Opening The Source Repository With Anonymous CVS

Charles D. Cranor
AT&T Labs-Research
Florham Park, New Jersey, USA
chuck@research.att.com

Theo de Raadt
The OpenBSD Project
Calgary, Alberta, Canada
deraadt@openbsd.org

Abstract

Anonymous CVS is an advanced source file distribution mechanism we created to allow *open source* software projects to distribute source code and information about code to Internet users. Built on top of the Concurrent Versions System (CVS) revision control system, Anonymous CVS safely allows anonymous read-only access to a CVS source repository. Prior to the introduction of Anonymous CVS, access to a CVS repository had to be restricted to a select group of privileged software developers. The advantage of open source software is that it promotes reliability and quality by allowing independent peer review and rapid evolution of source code. By introducing Anonymous CVS, we have extended the concept of open source software projects to *open source repository* projects. Having an open source repository allows users to take a more active role in the debugging and development of open source projects. In this paper we will examine and compare the mechanisms used by open source projects to distribute source code. We will present the design and implementation of the first Anonymous CVS server (used to distribute the OpenBSD operating system). We will explain some of our concerns (e.g., security) and some of the problems we faced when trying to adapt CVS for anonymous use. We also will present other more recent source file distribution mechanisms that make use of an open CVS repository. Anonymous CVS is currently bring used by a number of projects including OpenBSD, FreeBSD, Mozilla, Ecgs, Gnome, Python, and GNUstep.

1 Introduction

Over the past few years open source software has made significant inroads in the mainstream software world [7]. The popularity of open source operating systems such as Linux [12] and BSD [5] has generated great interest in the open source development software model. The key attributes of the open source model are that the source code for a software project is freely available (usually over the Internet) and that the code's license guarantees the right to read, redistribute, modify, and use the code freely. The advantage of open source software over closed proprietary software is that it promotes software reliability and quality by supporting independent peer review and rapid evolution of source code. Anyone on the Internet can download, examine, enhance, or debug an open source program. This enables an open source project to have a large Internet-based international developer community that is constantly working on improving the project.

While all users benefit from the open source model, only a relatively few users take advantage of having access to the source code. In fact, most users of open source programs install pre-compiled versions of programs from CDROM distributions or the Internet and never bother to either download, inspect, or modify the source code. The few users who do deal directly with the source code are usually open source developers. These developers have special needs that are only partly met by projects that that fit the standard definition of "open source" [7]. For example, in addition to having access to current snapshot of a project's source code, it is also useful to have access to older versions of source files, annotated per-file revision logs (GNU-style "ChangeLog" files are a poor substitute for this), and the ability to set the files in a source tree to a specific date or release. It is also useful to be able to update a source tree to the latest version without having to download the entire tree while preserving local changes. Historically, revision control systems such as the Source Code Control System (SCCS) [9] and the Revision Control System (RCS) [11] have provided some of these features on a local basis.

RCS and SCCS were designed to manage small-scale projects with a centralized set of developers thus they are not well-suited for large Internet-based open source software projects. Neither system has an Internet server function that allows developers to check out a working copy of a source tree on their local systems, make modifications to it, and then merge those changes back into the main repository. The introduction of the RCS-based Concurrent Versions System (CVS) [1] revision control system addresses these issues. It allows large source trees to be managed as a group under RCS, and it has a network server mode that allows developers to be distributed across the Internet and yet share the same CVS source repository.

Prior to our work on Anonymous CVS, in order to be able to use CVS to access a source repository one had to have an account on the machine hosting the repository. Furthermore, the account had to have write access to the RCS files in the repository. Thus, open source projects that used CVS to manage their source trees had to restrict access to their repositories to a select group of privileged software developers in order to protect themselves from malicious attacks on their source tree. An unfortunate side effect of this was non-privileged users and developers could not access the CVS-based source tree and thus were locked out from the information contained in it. Denying users access to this information runs counter to the open source philosophy and reduces the effectiveness of the open source development model by making it more difficult for non-privileged users to download, debug, and manage their source trees.

In the Fall of 1995 when we started our own open source operating system project called OpenBSD, we decided to use CVS to manage the OpenBSD source tree. Based on our experiences with the previous open source project we were involved with, we recognized the inherent conflict between trying to maintain an open environment while maintaining a private CVS source repository that only privileged users could access. To resolve this conflict we created Anonymous CVS — a mechanism that lets anonymous Internet users access a source repository without compromising its security. Anonymous CVS evolves the open source concept to the next level: *open source repository*. The advantage of open source repository projects over plain open source projects is that it puts the information and power contained in a CVS-based source repository in the hands of the average developer. With Anonymous CVS the revisions, histories, and branches of a CVS tree are public. Anonymous CVS makes it easy to keep a large source tree up to date, even over a slow-speed modem link. The OpenBSD project even ships its CDROM with a checked out CVS tree so that OpenBSD users who are interested in using CVS can start right away without having to download the whole tree from scratch. Anonymous CVS also acts as a training ground by allowing developers new to the project and/or CVS to safely get experience with the `cvs` command before being given write access to the repository. We believe that once developers have had experience with an open source repository project they will find the development environment offered by a plain open source project to be inadequate.

In this paper we examine the issue of source code distribution for open source projects. In Section 2 we examine non-CVS based distribution mechanisms used (for the most part) prior to the introduction of Anonymous CVS. In Section 3 we describe the design and implementation the first Anonymous CVS servers, including issues such CVS limitations, file locking, and security. In Section 4 we present other CVS-based open source repository distribution tools that were introduced after introduction of Anonymous CVS. Finally, in Section 5 we close by providing pointers to the source code of the currently available open source repository distribution tools and also a list of open source repository projects and their respective CVS servers.

2 Traditional Distribution Mechanisms

Traditionally, open source projects have distributed their source code through a number of non-CVS based mechanisms including USENET `comp.sources` newsgroups, anonymous FTP, web, and SUP. Recently projects have also started using Rsync and CTM for source distribution. While each of these mechanisms are useful for distributing code, they do not address the issue of distributing the types of meta information available in a CVS source repository. In this section we examine each of these mechanisms in more detail.

In the 1980s and early 1990s the moderated USENET `comp.sources` newsgroups were a popular way to distribute open source code. To submit a program, an author e-mailed the source code to the moderator of the appropriate USENET group. The moderator would then compile and test the code, and if the program functioned properly post it to the newsgroup as a series of articles. As the postings worked their way through the network, users would collect them and unpack, compile, and install the program. As the program evolved, the author of the program could forward patches to the moderator to test and post to the newsgroup. There are several problems that make these USENET newsgroups a less than ideal forum for the distribution of open source code. First, the group moderator is a bottleneck. Postings can be delayed weeks or even months awaiting the moderators attention. This does not mesh well with the rapid development environment associated with open source projects. While it is possible to have an unmoderated source newsgroup, it is not practical due to abundance of non-uniform and non-source postings (e.g., see `alt.sources`). Second, moderating an active source newsgroup is hard work and it is difficult to find volunteers to perform this thankless task. Third, handling multipart source postings is irritating for users since they must collect all the parts (tracking down parts that are missing) and then assemble them together. Given the abundance of Internet connectivity, it is often easier to just FTP the sources rather than try and collect them from USENET. Thus, it is not surprising that the USENET source newsgroups are now mostly inactive.

Anonymous FTP and web servers are popular ways to distribute both binary and source code. Archiving a collection of tar, zip, or RPM files containing snapshots of a project's source code allows users to conveniently access programs on demand through the Internet. Web servers have the additional advantage of being able to include explanatory information intermixed with hypertext links to source distribution files. There are a number of disadvantages to this type of distribution mechanism. First, it forces developers to break their distribution up into periodic releases. If there is a large amount of time between releases then there is a large delay between when changes are made and when they get distributed to developers on the Internet. If the amount of time between releases is short, then the FTP or web site becomes crowded with numerous release archive files, patch files, or both. If the distribution is large, then downloading new releases becomes painful for developers who are attached to the Internet via slow modem links because in order to stay current new releases must be constantly downloaded. If patch files are used, then developers have the added overhead of downloading and applying the patches. Finally, old releases are often removed from the FTP or web server in order to conserve disk space. This makes it difficult to retrieve and compare old versions of a distribution with new versions. The Linux kernel and GNU programs have traditionally been distributed through these mechanisms.

Another way to distribute code is through a Software Upgrade Protocol (SUP) server [10]. SUP servers operate by tracking the modification times of a collection of source files. SUP clients track the time they were last run successfully. When a SUP client is run it connects to a server and asks for files that have changed since the last successful run. The SUP server checks its timestamp database and delivers only those files. The SUP server can run the files through a compression program to reduce the bandwidth required to update a source tree. The advantage of SUP is that only the files that have changed are downloaded. The disadvantage of SUP is that local changes to source files are not preserved and entire files must be downloaded when they are changed. Also, SUP does not supply any revision information or allow older versions of files to be accessed. Both CMU and the BSD open source operating systems projects have made extensive use of SUP to distribute source files.

The Rsync distribution program performs a similar function to SUP, but in a more efficient way [13]. In SUP when a file is updated the entire file is transfered, however in Rsync only the changes are sent. Rather than using a timestamp database, Rsync simply compares the timestamps and sizes of the source and target versions of a file. If there is a match then the file is not transfered. On the other hand, if the file does not match, then Rsync performs a rolling checksum over the file to determine where changes have been made. Rsync uses the results of this checksum to generate the differences between the source and target versions of the file. The advantage of this approach is that it has lower bandwidth requirements because only the changes are transferred, and the rolling checksum algorithm eliminates the need to have both versions of a file on the server in order to generate a diff. However, Rsync still has the limitation that it does not preserve local modifications to source files, and it does not provide access to older versions of source files or access to the types of meta information stored in a CVS repository.

CTM ("Current Through e-Mail") is another software distribution mechanism that transfers only the changes made to a collection of files rather than entire files [4]. CTM was designed to use the electronic mail as a data transport mechanism. CTM operates by comparing an old and new source tree and generating the differences between them. The diffs are broken up into e-mail sized chunks and mailed to a mailing list. CTM users collect the diffs from the mailing list and apply them to their local source tree by using the CTM client program. The main advantage of CTM is that it does not require IP connectivity in order to use, but compared to CVS-based mechanisms it is still limited.

3 Anonymous CVS Design and Implementation

CVS can be used to manage the source files of a source tree. The source files are stored as a collection of RCS control files called the CVS repository. Developers check out working, fully writable versions of a source tree, make modifications to the files, and check the changes back into the repository. CVS can also merge in changes committed by other users into a local repository, display commit log messages, check out specific branches or dated versions of a source tree, annotate each line of a source file with the revision and author of that line, and update a source tree by transmitting only a compressed version of the changes made to a file. Thus, CVS provides a more powerful and useful abstraction for open source developers than any of the software distribution mechanisms described in the previous section. However, prior to the introduction of Anonymous CVS, CVS had a major limitation for open source projects: an account with write access to the CVS source repository was required in order to use CVS. One of our goals in creating Anonymous CVS was to allow greater access to the OpenBSD project's CVS source repository in order to have a more open project and to encourage developer interest. We wanted to allow anyone on the Internet to safely have anonymous[1] read-only access to our CVS repository — a practice that was unheard of at the time.

3.1 Anonymous CVS Goals

As we were designing OpenBSD's Anonymous CVS service, we had the following three goals in mind:

Security: While we wanted to allow the world to have read access to our CVS repository, we did not wish to allow anonymous *write* access to it. Thus we had to ensure that our Anonymous CVS system did not compromise the security of our source repository.

Efficiency: CVS server operations are known for being resource intensive. While we were eager to provide anonymous access to our repository, we did not want to do so at the expense of bogging down our CVS machine. Thus we had to ensure that Anonymous CVS did not place an undue burdon on our CVS system.

Convenience: If an Anonymous CVS service is difficult to access then no one will use it. Thus we designed our Anonymous CVS system to be as easy and convenient to use as possible. For systems with CVS installed, accessing our CVS repository is as easy as setting an environment variable and running CVS. No usernames, passwords, or special programs (other than CVS itself) are required to use Anonymous CVS.

3.2 Anonymous CVS Design

Based on our three goals we decided that Anonymous CVS service should be offered from a machine other than our main CVS server system. This provides security by keeping all anonymous connections off the main CVS server. The main server need only distribute a copy of its CVS-controlled RCS files to the anonymous system using a standard technique such as SUP. It does not have to trust the anonymous server system beyond that. This also provides efficiency by keeping Anonymous CVS server and networking load off the main server machine. This is important because our main CVS server (`cvs.openbsd.org`) is connected to the Internet by a low-bandwidth ISDN link. Figure 1 shows the relationship between the main OpenBSD CVS server and the primary OpenBSD Anonymous CVS server (`anoncvs.openbsd.org`). To access the Anonymous CVS service, users simply set their `CVSROOT` environment variable to `anoncvs@anoncvs.openbsd.org` and run CVS commands normally. The Anonymous CVS server will reject any attempt to modify its local copy of the CVS repository.

We also secured the environment on the Anonymous CVS server in order to prevent malicious tampering with CVS service. Anonymous CVS is accessed through the special account "`anoncvs`." While this account has no password (thus allowing anyone to log into it), it also has a special anoncvs shell that restricts what it can run to a single command: "`cvs server`." Any attempt to run a command other than the CVS server results in the anoncvs shell printing an error message and exiting. When the anoncvs shell receives a request to run the CVS server it uses the `chroot` system call to restrict access to the server to a sandboxed environment. In order to use `chroot` the anoncvs shell must be setuid "root." While this is not optimal, we note that the anoncvs shell is a small program that immediately drops privileges as soon as it uses `chroot`. We feel that the gains of using a restricted root environment are worth the risks of having a small setuid program. A partial listing of the anoncvs shell is shown in Figure 2.

The only files that reside in the `chroot` environment are the commands necessary to run CVS (the `cvs` binary and helper RCS commands) and the read-only copy of the CVS repository from the main CVS server. Note that CVS version 1.10 and later versions access RCS files directly and thus the RCS helper commands no longer need to be in the sandbox area. The copy of the CVS repository in the sandbox is owned by a user other than the anoncvs user to prevent any chance of an anonymous user writing to it. The only writable directory in the sandbox environment is `/tmp` which is required for CVS to operate properly. Also note that there are no setuid files in the sandbox. Thus, if an anonymous user was to break out of the CVS server (e.g., through a buffer overflow) it would be very difficult to do any damage other than interfere with other CVS server processes running under the anoncvs account. The possibility of such interference could be avoided by allowing the anoncvs shell to randomly distribute its UID among a specific range of UIDs reserved for anonymous access.

The main advantage of using an anoncvs shell rather

[1] By "anonymous" we mean that resources can be accessed without authenticating the user (as in anonymous FTP). Achieving truly anonymous access is a more difficult problem that was beyond the scope of our needs. A more anonymous access mechanism could be achieved by borrowing ideas from a system such as Crowds [8].

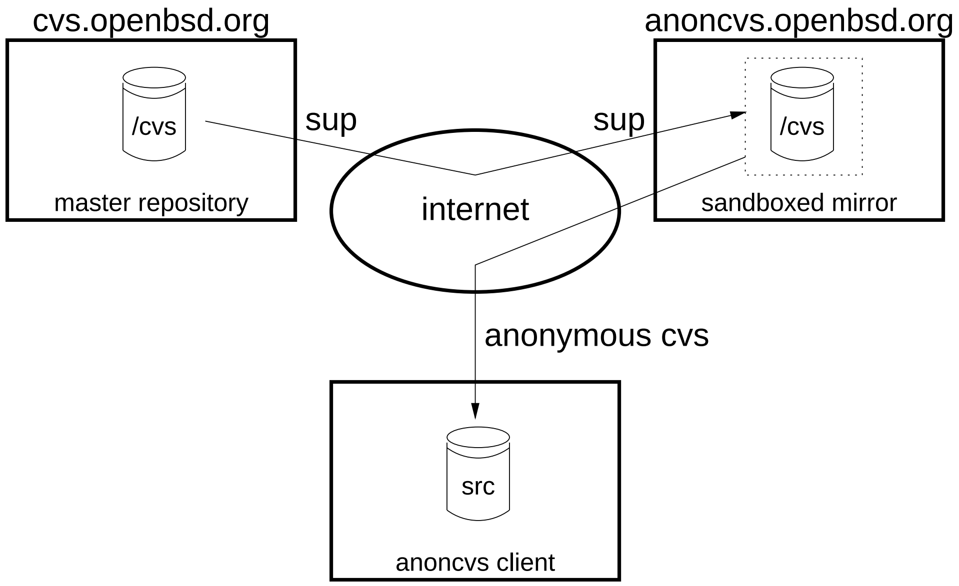

Figure 1: OpenBSD's Anonymous CVS service. The CVS repository is mirrored within the sandbox using SUP (Rsync could also be used).

than a specialized server program is that it integrates nicely with CVS's server system and can be used with standard login programs such as rsh and ssh. Many Anonymous CVS servers disallow rsh access and allow only ssh access for added security. For ssh, a non-standard server port such as 2022 can be used in addition to the standard port to make Anonymous CVS more firewall friendly. This allows users to work around poorly thought-out firewalls that are configured to block all unknown traffic in the reserved TCP port range.

Note that an Anonymous CVS server is more secure than most Anonymous FTP servers. This is because standard FTP servers never revoke their root access. Instead, they just swap UIDs when accessing files. On the other hand, the anoncvs shell permanently revokes root access before running the CVS server.

3.3 Anonymous CVS Implementation Issues

As we implemented Anonymous CVS, we encountered three issues relating to CVS that caused us some concern. First, we discovered that CVS did not run properly without write access to its log file. Since CVS itself is an open source program, we fixed this problem by adding the `CVSREADONLYFS` environment variable to CVS. If set, CVS ignores this error.

Our second concern with Anonymous CVS was how it interacted with CVS's file locking protocol. CVS controls access to RCS files by creating lock files in the CVS repository area. Under Anonymous CVS this is not possible since the Anonymous CVS server runs under a UID that does not have write access to the repository. To address this issue we disabled file locking for read-only access. Since commits are not allowed in the Anonymous CVS server's copy of the repository this is not a problem. However, another possible problem is that the Anonymous CVS server may encounter a partially complete RCS file in its copy of the repository. We examined the CVS documentation and source code and determined this was unlikely for the following two reasons. First, CVS on the main server updates its RCS files in one operation by creating a temporary RCS file, modifying it, and finally renaming it to the RCS file. Since the `rename` system call is atomic, there is no chance of the mechanism used to transfer RCS files from the main server to the anonymous server encountering an incomplete RCS file. Second, we use SUP to transfer our RCS files from the main server to the anonymous server. When SUP installs an updated file it uses the same atomic-rename technique that the CVS server uses to install a new file. This prevents the CVS servers running on the anonymous server from seeing an incomplete RCS file. One possible problem that could be encountered is if CVS reads a list of RCS files currently in the mirrored repository and SUP deletes one of those RCS files before CVS has a chance to open it. In reality the odds of this happening are very low because RCS files typically do not get removed from a repository. Of course when multiple files are being updated on the mas-

```
/* location of CVS tree relative to anonymous CVS user's home directory */
#define LOCALROOT       "/cvs"

/* remote hostname */
#define HOSTNAME        "anoncvs@anoncvs1.usa.openbsd.org"

/* cvs root */
#define CVSROOT         __CONCAT3(HOSTNAME,":",LOCALROOT)

/* default environment */
char * const env[] = {
    "PATH=/bin:/usr/bin", "SHELL=/bin/sh",
    __CONCAT("CVSROOT=",LOCALROOT),
    "HOME=/", "CVSREADONLYFS=1",
    NULL
};

int main(argc, argv)

int argc;
char *argv[];

{
    struct passwd *pw;

    pw = getpwuid(getuid());
    if (pw == NULL || pw->pw_dir == NULL)
        errx(1, "no user/dir for uid 0", getuid());

    setuid(0);
    if (chroot(pw->pw_dir) == -1)
        errx(1, "chroot");
    chdir("/");
    setuid(pw->pw_uid);

    /* program now "safe" in sandbox with root privs dropped */
    if (argc != 3 || strcmp("anoncvssh",  argv[0]) != 0 ||
        strcmp("-c", argv[1]) != 0 || (strcmp("cvs server", argv[2]) != 0 &&
          strcmp(__CONCAT3("cvs -d ",LOCALROOT," server"), argv[2]) != 0)) {
            fprintf(stderr, "\nTo use anonymous CVS install the latest ");
            fprintf(stderr,"version of CVS on your local machine.\n");
            fprintf(stderr,"Then set your CVSROOT environment variable ");
            fprintf(stderr,"to the following value:\n");
            fprintf(stderr,"\t\n\n", CVSROOT);
            sleep(10);
            exit(0);
    }
    execle("/usr/bin/cvs", "cvs", "server", NULL, env);
    perror("execle: cvs");
    fprintf(stderr, "unable to exec CVS server!\n");
    exit(1);
}
```

Figure 2: Partial listing of anoncvs shell program

ter CVS server there is still a chance that an anonymous user will end up fetching some old file and some new files from the group of files being updated. Indeed, this is even a problem for normal CVS users because updates are often checked in multiple chunks. However, in practice we have found that this problem does not occur that often. In the future it may be useful to determine ways to extend parts of CVS's file locking to anonymous servers. It would also be useful to create a mechanism where CVS updates on the master server are pushed to the slave anonymous servers as soon as they happen.

The third issue relating to CVS that caused us concern was CVS's poor handling of network flow control. CVS's server function was designed to run in a high-bandwidth network environment with a relatively small source repository. This environment is fundamentality incompatible with our target environment. The OpenBSD source tree consists of 250MB of source files, and we distribute it to many anonymous users connected to the Internet via slow speed PPP links. We found that CVS did not run well in this environment because it was designed to minimize the amount of time it holds a lock on a directory in a repository. In order to do this, when checking out source code the CVS server splits into two processes. The first process walks the CVS repository's directory tree as fast as possible performing the requested action. The second process buffers the output from the first process in its memory and sends it out over the network connection. The second process uses non-blocking I/O to ensure that it does not block on a slow network connection. This allows the first process to run to completion without blocking on full network I/O buffers while holding a lock on a repository directory. The problem with this design is that the CVS developers did not put a limit on the amount of data the second process was willing to buffer. The result of this is that for a large checkout over a slow link the second process can grow and consume large chunks of virtual memory. We found that if multiple Anonymous CVS servers were running at the same time they quite often exhausted all available virtual memory on our Anonymous CVS server machine thus creating a denial of service. This problem was especially annoying since locking is not an issue with a read-only CVS repository.

To fix this problem, we modified CVS to limit the amount of data the second process can buffer. In our environment it is better to let the first process block than to run our server out of virtual memory. Partly due to our complaints about the behavior of CVS in this case, the maintainers of CVS modified it to address this issue. Their fix was to modify the first process in a CVS checkout to be non-blocking only on a per-directory basis. This allows the second process to catch up to the first after the first has completed a directory. The advantage of this fix is that it minimizes the time a CVS directory lock can be held. There is still potential for problems if CVS encounters a single directory with a large number of modified files. In this case it is still possible for the CVS server to use a significant chunks of system virtual memory. However, as most source files in large sources trees are distributed among several directories this should not be a problem.

One remaining unsolved issue is the fact that CVS requires a writable `/tmp` directory in order to function. For better security we would like for an Anonymous CVS server to be able to function without any write access to the filesystem in the `chroot` environment in which it operates.

4 Other CVS-based Distribution Mechanisms

As open source repository projects became more widespread, several new tools including CVS's Pserver, CVSWeb, and CVSup were developed to take advantage of this powerful new environment.

CVS's Pserver was created by the CVS development team partly in response to the demand for anonymous support within CVS itself. Rather than use the standard CVS server with the anoncvs shell that we created, CVS's Pserver bypasses rsh/ssh and listens on its own TCP port for connections. Pserver's user interface requires the use of a login and password (even for anonymous access – an annoying inconvenience for users getting started with Anonymous CVS), and it transmits this data over the wire in clear-text. Pserver often does not operate in a `chroot` environment, and thus it is more of a security risk than our version of Anonymous CVS. It is possible to run Pserver in a `chroot` environment, but it requires more files to be added to the sandbox environment in order for Pserver to authenticate the user, especially on systems that support complex user authentication mechanisms like PAM [3]. Pserver, unlike Anonymous CVS, also does not fully give up root privileges if it has them. In the context of anonymous access, the main advantage of Pserver is that it is included with the main CVS distribution.

The CVSWeb system was developed by Bill Fenner of the FreeBSD project to allow anonymous access to a CVS repository through a standard web browser [2]. Although CVSWeb cannot be used in the same way as Anonymous CVS to update a local source tree, the big advantage of CVSWeb is that it allows anyone with a web browser to easily browse the content of a CVS repository using a graphical user interface. This can often be more convenient than using the standard CVS interface.

Tool	Location
SUP	ftp://ftp.openbsd.org/pub/OpenBSD/src/usr.bin/sup
Rsync	http://samba.anu.edu.au/rsync/
CTM	http://www.freebsd.org/handbook/synching.html#CTM
CVS (includes Pserver)	ftp://ftp.gnu.org/gnu/cvs
Anonymous CVS	/anoncvs.shar
CVSWeb	http://www.freebsd.org/~fenner/cvsweb/
CVSup	http://www.polstra.com/projects/freeware/CVSup/

Table 1: Source distribution tools

Project	Information Pointer
CMU Common Lisp	http://www3.cons.org/cmucl/
Ecgs	http://egcs.cygnus.com/cvs.html
FreeBSD	http://www.freebsd.org/handbook/synching.html#ANONCVS
Gnome	http://www.tw.gnome.org/software/anoncvs.shtml
GNUstep	http://www.gnustep.org/resources/Anoncvs.txt
Guile	http://www.red-bean.com/guile/guile-anon-cvs.html
Mozilla	http://www.mozilla.org/cvs.html
Obtuse	http://www.obtuse.com/open_source/
OpenBSD	/anoncvs.html
OpenLDAP	http://www.openldap.org/software/repo.html
Python	http://www.python.org/download/cvs.html
Quinn Diff	http://quinn-diff.nocrew.org/anoncvs.html
Sudo	http://www.courtesan.com/sudo/anoncvs.html

Table 2: Open source repository projects on the Internet

The current state of the art in open source repository source distribution tools is John Polstra's CVSup package [6]. CVSup is an efficient and flexible file distribution system. CVSup's efficiency is due to two factors. First, the control protocol used by CVSup streams multiple requests between client and server rather than making the client wait for a request to be satisfied before issuing the next request. This helps CVSup make the most of available network bandwidth. Second, CVSup takes advantage of knowledge of the internal formats of certain types of files to reduce the overhead of sending an update. CVSup knows the format of RCS files, CVS repositories, and append-only log files. CVSup can use this knowledge to easily extract the minimal amount of data necessary to send changes from these types of files over the network (the data can optionally be compressed before being transmitted). For files whose format CVSup does not understand, CVSup uses the Rsync algorithm. CVSup includes both a command line and GUI interface.

CVSup has two features that are especially useful for accessing CVS repositories. First, the CVSup client program can be used to request a specific version of a source tree. The version can be specified by date or by symbolic name. Second, CVSup can be used to download changes from a master CVS repository and merge them directly into a local CVS repository[2]. This allows developers to maintain their local changes within a private branch of their copy of the master CVS repository. In order to achieve the same effect with traditional Anonymous CVS, one would have go through the time consuming process of checking out a clean version of the master source tree (via Anonymous CVS) and then importing it into the vendor branch of a local CVS repository. CVSup can do the same job with much less overhead.

There are two drawbacks to CVSup. First, it is difficult to compile and install because it is written in Modula3 rather than C. While there are open source Modula3 environments available, compiling and installing them is a difficult task (especially for unsupported platforms). However, there are precompiled binaries available from the PostgreSQL project[3]. The second drawback of CVSup is that it can only access a set of pre-determined collections of files, while Anonymous CVS can access anywhere from a single file to the entire source tree.

[2] Care must be taken to avoid version number conflicts and deleted RCS files, see the CVSup FAQ for details.

[3] See /pub/CVSup on ftp.postgresql.org

However, if CVSup is used to download a copy of the entire repository, then standard CVS can be used on that repository to access individual files in that repository.

5 Conclusions

In this paper we have examined the issue of distributing the source code of open source projects to Internet developers. Table 1 contains a list of the tools discussed in this paper and pointers to where to get them. We examined the evolution of open source code distribution from early channels such as USENET and anonymous FTP to modern mechanisms such as Anonymous CVS, CVSWeb, and CVSup. Our contribution was the design and implementation of Anonymous CVS. Since the introduction of OpenBSD's Anonymous CVS service many other open source projects have opened up their CVS repositories. Table 2 contains URLs for some of the open source repository projects currently on the Internet. We believe Anonymous CVS has made a significant positive impact in the open source community. Anonymous CVS certainly had a positive impact on OpenBSD. We currently see around 2000 anoncvs transactions per-week on our Canadian-based Anonymous CVS server. Several of our other Anonymous CVS servers report similar usage. We hope to see more open source repository tools and projects appear on the Internet in the future.

References

[1] B. Berliner. CVS II: Parallelizing software development. In *USENIX Conference Proceedings*, pages 341–352. USENIX, 1990.

[2] B. Fenner. CVSWeb. See `www.freebsd.org/~fenner/cvsweb.html`.

[3] Open Group. X/Open single sign-on service (XSSO) - pluggable authentication. See `www.opengroup.org /pubs/catalog/p702.htm`.

[4] P. Kamp. Current through e-mail (CTM). See `www.freebsd.org /handbook/synching.html#CTM`.

[5] M. McKusick, K. Bostic, M. Karels, and J. Quarterman. *The Design and Implementation of the 4.4BSD Operating System*. Addison Wesley, 1996.

[6] J. Polstra. CVSup. See `www.polstra.com /projects/freeware/CVSup/`.

[7] E. Raymond. Open source home page. See `www.opensource.org`.

[8] M. Reiter and A. Rubin. Anonymity loves company: Anonymous web transactions with crowds. *Communications of the ACM*, 42(2):32–48, February 1999.

[9] M. Rochkind. The source code control system. *IEEE Transactions on Software Engineering*, SE-1(4):364–370, December 1975.

[10] S. Shafer. The sup software upgrade protocol. Technical report, Department of Computer Science, Carnegie Mellon University, 1985.

[11] W. Tichy. RCS – a system for version control. *Software – Practice & Experience*, 15(7):637–654, July 1985.

[12] L. Torvalds et al. The Linux operating system. See `www.linux.org`.

[13] A. Tridgell and P. Mackerras. The rsync algorithm. Technical report, Department of Computer Science, Australian National University, 1998.