InterNetNews: Usenet transport for Internet sites

Rich Salz – Open Software Foundation

ABSTRACT

NNTP, the Network News Transfer Protocol, has been labelled the most widely implemented elective protocol in the Internet. The growth of the Internet has meant more sites exchanging NNTP data. While the explosive growth in Usenet traffic places demands on all sites, the goal of fast network access puts particular demands on NNTP hosts.

InterNetNews is an implementation of the Usenet transport layer designed to address this situation. It replaces the standard UNIX server architecture with a single long-running server that handles all incoming connections. It has proven to be quite successful, providing quick and efficient news transfer.

Introduction

Usenet is a distributed bulletin board system, built as a logical network on top of other networks and connections. By design, messages resemble standard Internet electronic mail messages as defined in RFC822 [Crocker82]. The Usenet message format is described in RFC1036 [Adams87]. This defines some additional headers. It also limits the values of some of the standard headers as well as giving some of them special semantics.

Newsgroups are the classification system of Usenet. The required Newsgroups header specifies where a message, or article, should be filed upon reception. Sites are free to carry whatever groups they want. Most sites carry the core set of so-called ''mainstream'' groups. There are currently about 730 of these groups, and one or two new ones is created every week.

Messages generated at a site are sent to the site's ''neighbors'' who process them and relay them to their neighbors, and so on. Sites can be interconnected – indeed, on the Internet, this is quite common. See Figure 1.

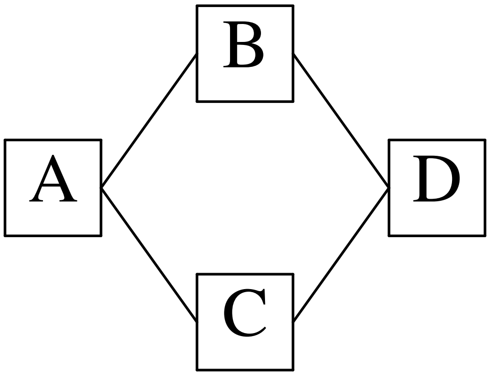

Figure 1: Small Usenet topology (all links are two-way).

The Path header is used to prevent message loops. For example, an article written at *A* could get sent to *B*, *D*, *C*, and then back to *A*. Before propagating an article, a site prepends its own name to the Path header. Before propagating an article to a site, the receiving host checks to make sure that the site that would receive the article does not appear in the Path line. For example, when the article arrived at site *C*, the Path would contain *A!B!D*, so site *C* would know not to send the article to *A*.

Sites also keep a record of the Message-ID's of all articles they currently have. If *D* receives an article from *B*, it will reject the article if *C* offers it later. For self-protection, most sites keep a record of recent articles that they no longer have. This is very useful when another site dumps a (usually quite large) batch of old news back out to Usenet.

For the past few years, the amount of data generated by Usenet sites has been doubling every year. A site that receives all the mainstream groups is receiving over 17 megabytes a day spread out over 11,000 articles [Adams92]. About 20% of the data is article headers, and while all of them must be scanned only half of it is must be processed by the Usenet software.[1]

The number of sites participating in Usenet has been growing almost as quickly. Based on articles his site receives and survey data sent in by participating sites, Brian Reid estimates that there are 36,000 sites with 1.4 million participants [Reid91]. A ''sendsys'' message to the ''inet'' distribution in June of 1989 received about 200 replies in the first twenty-four hours. A year later, nearly 700 replies were received. (Sendsys is a special article that asks all receiving sites to send back an email message, usually without human intervention; by convention, inet is primarily the set of sites on the Internet.)

[1]Yes, this means that, as far as the software is concerned, Usenet is over 90% noise.

The NNTP protocol is defined in Internet RFC 977 [Kantor86] published in February, 1986. This was accompanied by the general public release of a reference implementation, also called ''nntp.'' This has been the only NNTP implementation that is generally available to UNIX sites.

Usenet Software

In addition to InterNetNews, there are two major Usenet packages available for UNIX sites. All three share several common implementation details. A newsgroup name such as comp.foo is mapped to a directory comp/foo within a global spool directory. An article posted to a group is assigned a unique increasing number based on a file called the *active* file. If an article is posted to multiple groups, links are used so that only one copy of the data is kept. A *sys* file contains patterns describing what newsgroups the site wishes to receive, and how articles should be propagated. In most cases, this means that a record of the article is written to a ''batchfile'' that is processed off-line to do the actual sending.

The first Usenet package is called B News, also known as B2.11. The B news model is very simple: the program *rnews* is run to process each incoming article. Locking is used to make sure that only one *rnews* process tries to update the active file and history database. At one site that received over 15,000 articles per day, the locking would often fail so that 10 to 100 duplicates were not uncommon. Because each article is handled by a separate process, it is impossible to pre-calcuate or cache any useful data.

More importantly, file I/O had become a major bottleneck. A site that feeds 10 other sites does over 150,000 open/append/close operations on its batchfiles. It is generally agreed that B news cannot keep up with current Usenet volume; it is no longer being maintained, and its author has said more then once that the software should be considered ''dead.''

C News gets much better performance then B news by processing articles in batches [Collyer87]. The *relaynews* program is run several times a day to process all the articles that have been received since the last run. Since only one *relaynews* program is running, it is not necessary to do fine-grain locking of any of the supporting data files. More importantly, it can keep the entire active and sys file in memory. It can also use buffered I/O on its batchfiles, reducing the amount of system calls by one or two orders of magnitude.

An alpha version of C News was released in October, 1987. Within four years it surpassed B news in popularity, and there are now more sites running C News then ever ran B news.

From the beginning, the NNTP reference implementation was layered on top of the existing Usenet software: an article received from a remote NNTP peer was written to a temporary file and the appropriate *rnews* or *relaynews* program processed it. In order to avoid processing an article the system already has, it first does a lookup on the history database to see if the article exists. It soon became apparent that invoking *relaynews* for every article lost all of C News's speed gain, so the NNTP package was changed to write a set of articles into a batch, and offer the batch to *relaynews*.

When articles arrive faster then *relaynews* can process them, they must be spooled. If two sites (*B* and *C* in the previous examples) both offer a third site (*D*) the same article at the ''same time'' then an extra copy will be spooled, only to be rejected when it is processed, wasting disk space; this problem multiplies as the number of incoming sites increases.[2] To alleviate this problem, most sites run Paul Vixie's *msgidd*, a daemon that keeps a memory-resident list of article Message-ID's offered within the last 24 hours. The NNTP server is modified so that it tells this daemon all of the articles that it is handing to Usenet and queries the daemon before telling the remote site that it needs the article. This is not a perfect solution – if the first, spooled, copy of the article is lost or corrupted, the site will likely never be offered the article after the *msgidd* cache entry has expired. Going further, *msgidd* is work-around for a problem inherent in the current software architecture.

Other problems, while not as severe, lead to the conclusion that a new implementation of Usenet is needed for Internet sites. For example:

- Since all articles are spooled, *relaynews* cannot tell the NNTP server the ultimate disposition of the article, and the server cannot tell its peer at the other end of the wire. This hides transmission problems. For example, a site tracing the communication has no way of finding out an article was rejected because the remote site does not receive that particular set of newsgroups.
- The NNTP reference implementation is showing signs of age. Maintaining the server is becoming a maintenance nightmare; over one-tenth of its 6,800 lines are *#ifdef*-related.
- All articles are written to disk at extra time. Disks are getting bigger, but not faster, while CPU's, memory, and networks are.

InterNetNews architecture

There are four key programs in the InterNetNews package (see Figure 2):

Innd is the principal news server for incoming newsfeeds;

[2] This is quite common for Internet sites, where redundant fast newsfeeds are common and where many Usenet administrators seem to be avid players of the ''exchange news with as many other people as possible'' game.

innxmit reads a file identifying articles and offers them to another site;
ctlinnd sends control commands to *innd*;
nnrpd is an NNTP server oriented for newsreaders.

Of these programs, the most important is *innd*. We first mention enough of its architecture to give a context for the other programs, and then discuss its design and features in more detail at the end of this section.

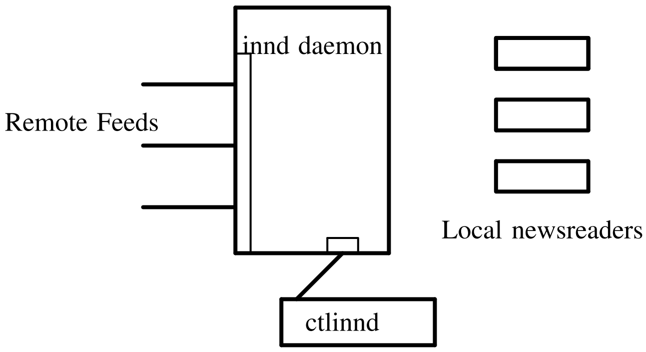

Figure 2: Innd architecture

Innd is a single daemon that receives all incoming articles, files them, and queues them up for propagation. It waits for connections on the NNTP port. When a connection is received, a *getpeername* (2) is done. If the host is not in an access file, then an *nnrpd* process is spawned with the connection tied to its standard input and output.[3] It is worth noting that *nnrpd* is only about 3,500 lines of code, and 20% of them are for the ''POST'' command, used to verify the headers in a user's article. *Nnrpd* provides all NNTP commands except for ''IHAVE'' and an incomplete version of ''NEWNEWS''. On the other hand, it does provide extensions for pattern-matching on an article header and listing exactly what articles are in a group. The NNTP protocol seems to be a good example of the UNIX philosophy: it is small, general, and powerful and can be implemented in a very small program.

Articles are usually forwarded by having *innd* record the article in a ''batchfile'' which is processed by another program. For Internet sites, the *innxmit* program is used to offer articles to the host specified on its command line.[4] The input to *innxmit* is a set of lines containing a pathname to the article and its Message-ID. Since the Message-ID is in the batchfile, *innxmit* does not have to open the article and scan it before offering the article to the remote site. This can give significant savings if the remote site already has a percentage of the articles.

[3]Unlike other implementations, no single INN program implements the entire NNTP protocol.

[4]Other programs, like *nntplink*, are supported but not part of the INN distribution.

Until recently, *innxmit* used *writev* to send its data to the remote host. At start-up it filled a three-element *struct iovec* array with the following elements:

```
[0]   { ".", 1 };
[1]   { placeholder };
[2]   { "\r\n" }
```

To write a line, the *placeholder* was filled in with a pointer to the buffer, and its length, and a single *writev* was done, starting from either element zero or one. While this implementation was clever, and simpler then what was done elsewhere, it was not very fast. *Innxmit* now uses a 16k buffer and only does a *write* when the next line would not fit. This is also consistent with ideas used throughout the rest of INN: use the *read* and *write* system calls, referencing the data out of large buffers while avoiding the copying commonly done by the standard I/O library.

The *ctlinnd* program is used to tell the *innd* server to perform special tasks. It does this by communicating over a UNIX-domain datagram socket. The socket is behind a firewall directory that is mode 770, so that only members of the news administration group can send messages to it. It is a very small program that parses the first parameter in its command line and converts it to an internal command identifier. This identifier and the remaining parameters are sent to *innd* which processes the command, and sends back a formatted reply. For example, the following commands stops the server from accepting any new connections, adds a newsgroup, and then tells it to recompute the list of hosts that are authorized newsfeeds:

```
ctlinnd pause "Clearing out log files"
ctlinnd newgroup comp.sources.unix m \
        vixie@pa.dec.com
ctlinnd reload newsfeeds "Added OSF feed"
ctlinnd go ""
```

The text arguments are sent to *syslog* (8) for audit purposes.

The most commonly-used *ctlinnd* command is ''flush.'' This directs the server to close the batchfile that is open for a site, and is typically used as follows:

```
mv batchfile batchfile.work
ctlinnd flush sitename
innxmit sitename batchfile.work
```

The flush command points out another difference between INN and other Usenet software. The B News *inews* program needed no external locking – files were opened and closed for a very short window, the time needed to process one article. The C News *relaynews* could be running for a longer period of time. The only way to get access to a batchfile is to either lock the entire news system, which is overkill for the desired task, or to rename the file and wait until the original name shows up again. The INN approach is more efficient and conceptually cleaner.

Innd Structure

When *innd* starts up it reads the active file into memory. An array of NEWSGROUP structures is created, one for each newsgroup, that contains the following elements:

```
char  *Name;        /* "comp.sources.unix */
char  *Dir;         /* "comp/sources/unix/" */
long  Last;         /* 0211 */
int   LastWidth;    /* 5 */
char  *LastString;  /* "00211..." */
char  *Rest;        /* "m\n..." */
int   SiteCount;    /* 1 */
SITE  **Sites;      /* defined below */
```

The C comments above show the data that would generated for the following line in the active file:

```
comp.sources.unix 00211 00202 m
```

The *Last* field specifies the name to be given to the next article in the group. The *LastString* element points into the in-memory copy of the file. This number is carefully formatted so that the file can be memory-mapped, or updated with a single *write*.

A hash table into the structure array is built, using a function provided by Chris Torek [Torek91]. The hash calculation is very simple, yet empirically it gives near-uniform distribution. The secondary key is the highest article number, so groups with the most traffic tend to be at the top of the bucket.

The INN equivalent of the *sys* file read next. An array of SITE structures is created, one for each site, that contains the following elements:

```
BOOL  Sendit;
char  FileFlags[10];
```

The *FileFlags* array specifies what information should be written to the site's data stream when it receives an article. The subscription list for the site is then parsed, and for all the newsgroup that it recives, the matching NEWSGROUP structure will contain a pointer to the SITE structure.

Using these two structures it is easy to step through how an article is propagated:

```
extern ARTDATA *art;
extern SITE *Sites, *LastSite;
extern int nSites;
char **pp;
SITE *sp;
NEWSGROUP *ng;
int i;

while (*pp) {
  ng = HashNewsgroup(*pp++);
  if (ng == NULL)
    continue;
  AssignName(ng);
  for (i = 0; i < ng->nSites; i++) {
    if (MeetsSiteCritera(ng->Sites[i], art))
      ng->Sites[i]->Sendit = TRUE;
  }
}

for (sp = Sites; sp < LastSite; sp++) {
  if (!sp->Sendit)
    continue;
  for (p = sp->FileFlags; *p; p++)
    switch (*p) {
    case 'm':
      /* Write Message-ID */
    case 'n':
      /* Write filename */
    . . .
    }
}
```

The ARTDATA structure contains information about the current article such as its size, the host that sent it, and so on. The *MeetsSiteCriteria* function is an abstraction for the in-line tests that are done to see if an article really should be propagated to a site (e.g., checking the Path header as described above). *AssignName* is described below.

At its core, *innd* is an I/O scheduler that makes callbacks when *select*(2) has determined that there is activity on a descriptor. This is encapsulated in the CHANNEL structure, which has the following elements:

```
enum TYPE   Type;
enum STATE  State;
int         fd;
FUNCPTR     Reader;
FUNCPTR     WriteDone;
BUFFER      In;
BUFFER      Out;
```

The *Type* field is used for internal consistency checks. There four different types of channels – local-accept, remote-accept, local-control (used by *ctlinnd*) and NNTP connection. Each type is implemented in anywhere from 100 to 1200 lines of code. The *Reader* and *WriteDone* function pointers, and the *State* enumeration are used for protocol-specific data. For example, *State* field is used by the NNTP channel code to determine whether the site is sending an NNTP command or an article. The *BUFFER* datatype contains sized reusable I/O buffers that grow as needed.

At start time *innd* calls *getdtablesize*(2) to create an array of channels that can be directly indexed by descriptor.

The code to listen on the NNTP port is show in Figure 3. When a host connects to the NNTP port, *select*(2) will report activity on the descriptor and call *RemoteReader* which will accept the connection and possibly create fill in a new CHANNEL out of the resultant descriptor.

It took a bit of effort to write the callback loop so that it was fair – i.e., so that the lowest descriptors did not get priority treatment. The problem was complicated because other parts of the server can add and remove themselves from the *select*(2) read and write mask, as needed.

Once the NNTP channel has been created for a site, the server is ready to accept articles from that site. The reader function for NNTP channels reads as much data as is available from the descriptor. If it is in ''get a command'' state, it looks for a simple \r\n terminator; if it is in ''reading an article'' state, it looks for a ''\r\n.\r\n'' terminator. If not found, it just returns; the data will become available at some point. If the terminator is found, it processes the data in the buffer. For filing an article, this means cleaning up the NNTP text escapes, and calling the article abstraction to process the article.

Processing the article is the largest single routine in the server. The *AssignName* shown above increments the high-water mark for the newsgroup. If the article has already been written to disk, a link is made under the new name. (Symbolic links, if available, can be used if the spool area spans partitions.) If the article has not been written, a *struct iovec* array is set up as shown below, where vertical bars separate each iovec element:

```
First headers...
Path: |LOCAL_PATH_PREFIX|rest of path...
Second headers...
|XREF_HEADER|

|Article body.
```

This is a very fast way of writing the article to disk; it avoids extra memory copies, and is only possible because the entire article is kept in memory until the last moment.

Future work

RFC 977 follows the SMTP protocol for sending text: line are terminated with \r\n, a period is placed before all lines starting with a period, and data is terminated with a line consisting of a single period [Postel82]. *Innd* must scan the text of all articles it receives and convert them to standard UNIX format. On the transmission side, *innxmit* must read the articles a line at a time in order to add the extra data. If all newsreading is done via NNTP, then articles could be stored directly in NNTP format, and *innxmit* could read and write the article in two system calls. The *innd* gains would not be as dramatic, but tests show it would still be somewhat measurable.

There is no NNTP ''TURN'' command, so that a single connection cannot be used for bidirectional article transfer. Turn-around is very successful on UUCP over conventional phone lines, but seems of limited use on higher-performance network links. The SMTP protocol has had a ''TURN'' command since its inception, but it has received no practical use. Several people find the idea of adding outgoing transfer to *innd* attractive, since it is already structured for multi-host I/O and the idea of caching

```
int
RemoteReader(cp)
    CHANNEL *cp;
{
    int newfd;
    struct sockadr_in remote;
    int remsize;

    newfd = accept(cp->fd, &remote, &remsize);
    if (InAccessFile(remsocket))
        CHANcreate(newfd, TYPEnntp, STATEgetcmd, NCreader, NCwritedone);
    else {
        ForkAndExec("nnrpd", newfd);
        close(newfd);
    }
}

int
RemoteSetup()
{
    int fd;

    fd = GetSocketBoundToNNTPPort();
    CHANcreate(fd, TYPEremote, STATElisten, RemoteReader, NULL);
}
```

Figure 3: Listening on an NNTP port

recent articles in memory has its appeal. Adding outgoing transfer to *innd* would take a moderate effort.

Conclusions and Comparisons

The InterNetNews architecture works. Profiling a production installation for 24 hours showed that *open*(2) accounted for 10% of the run time. Since the server only does one *open*(2) per article, it is not clear if any other performance tuning is needed. The profiling overhead accounted for 5% of the runtime.

Several optimizations are available because there is only one process, and because it is always running. For example, avoiding duplicates is an integral part of the server. If a second site offers an article while a first site is sending it, the NNTP code will put the channel to ''sleep'' for a short while before replying to the second offer. This is usually enough time to have the first site finish sending the article, reducing the number of duplicates from hundreds to nearly none, with no external programs.

Since the server is always running, the system has a much smoother performance curve. As a result, it ''feels'' much faster to users.

Another unexpected benefit is that articles are accepted or rejected synchronously. A user can post an article, and by the time their posting agent has returned, it has been written to the spool directory and queued for remote transfer. If there is a problem such as having an illegal newsgroup specified, the user founds out immediately.

The design of the server seems to be very good, split into abstractions that are very independant. For example, sites have no knowledge of incomming NNTP connections. Using callbacks lets each portion of the server safely do I/O without worrying that it might affect other parts. Much of the Usenet processing becomes trivial when serialized, such as access to the history file.

The design has also led to a fairly small program: it is under 13,000 lines, and about 20% of them are comments. This compares favorable to the 7,400 lines in the equivalent C News program and the 7,600 lines in the NNTP reference implementation.

Availability

The INN package is freely redistributable, and is available for anonymous FTP from ftp.uu.net as ~ftp/news/inn.tar.Z. It is discussed in the Usenet newsgroups news.software.nntp and news.software.b.

References

[Adams87] Rick Adams, Mark Horton, *Standard for Interchange of USENET Messages*, Request For Comments 1036, Marina del Rey, CA: Information Sciences Institute, 1987.

[Adams92] Rick Adams, *Total traffic through uunet for the last 2 weeks*, Usenet message <<1992Apr8.193050.8963@uunet.uu.net> in *news.lists*, April, 1992.

[Collyer87] Geoff Collyer and Henry Spencer, *News Need not be Slow*, Usenix Winter Conference, 1987.

[Crocker82] David H. Crocker, *Standard for the Format of ARPA Internet Text Messages*, Request For Comments 822, Marina del Rey, CA: Information Sciences Institute, 1982.

[Kantor86] Brian Kantor, Phil Lapsley, *Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News*, Request for Comments 977, Marina del Rey, CA: Information Sciences Institute, 1986.

[Postel82] Jonathan B. Postel, *Simple Mail Transfer Protocol*, Request For Comments 821, Marina del Rey, CA: Information Sciences Institute, 1982.

[Reid91] Brian Reid, *Usenet Readership Summary Report for May 91*, Usenet message <1991Jun2.141124.12753@pa.dec.com> in *news.lists*, June, 1991.

[Torek91] Chris Torek, *Hash function for text in C*, Usenet message <27038@mimsy.umd.edu> in *comp.lang.c*, October, 1990.

Author Information

Rich Salz is a Senior Software Engineer at the Open Software Foundation, where is a member of the DCE group. His current areas of concentration are RPC and the distributed time service. He joined OSF after working at BBN for nearly five years, working on the Cronus Distributed Programming Environment. Rich attended MIT. He can be reached via U.S. Mail at Open Software Foundation; 11 Cambridge Center; Cambridge, MA 02142. Reach him electronically at rsalz@osf.org.