responsive to questions and answers. Additionally, the decrease in propagation latency of articles has the effect of smoothly distributing a graph of the traffic flowing over backbone network links.

Muse was carefully designed so that an unlimited number of news listeners can be supported from a single multicast news server. News distribution via *Muse* should scale without limit, as the total number of listeners is bound only by the size of the *Mbone*, and not by exhaustion of centralized resources [Salz92], as is the case in all traditional news systems. Each additional listener consumes only the local resources of the listener (e.g. network bandwidth, disk I/O, CPU cycles). In all cases, the same amount of effort is expended on the *Muse* transmitter for no *Muse* listeners as for a thousand *Muse* listeners.

Muse was conceived in such a way that widespread deployment of this software should actually decrease the amount of traffic due to USENET news that travels various backbone connections. Since *Muse* performs a flood-fill of a single news article over a significant portion of the Internet, only one copy of the article should traverse most backbone links. Common sense dictates that it should be much more efficient to implement a broadcast mechanism via a multicast network, rather than modeling the broadcasts via many unicast network links. USENET, after all, is a broadcast network that is currently implemented over many unicast links.

A final motivation of the authors was the hope that by basing *Muse* on the *Mbone*, it would encourage others in the networking community to write innovative software that addresses old problems in a new fashion. By basing our transport system on a multicast transport, we wish to give another reason to stabilize the *Mbone* so that it will be more reliable in the future and other production services might be offered on top of it.

3. General Architectural Overview

Muse is a USENET news transport layer that is implemented in two logical parts. The first part is the news transmitter, which accepts news articles, checksums the article, digitally signs the checksum using the RSA algorithm and multicasts them. The listener receives the multicast, checksums the article independently and verifies the checksum against the encoded checksum included with the article.

4. Introduction to Technologies Used

Muse uses several technologies that are not yet in widespread use across the Internet. Because these technologies are not all in day-to-day use across the Internet, many people are unfamiliar with them. A short description and explanation of these technologies is contained in this section of the paper.

4.1 The Mbone

The *Mbone* is the experimental multicast backbone that various network service providers, universities, research institutions, and corporations have layered on top of the Internet. It can best be described as a large distributed experiment in multicasting. To better understand the *Mbone* it is important to understand the basics of multicasting. The following description of multicasting datagrams is taken from [Deering89].

> IP multicasting is the transmission of an IP datagram to a "host group" a set of zero or more hosts identified by a single IP destination address. A multicast datagram is delivered to all members of its destination host group with the same "best-efforts" reliability as regular unicast IP datagrams, i.e., the datagram is not guaranteed to arrive intact at all members of the destination group or in the same order relative to other datagrams.

> The membership of a host group is dynamic; that is, hosts may join and leave groups at any time. There is no restriction on the location or number of members in a host group. A host may be a member of more than one group at a time. A host need not be a member of a group to send datagrams to it.
>
> A host group may be permanent or transient. A permanent group has a well-known, administratively assigned IP address. It is the address, not the membership of the group, that is permanent; at any time a permanent group may have any number of members, even zero. Those IP multicast addresses that are not reserved for permanent groups are available for dynamic assignment to transient groups which exist only as long as they have members.

4.2 MD5 – Message Digest checksum

In the *Muse* system, we have a need for a fast, cryptographically secure checksum algorithm. We selected the MD5 Message Digest algorithm. The following description of the MD5 article checksum is taken from [Kaliski93].

> A message-digest algorithm maps a message of arbitrary length to a "digest" of fixed length, and has three properties: Computing the digest is easy, finding a message with a given digest—"inversion"—is hard, and finding two messages with the same digest—"collision"—is also hard. Message-digest algorithms have many applications, including digital signatures and message authentication.
>
> RSA Data Security's MD-5 message-digest algorithm, developed by Ron Rivest [Rivest92], maps a message to a 128-bit message digest. Computing the digest of a one megabyte message takes as little as a second. While no message-digest algorithm can yet be *proved* secure, MD5 is believed to be at least as good as any other that maps to a 128-bit digest. Inversion should take about 2^{128} operations, and collision should take about 2^{64} operations. No one has found a faster approach to inversion or collision. [Robshaw93]

4.3 Public key cryptography systems

The *Muse* system requires a method of distributing the message-digest for each article in a form that may be deciphered by the *Muse* listeners, but may not spoofed by illegitimate multicast transmitters. The RSA public key cryptography system was chosen as the public key system for use, mainly due to its freely available (in the U.S.A.) reference implementation. The following description of how public key cryptography systems is taken from [Fahn93].

> Traditional cryptography is based on the sender and receiver of a message knowing and using the same secret key: the sender uses the secret key to encrypt the message, and the receiver uses the same secret key to decrypt the message. This method is known as secret-key cryptography. Public-key cryptography was invented in 1976 by Whitfield Diffie and Martin Hellman in order to solve the key management problem. In the new system, each person gets a pair of keys, called the public key and the private key. Each person's public key is published while the private key is kept secret. The need for sender and receiver to share secret information is eliminated: all communications involve only public keys, and no private key is ever transmitted or shared.

4.4 The NNTP Protocol

The *Network News Transfer Protocol* as described in [Kantor86], provides a news articles transfer protocol that runs on top of a reliable stream connection, such as TCP. The protocol's sample implementation was originally released as the software package called *"nntp"*. This software package has been widely ported and is available on a large number of systems. NNTP has a well defined protocol command set, response codes and transfer mechanism for news articles. Additionally, NNTP has the distinction of having been independently implemented several times. For IP connected sites, NNTP is the preferred mechanism for transferring USENET news.

The NNTP *"IHAVE"* command specifies a the start of a command sequence that a news server and client typically exchange. The news server will assert "IHAVE <message-id>" for a given message. The receiving NNTP host will respond "335 send article," which indicates that it has not seen that particular message before and desires a copy of it. Or, the receiver will send "435 article not wanted," which typically indicates that the receiver has already gotten a copy of that article and does not need a duplicate copy.

5. Muse: Overview of Operation

Muse operates in a classical transmitter–listener scenario. The reason that *Muse* can multicast to an unlimited number of machines is that *Muse* requires that *no* packets are sent from the listener back to the multicast transmitter. As a consequence of this, it is easy to break the the examination of the transmitter and the listener into two discrete parts. In fact, the *Muse* system currently consists of two programs, *n2m*, which implements the transmitter functions, and *m2n*, which implements the listener functions.

Muse's transmitter accepts the article to be multicast through a standard NNTP "IHAVE" protocol transaction. This approach was taken to minimize the the integration effort required to make *Muse* talk to existing news systems. Because of this standard interface to other news systems, *Muse* just looks like another news feed. As such, all the customary NNTP news feed controls available on a NNTP server may be applied to the *Muse* newsfeed.

Once *Muse* has received the article, it checks to see if *Muse*'s news-name already appears in the *Path:* header. The news system that is sending articles to the *Muse* transmitter should not attempt to feed any such articles to *Muse*. The extra check is fairly cheap and prevents a potential waste of bandwidth. *Muse* follows the guidelines listed in RFC1036 [Adams87] for parsing the *Path:* header. If it finds its news-name already in the header, it will drop the article. This is done to ensure that a given *Muse* transmitter will not multicast an article a second time. If *Muse* does not find its name, it adds its news-name to the *Path:* header, fills in the rest of the protocol header fields and calculates the MD5 checksum of the article. Once the *Muse* listener has the MD5 checksum, it signs the checksum using its private key. The complete *Muse*-modified article is then ready to be multicast to the *Muse* listeners. The transmitter will send the *Muse* article to the appropriate pre-arranged multicast group as a single (possibly large) UDP packet. In its present form, *Muse* depends on the operating system to fragment the UDP packet correctly to ensure transmission across any networks.

After being transmitted, the news article (along with the *Muse* header), is in transit on the *Mbone*. The articles representation is one or more IP packets, representing the entire UDP packet. The *Mbone* will deliver this packet stream to those machines that are interested in receiving the news multicast.

It is important to keep in mind that the receiving and verifying stage of the *Muse* transport system happens more or less simultaneously on all *Muse* listeners across the *Mbone*. Thus, each listener will go through this process.

The *Muse* multicast listener (*m2n*) performs work according to the following outline. The first job is to alert the operating system kernel to its desire to listen on the correct (pre-arranged) multicast channel. The operating system kernel is relied upon to defragment the collected IP fragments into a complete UDP packet. If not all the fragments of the original UDP packet make it to the destination host, the operating system kernel of the destination will drop the remaining packets of the article.

Once the entire *Muse* article has been received, a combination of the received key number and the last news-name in the *Path:* header is used to look up the appropriate public key in the local matrix of public keys. If no key is found for the sending news host, the article is dropped and the condition is logged to a file on disk. Assuming an appropriate public key is found for the transmitting news site, the listener calculates a local version of the MD5 checksum for the article as it was received. Then, the listener extracts the received MD5 checksum from the *Muse* header, and compares it to the locally generated checksum. If the comparison shows a difference, the article is dropped, as one of the following invalidating actions has occurred: the article was mangled or modified in transit, the article was sent by a multicast server that was spoofing an authentic *Muse* server, or the public key used to decrypt the MD5 checksum was invalid. If the article was rejected for any of these reasons, a timestamp and message-ID (if one can be recovered from the article) are logged to a file on disk, to assist in problem tracking. Assuming the checksums matched the received and verified article is then sent via a NNTP "IHAVE" transaction to the receiving NNTP server. Note that the receiving NNTP host does not need to be the host running the *Muse* listener software.

Muse provides a mechanism for sending various control messages across the *Mbone* to all running *Muse* listeners. These control message are intended for collecting listener audience sizes and statistics gathering. *Muse* has implemented a *Version* control message, which is similar in spirit to the more traditional USENET *Version* control message. In addition to the statistics gathering message types, a *Revoke* message has been implemented, to partially address the issue of key management for the public keys that *Muse* uses. If a *Revoke* control message is received, the public key that was used to sign the message is permanently revoked and should be removed from use. Another control message that has been implemented for this release of *Muse* is the *NOP* (no operation). This message is broadcast once every 60 seconds, if no other news articles have been sent during that time period. This is intended as a *Muse* keepalive mechanism, to be used in conjunction with the results of research being conducted into how to best handle failure of the primary NNTP news feed into the *Muse* transmitter. The final control message implemented is the *News* control message. It is the "standard" type of packet sent by the *Muse* transmitter and is used to indicate a news article is contained in the packet.

6. Muse: Justification of Design Decisions

- Why use UDP at the transport layer?

UDP was chosen as the transmission method for two major reasons. First, since it is a datagram based protocol, UDP maps closely to the type of traffic that characterizes USENET news. This type of traffic could be characterized as having many totally independent packets, with no retained state across any of the packet flows. This is very important if one looks at the scaling issues of running a global news system. As UDP does not require any acknowledgment of received packets, it is ideal for building an application that will scale without bound. Second, UDP was chosen because there is no need to retain any state across the server and the listeners. This stateless nature makes *Muse* relatively maintenance free in the face of network outages and server crashes.

The simplicity of the UDP model was the reason that we choose to allow IP fragmentation for breaking apart large articles, as opposed to coding our own article fragmentation system. While relying on IP fragmentation, which limits *Muse* to a theoretical maximum 64 kilobyte sized packet and an operating system imposed 9 kilobyte article size limit (in both cases, minus the *Muse* protocol overhead), analysis has shown that 9 kilobyte packets are sufficient to transfer approximately 96% of all news articles.[1]

- Why is fragmentation left to the UDP layer in the kernel?

The decision to rely on UDP fragmentation does not close the door on implementing a different fragmentation scheme for later releases of the software. The option of coding this for the initial release was deemed unnecessary, as the stability of the *Mbone* also comes into play. The *Mbone*, as it exists currently,[2] sometimes exhibits severe packet loss during high network loads. The probability of receiving 64kilobytes of data with no fragments missing was deemed unlikely. We are content to accept the artificially low limit of 9000 byte articles in the initial release of the software. While it would be possible to retransmit these articles, the *Muse* transmitter has no way of knowing that the article was dropped in transit, nor does it have a way of "replaying" a given article, as it retains no state.

We are examining other work that has been performed in this area, notably the *Coherent File Distribution Protocol.* [Ioannidis91] Further discussion of this topic is in the *Future Work* section of this paper.

- When selecting a public key cryptography system, why use RSA instead of NIST's Digital Signature Standard (DSS)?

The decision of choosing RSA's public key cryptography system over NIST's was influenced by the following reasons. There was a freely available RSA reference implementation. There is not, to the best of

[1] Based on traffic analysis performed on two weeks of articles passing through the news site *darwin.sura.net*, during February and March, 1993.

[2] July–October, 1993

our knowledge, a similar reference implementation for the DSS. The other mitigating factor was that the RSA algorithm takes a long time to sign each checksum, but the decode of the checksum is relatively quick. With the DSS, the opposite is true – the initial signing of the checksum would be quick, and the verification stage would be slow, due to high CPU usage.

- Why do we need to RSA sign the articles at all?

To the casual observer of USENET news, it would not seem necessary to be concerned with the problems of security along this transport path. However, we contend that security of this transport is paramount to *Muse*'s success. What made the authors of *Muse* so security conscious? Because the transmitter and listeners collect no state during normal news transmission, it would be easy for anyone to inject a forgery into the news stream, simply by source-routing an "article" to a site that is known to be collecting news via the *Muse* multicast. This packet could have had all the incriminating data in it removed – e.g. the source IP address could be fake or the *Path:* header could be wrong. There would be no method to determine if an article was faked without the digital signature, leaving the listener no option but to believe the received article is valid.

This would have severe repercussions to the news community. Since the goal of the *Muse* software is to make the transport of news nearly instantaneous from coast–to–coast, it raises the possibility of USENET terrorism to new heights. Not only could people receive articles very quickly, but they could cancel them just as quickly. Often, this would be before another human ever saw the article! Silencing an unruly upstart on a newsgroup would be a trivial matter of coding and would be entirely untraceable, without the digital signing of the articles.

The traditional USENET news article has a *Path:* header that provides a trace of which news sites an article has traveled through. While this trace is not completely trustworthy in the case of forged articles, it is possible to track down where a forged news article was injected with the assistance of the news administrators along the way. This is mainly due to logging of the sites to which a news system forwards a copy of a given article. With multicast IP traffic, the situation is considerably more difficult. In fact, this problem is intractable with the current multicast system. Just as no IP routers can effectively track all packets sent through them on a packet by packet basis, no multicast router can log all packets traversing that machine. With traditional news systems, only connections and connection summaries need to be logged to trace the flow of news. With *Muse*, the headers of every packet would need to be saved in order to provide a comparable level of logging to allow tracking of which articles came from where. Because the source address on UDP packets is nearly worthless as a security check, a different authentication mechanism needed to be deployed, one where the identity of the *Muse* transmitter could be verified by the listener.

- Why have it use NNTP, rather than designing something else to take its place?

NNTP is a well established protocol, with many robust implementations in existence and use. Using NNTP as both the initial acceptance mechanism for the newsfeed and the final presentation protocol allows rapid widespread deployment across the Internet. Since the *Mbone* is not widely available currently, it was deemed useful to be able to receive articles on an *Mbone*-reachable machine and direct the resulting NNTP feed to a host that is not necessarily directly on the *Mbone*.

- Why does *Muse* have a news-name when it is not a complete news system in the traditional sense?

We chose to give the *Muse* transmitter a news-name, even though it is not a complete news system in the traditional sense. This was done to allow statistics gathering on articles that are multicast and accepted by other news systems. This will allow immediate generation of "news influence" ratings, as the software that generates these statistics is already in place across the USENET backbone.

The second reason for adding our own news-name to the article is to avoid a duplicate multicast of the article in the case that the transmitter is sent an article a second time. This will become more important in the future, when the transmitter host has the ability to accept more than a single incoming news-feed.

- How is access control implemented in the *Muse* transmitter?

Quite frankly, there is currently no access control mechanism built into the *Muse* transmitter (*n2m*). We rely on *tcpwrapper* [Venema92] to provide access controls to the machine that runs the *Muse* transmitter. *Tcpwrapper* allows replacement of a network daemon by a stub program, that validates or rejects network connections before actually invoking the real network daemon.

Logically, access control for the *Muse* listener machines is divided into two parts, that of the link between the transmitter and the listener and the link between the listener and the NNTP server that will actually accept the news. The UDP based "connection" from the transmitter to the listener is protected by the RSA signed MD5 checksum on the file. The connection between the *Muse* listener and the NNTP receiver is handled via the traditional access controls to sites running NNTP transport systems.

- How does *Muse* get around the problem of dropped articles and an unreliable transport layer?

In the current implementation, *Muse* does not directly address the case of resending dropped packets in a given article stream. Because *Muse* is not a reliable news transport mechanism, it must be used in conjunction with a backup, fully redundant news transport system, such as a traditional NNTP newsfeed or other backup newsfeed (possibly via an IHAVE/SENDME newsfeed). It is hoped that by adapting basic ideas behind the *Coherent File Distribution Protocol* we can design a way of extending *Muse* to allow resending of dropped article packets. Additionally, implementation of a protocol derived from *Coherent File Distribution Protocol* would provide a method for transmitting articles more than 9000 bytes in length.

Depending on the implementation, adding this code could complicate the *Muse* software greatly. It also has the undesirable effect of limiting, at least to some extent, the number of listeners that can be supported from a single multicast transmitter. This limitation is driven by the need for communication to the transmitter from listeners that have received damaged articles, either due to packet corruption or due to fragment loss.

- How is buffering implemented?

Muse needs no complex buffering system, such as an on-disk article spool that other news systems typically use. (Which, while perhaps not commonly regarded as a "buffer", serves that purpose for article forwarding.) Since the authors of the *Muse* software valued reduced transfer times and ease of administration over reliability of transport, the buffering system that has been developed for *Muse* was both easy to implement and does not require any cleanup after a server crash. The basic idea is to request a fairly large buffer from the kernel (64 kilobytes) on the connection (for both the transmitter and the listener). Since the transmitter is driven by a NNTP server that has already performed all disk I/O, disk latencies are not expected to matter for the transmitter. The transmitter can multicast the article as soon as it receives the article via NNTP and calculates the RSA signature of the MD5 checksum. The kernel will buffer outgoing articles on the transmitter side and multicast them as the ethernet device becomes free.

The *Muse* listener is in much greater need of buffering, since it must offer the articles to other news systems, which must eventually commit the articles to a disk. Traditional news systems almost always have some amount of delay, either due to disk I/O in retrieving the message-id from the *history* file, or in actually committing the article to the disk. During the wait for disk I/O to complete, it is not unreasonable to expect more news to arrive at the listener system, which is what drives the need for effective buffering on the listener. The listener will only remove a single article at a time from the the queue of de-fragmented UDP packets received. The remaining space in the UDP queue will act as a buffer for up to five maximally-sized articles. Because the UDP listener buffer is a queue of items, we can be assured that the oldest articles in the queue will be processed first. Unfortunately, the news articles towards the front of the queue are least likely to be worthwhile processing, as they are older and more likely to be duplicate articles.

7. Performance Numbers and Traffic Analysis

The test data for the following analysis was collected over approximately one month. We have results of our testing from four different news sites, which all entered into the test on different days. Because of this, the number of articles that were sent to each site varies.

Our first graph [Figure 1] shows the total number of articles that we believe were "eligible" to be received by each of the listeners. This number is based on the accounting logs that were generated by each listener. Our method for collection of the statistics gathered the results every eight hours. During each eight hour run, an assumption is made that the listeners are eligible for all articles between the first and the last article appearing in the transmitter's logs that were multicast during that time period. Thus, if a listener first logs reception of a multicast article four hours into a time period, that listener's statistics would at most be accounted for the articles that were multicast during the next four hours of time.

This graph has been included to allow the reader to judge the reliability of the data in the following graphs. All four sites shown have been operating the receiver software long enough to receive at least a thousand articles, and three of them have over three thousand articles. We believe this is a sufficient volume to allow analysis without too much danger of anomalous data coloring the results.

The significant variation in number of articles received is primarily due to the fact that there is a wide difference in the length of time each site has been operating the receiver software. The other factor which is likely to have an effect is the reliability of the news system that is receiving the article. There were also a few problems in the receiver (since believed fixed), which caused the receiver not to run after a reboot until restarted manually.

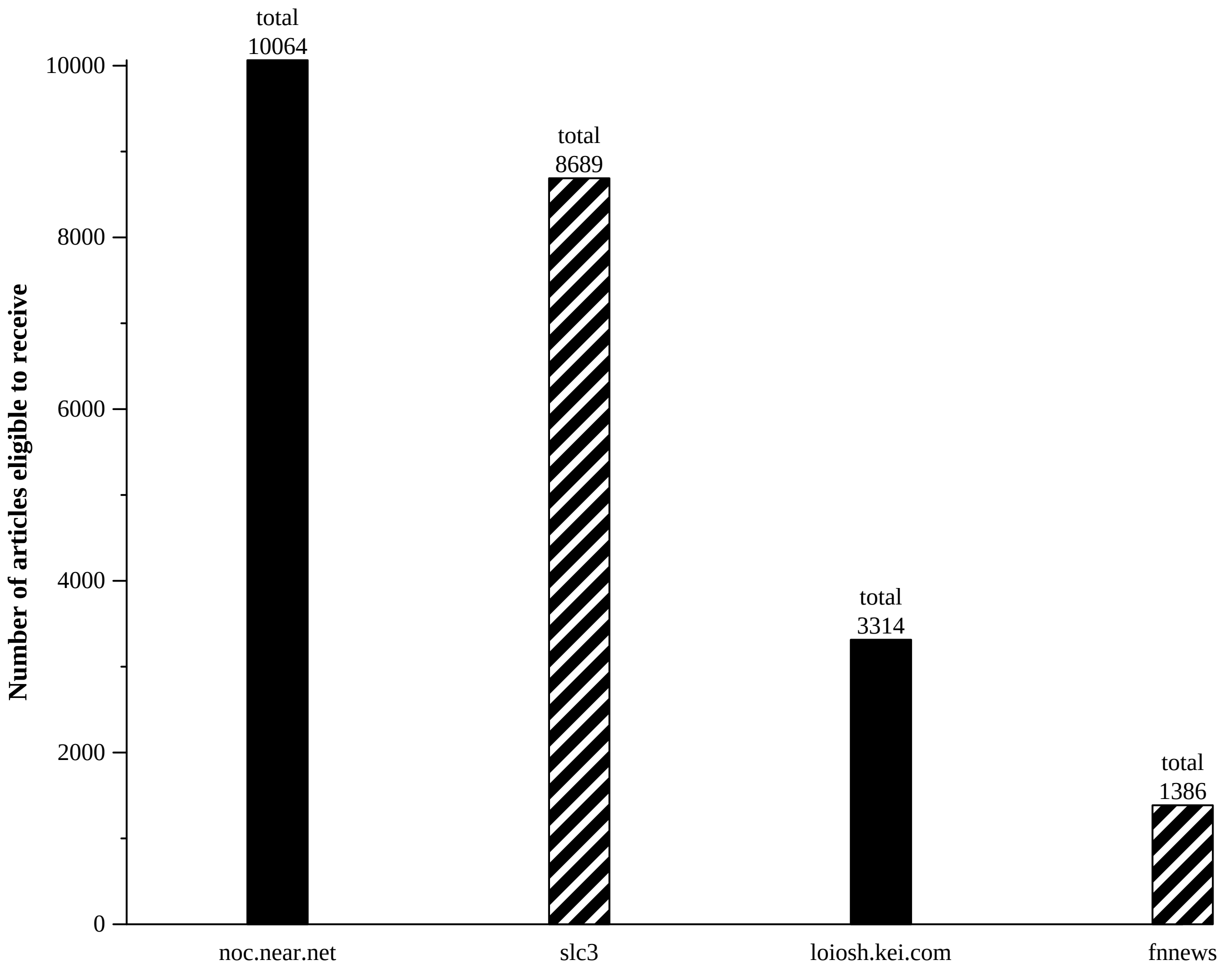

Figure 1

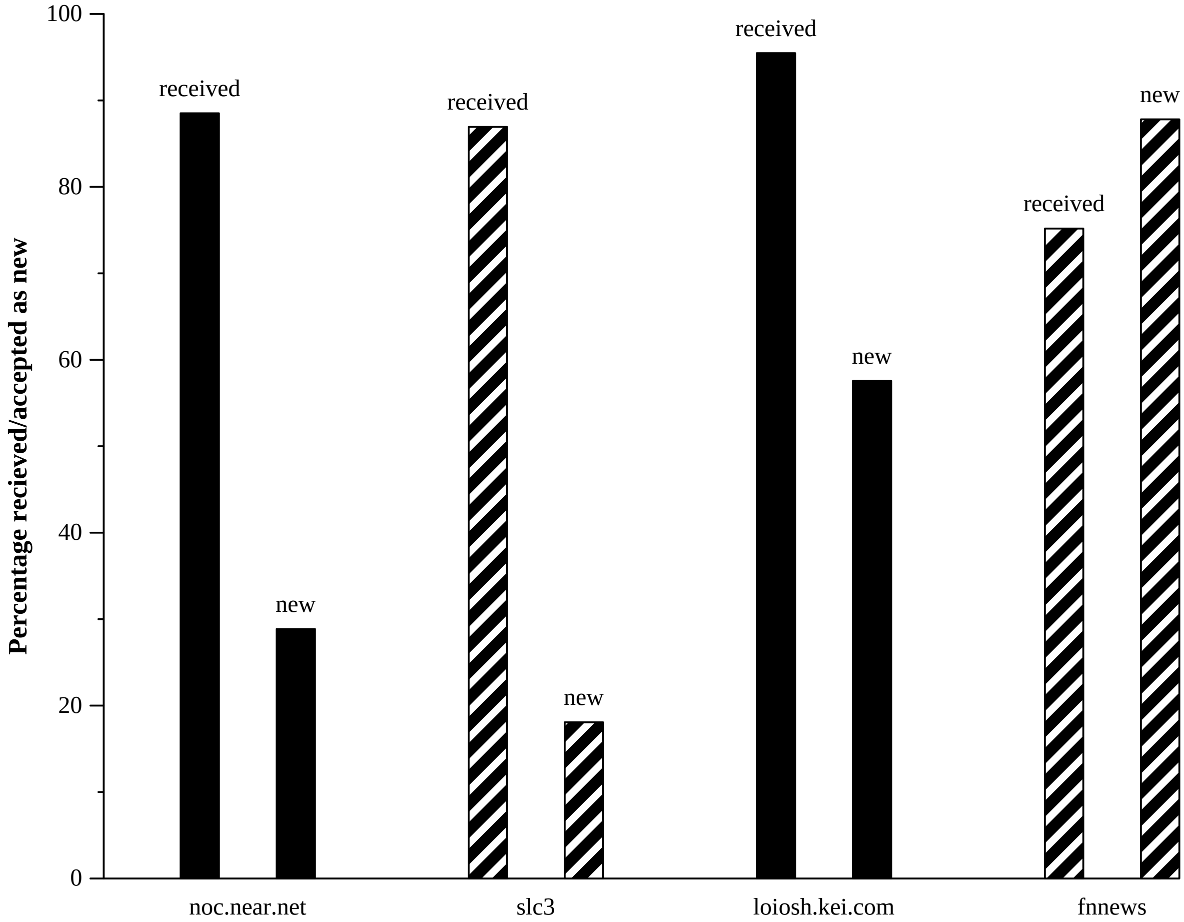

Figure 2

This bar chart [Figure 2] shows the percentage of complete articles (see Figure 1) that were received at the test sites. Additionally, it graphs the percentage of the received articles that were accepted by the NNTP server to which the listener offers the article. Because the "new" column is a percentage of received articles (as opposed to percentage of eligible articles) the "new" value can be larger then the "received" value, as is the case for the site fnnews.

The variation in the articles received at the various receivers can be mostly attributed to the widely varying placement of the receivers on the mbone. The closest is about two hops away, and the farthest is over ten.

The rather low percentages of "new" articles accepted at some sites is probably due to the quality of the "backup" news feeds to those sites. If the "backup" news feeds are sufficiently fast, they will deliver a given news article to the news systems before *Muse* can calculate the RSA signature and multicast the article.

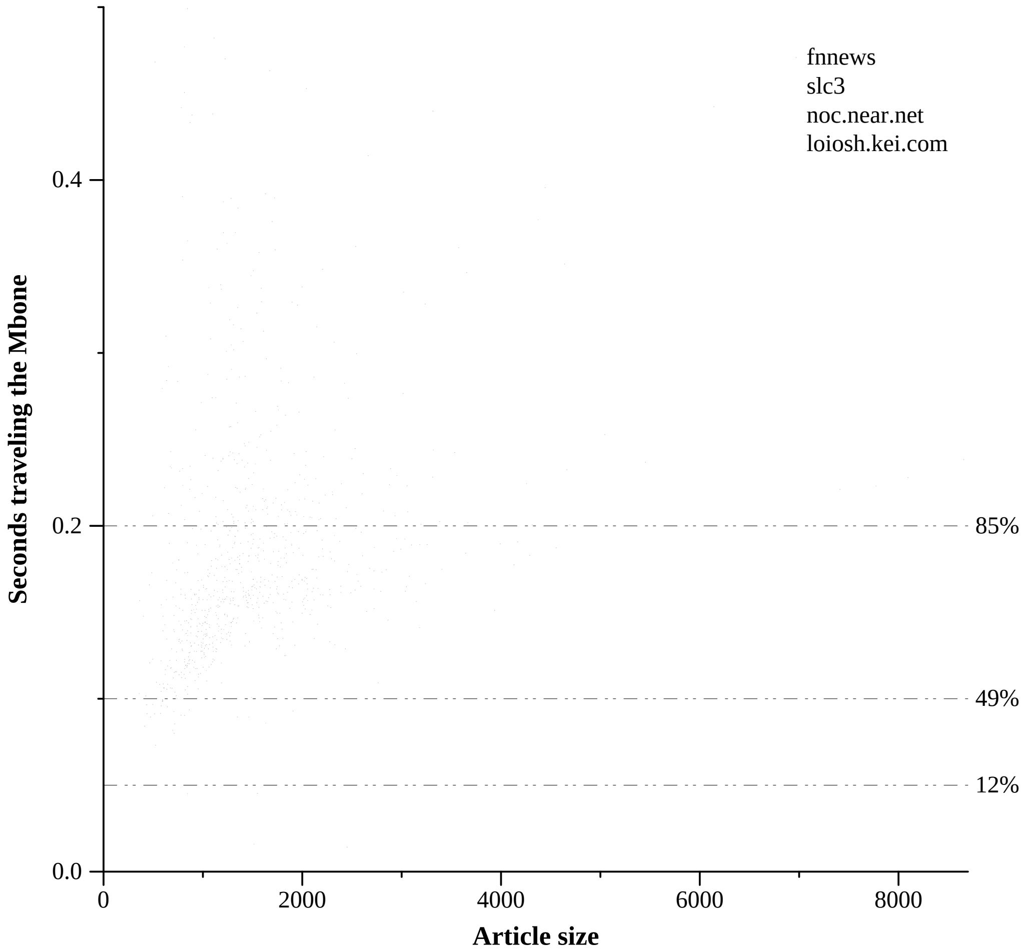

Figure 3

This scatter plot [Figure 3] shows the time in travel on the *Mbone* for several thousand articles. The major points of interest in this graph are the average small size of the articles and the low latency (almost always less then 200 milliseconds, and in fact quite often less then 100 milliseconds) time of the *Mbone* as a transport network. It is also interesting that the article size has a minimal effect on the time it takes to travel the Mbone.

There are several anomalies on this graph. A small number of articles (about five) appear to have arrived via before they were sent - we think this is most likely due to the clocks on the sending and receiving machines being out of phase. (Although all machines involved are synchronized using NTP, all the clocks involved will not match exactly in all cases. This is especially true for Sun SPARC machines running SunOS which disable clock interrupts when writing to the console.) There is also the smattering of times far above the mean. One possible cause of this is the fact that the timings are from user-space to user-space, and thus are affected by paging activity and competition for CPU time with other processes. Some of the delays are also attributable to the synchronous nature of the listener, which waits for any outstanding NNTP transaction to complete before it checks for new incoming news. (Both C News and INN can exhibit occasional relatively long delays in IHAVE transactions.)

(The rather noticeable "line" on the left side of the plot was caused by several hundred cancel messages sent out from brl.mil, closely clustered at and below size 487.)

8. Muse: Future Directions

- Faster Signing

One of the current limitations on the volume of news multicast is the fairly considerable cpu burden imposed by signing each individual message, which has caused us to limit ourselves to several hundred articles a day. We are looking at ways of improving this. (Either better code, a faster machine, or perhaps more than one article per packet.)

- Increased Multicast Selectivity

Once we can consider multicasting some large subset of all news traffic, it will become more interesting to break down the articles into multiple channels, to aid both receivers in selecting which articles to receive and the *Mbone* routers in insuring that each site receives only what it is interested in. This will be done by having the transmitter select a multicast group to send to based on the newsgroup of the article. Crossposts will probably be handled by broadcasting the article multiple times.

- Implementing full control message classes, message reply sending algorithms, timers, statistics gathering methods

Muse supports only the most basic control messages in this release of the software. Future work will be directed to more fully instrumenting the transmitter with a wider range of the possible control messages and the listener with the additional code to allow them to react to the control messages. Future control messages will allow for the gauging of the size of the listener pool at each distance from the transmitter, where the distance is measured in the hop-count to the transmitter. It is worthwhile to note that the transmitter of the control codes is not necessarily the machine that will handle the resulting replies from the *Muse* listeners. The basic control message will have a list of possible IP addresses to return the control message response to, to avoid overloading a single machine with the number of responses. Additionally, the response reply code will have a random-timer function in it, to avoid having all the machines that will be responding answering at once.

- A better receiver

The current receiver could be considerably improved in other ways as well. For example, the receiving and offering of the articles could be decoupled, which would allow the collection of slightly more accurate statistics. The receiver could also be able to offer a given article to more than one news server.

- What happens if the main transmitter or news sender goes down?

We are considering automatic switchover to another *Muse* transmitter or another news sender in the case of failure. A keep-alive control message would need to be implemented so that a secondary transmitter would not attempt to take over during periods with no news.

9. Conclusions

We developed *Muse* in order to make the propagation of USENET news more efficient, in both bandwidth consumed and speed of delivery. An important constraint on the solution was our requirement that it scale to hundreds or thousands of hosts. We believe that initial testing of *Muse* indicates that it can meet these goals. Specifically, utilizing the *Mbone* we can provide 200 millisecond simultaneous delivery for the vast majority of all articles to well-connected sites, with the number of possible receivers restrained only by the *Mbone* itself. We intend to pursue further testing and development of *Muse*, particularly in increasing the possible throughput.

10. Acknowledgments

- Thanks to Dave Hsu for suggesting the title for this paper. (hsu@pix.com)
- Thanks to Debbie Greenberg for proof-reading innumerable drafts of this paper. (dag@pix.com)
- Thanks to the authors of INN and C News for creating efficient news systems that inspired our own efforts.

- Thanks to Dennis Ferguson for the NTP timestamp code that we borrowed from xntp3.
- Thanks to those who have made the *Mbone* what it is today.
- Thanks to UUNET Technologies for allowing us to work and release this stuff, as well as running the *Muse* transmitter.

11. Availability

The source code for the client side implementation will be available via anonymous ftp from the host *ftp.uu.net* in the file */networking/news/muse/muse-client.tar.Z.*

12. References:

[Adams87] Rick Adams, Mark Horton. *Standard for Interchange of USENET Messages* Request For Comments 1036, Falls Church, VA: Center for Seismic Studies, December 1987.

[Deering89] Steve Deering. *Host Extensions for IP Multicasting* Request For Comments 1112, Stanford, CA: Computer Science Department, August 1989.

[Fahn93] Paul Fahn. *Frequently Asked Questions About Today's Cryptography* Redwood City, CA: RSA Laboratories, September 1993.

[Ioannidis91] J. Ioannidis, G. Maguire Jr. *The Coherent File Distribution Protocol* Request For Comments 1235, Columbia University: Department of Computer Science, June 1991.

[Kaliski93] Burton S. Kaliski. *Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services* Request For Comments 1424, Redwood City, CA: RSA Laboratories, February 1993.

[Kantor86] Brian Kantor, Phil Lapsley. *Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News* Request For Comments 977, San Diego, CA: University of California, San Diego, February 1986.

[Moore88] J. Moore. *"Protocol Failures in Cryptosystems" Proceedings of the IEEE*, Vol. 76, No. 5, Pg. 597, May 1988.

[PKCS91] Public Key Crypto Systems #1. *RSA Encryption Standard, Version 1.4* Redwood City, CA: RSA Data Security, Inc., June 1991.

[Rivest92] Ronald L. Rivest. *The MD5 Message-Digest Algorithm* Request For Comments 1321, Cambridge, MA: Massachusetts Institute of Technology, April 1992.

[Robshaw93] M. Robshaw, Burton S. Kaliski. *On "Pseudocollisions" in the MD5 Message-Digest Algorithm* Redwood City, CA: RSA Laboratories, April 1993.

[Salz92] Rich Salz. *InterNetNews: USENET transport for Internet sites* USENIX Proceedings, Summer 1992, Pg. 93. Boston, MA: Open Software Foundation, June 8-12, 1992.

[Venema92] Wietse Zweitze Venema. *TCP Wrapper: Network Monitoring, Access Control, and Booby Traps.* Eindhoven University of Technology: Mathematics and Computing Science, July 1992.

[Waitzman88] D. Waitzman, C. Partridge, S. Deering. *Distance Vector Multicast Routing Protocol* Request For Comments 1075, Cambridge, MA: BBN, November 1988.

13. Author Information

Kurt Lidl is a Senior AlterNet Engineer at UUNET Technologies. His current areas of concentration are dialup IP networking, *Mbone* deployment and process automation. Other interests include real-time combat simulation software, distributed computing and authentication systems, and public access Unix systems. Kurt attended the University of Maryland. He can be reached via E-Mail as *lidl@uunet.uu.net.*

Josh Osborne is a General Programmer at UUNET Technologies. His current areas of concentration are process automation and user services implementation. Other interests include real-time combat simulation

software, computer architecture design, kernel device drivers, WWW, fine beers, and computer languages. Prior job experience includes programming coin operated video games, and Unix system administration. Josh attended the University of Maryland. He can be reached via E-Mail as *stripes@uunet.uu.net*.

Joseph Malcolm is an AlterNet Engineer at UUNET Technologies. Interests include message systems, premises wiring plans, distributed environments, and WWW. He attended the University of Maryland. He can be reached via E-Mail as *jmalcolm@uunet.uu.net*.