Project Athena's Release Engineering Tricks

Don Davis, MIT Staff

E40-319, MIT, Cambridge, MA 02139

ABSTRACT

This paper discusses Project Athena's approach to preparing software releases. As a large, centrally-administered UNIX network, Athena has had to solve some difficult software-management problems. Athena's Release Engineering Group relies on *easily-taught* work-disciplines and procedures, and on a *few* high-leverage tools, so as to avoid cultivating the usual reliance on release-building expertise.

Introduction

Project Athena is the largest *centrally-administered* workstation network in the world; we have 850 cpu's: 600 μVAXen, 200 IBM RT/PCs, and 50 VAX servers, all on 23 Ethernets and a Proteon proNET-10 fiber-optic spine. We've merged features of 4.3BSD, IBM's AOS 4.3, DEC's ULTRIX 2.0, and Sun's NFS, and have added many features of our own, including X Window System, Kerberos authentication, Hesiod name service, Zephyr notification service, and the Moira Service Management System.[1] Almost all of our code runs without modification on both VAXen and RTs. Our additions comprise ~20M of source-code. We will soon begin to evaluate IBM 8570 (386-based PS/2) and DECstation 3100 (PMAX) workstations, and are evaluating others now. We have about 20 full-time system programmers overall.

Of Athena's workstations, 600 are installed in public areas. Almost all of our users are highly-pressured, computer-naive undergraduate students. We currently have about 11,000 users; 5,000 of them login at least once per week. Our system serves about 3,000 students each day. Multiply these numbers by the fact that most of these people never get enough sleep, and you can see that we *suffer* when the system breaks!

Central administration of such a large net is difficult; indeed, most of our in-house development has been invested in the aforementioned system-administration tools. We at MIT *had* to make this investment, because our users can't be expected to help keep the system alive. Thus, our most carefully guarded resource is not the spine-bandwidth of 10 Mbit/sec, but our 5 Operations system-programmers. Consider that figure: five people, to run 850 machines! We claim to be able to expand to 10,000 workstations; in fact, we have only 850 for political reasons, not technical ones. In brief, Athena's central constraints are:

- "No shoe leather."
- "No network broadcast."
- "Does it scale well?"
- "The sponsor is always right."

---

[1] G.W. Treese, "Berkeley UNIX on 1000 Workstations: Athena Changes to 4.3BSD." In *USENIX Conference Proceedings* (February 1988), pp. 175-182.

Athena's Release Engineering staff are 3 of the 15 System Development programmers. Our priorities are:

1. providing identical software functionality on all types of µVAXen and RTs;

2. facilitating a centralized administration;

3. stabilizing the system, so that releases can be more invisible, frequent, and automatic;

4. merging the VAX and RT source trees (mostly complete);

5. smoothing the release cycle, so that a new student-hire can turn the crank.

Priority #1 is an ongoing chore, requiring a particular work-discipline. #2 was the motivation for many of Athena's added services. #4 remains an episodic chore, and we expect no relief. Priority #5, "Smoothing the Release Cycle," is the main topic of this article.

Software Releases at Athena

An Athena release consists of two sets (VAX and RT) of 12,000 files, plus update scripts and installation media. We provide educational software separately; it's *not* in the release. We keep a minimum (4M VAX, 6.5M RT) of software on the workstations' local disks. The workstations usually get executables from Remote Virtual Disk (RVD) servers, but may also use NFS or AFS for this purpose.[2] These RVD servers' contents and the workstations are updated around 3-4 times per year; we recompile the clean source tree about once per year. Most workstations are updated automatically, without a human visit. Each workstation initiates its own update asynchronously. The servers' own software update is a modification of the workstation update.

Athena's release-cycle is complicated by several of our system's features:

Common User Interface: We insist that the VAX and RT user-interfaces be as similar as keyboard-differences permit. To achieve this, we've merged 85% of the two hosts' source-trees. New releases from Berkeley, IBM, and Sun get merged anew, at great cost. At release-time, we compare the VAX and RT releases for inode-level differences (uid, gid, type, and mode).

Local Disks: All workstations have local disks for swap, and for always-resident executables: kernel, network code, X server, shells, etc. We're trying to decrease these local root contents, so as to reduce workstations' update-frequency. We'd like to be able to make some releases by updating only the binary-servers, without updating the workstations' roots, but our kernels aren't yet static enough.

Remote Virtual Disk: To ship binaries, we usually use Remote Virtual Disk (RVD) service, which is faster than NFS. RVD is a lower-level protocol than NFS; while NFS-requests access *files*, RVD-requests access *disk-blocks*. We've had to put a lot of work into hiding RVD's grotesque user-interface from our users, but RVD has one redeeming feature: it is very hard to write to an RVD pack from which another user is reading. This is convenient for our central

[2] All of our kernels support both RVD and NFS (Sun Microsystem's Network File System). We're now preparing to add CMU's AFS (Andrew File Service, previously known as VICE).

administration needs, because it prevents any field-modification of binary-servers' contents. Thus, when a user complains of a bug, we don't have to check his binary-server's contents for subnet-specific "fixes;" we know that our master copy of those contents is identical to all others.

Two-Piece Releases: We ship releases in two filesystems: less than 10M of root, /bin, /dev, /etc, and /lib; and 110M of /usr. The local root-contents all come from the smaller pack. We try to keep all of the volatile code in the smaller pack, too, so that the larger one needs less-frequent updates. The small pack is now *too* small for RT releases, so that the 30 RVD servers' disks need to be reformatted.

Service Interdependence: Athena's additions to the UNIX environment (Kerberos, X, Zephyr, Hesiod, Moira) all depend on each others' libraries. Further, several of these subsystems are in active development, so that half of the annual "clean-source build" is taken up with repeated efforts to meld these systems' modifications.

Smoothing the Release-Cycle

To repeat: our first priority here is to reduce Release Engineering's reliance on expertise. To be blunt, experts not only can die, they do quit, and MIT doesn't pay enough to keep them all from quitting. Our goal is to "haze" all new student-hires, by having each one do a release.

Work-Disciplines

The simplest improvement we've made in our release-cycle is to follow some easily-taught work-disciplines:

Source Access: Only Release-Engineering members change the central source-trees. This allows us to ensure that the VAX and RT copies of the sources remain identical.

Specification: Management leads in the construction of the "Release Notes" document, which specifies in advance what is and is not to go into the release. This preparation is a political free-for-all and the published document is thus invaluable as a contract; even better, Release Engineering gets to edit the final draft, once the release is ready for distribution.

One Person/ One Build: We divide the release into two parallel efforts: one person invokes "make clean; make all; make install" in the VAX source-tree, another does the same thing in the RT source-tree, and a third plays gofer for both. The gofer helps by shielding the builders from distractions.

Worklogs and Makelogs: The two builders build **make** logfiles, and keep personal work logs as well. All three parties communicate via a *private* conferencing system.[3] We have found that work logs are crucial, because they record last-minute source changes and *why* they were necessary. RCS logs are necessary, too, but are *not* sufficient, because they are distributed throughout the two 800M source-trees.

---

[3]At Athena, we use **discuss**, an authenticated network conferencing system. See Ken Raeburn, Jon Rochlis, William Sommerfeld, and Stan Zanarotti, "*Discuss*: An Electronic Conferencing System for a Distributed Computing Environment." In *USENIX Conference Proceedings* (Winter 1989).

No Compiler Warnings: We try to make everything compile, load, and install correctly, without warning-messages. This makes it easier to detect new breakage in the makelogs, which run to thousands of lines. This goal was hard to achieve, because **hc**, the IBM C compiler for the RT, offers lots of **lint**-like warnings. To do this, we've had to tweak a lot of source, and most of our 1300 Makefiles.

One Person/ Two Fixes: Finally, the three workers check that each change promised by the Release Notes is in place. Here, they specialize differently: each person ensures that a given change has been installed correctly for *both* host-types, logging his changes as usual. *Only with the invention of this discipline* did Athena meet the goal of "identical software functionality on VAXen and RTs."

Release Tools

Prot_sources is a tool that massages the sources' access permissions every night. This allows builders to start a build at any time, without using root access. **Prot_sources** knows a great deal about source-trees, and is tuned for speed; it traverses our VAX sources (800M) in about 1.5 hours.

The tool's purpose is two-fold: to facilitate grandiose, system-building "make all" and "make clean" invocations, and to touch up security holes. In other words, we want the sources to be protected neither too tightly, nor too loosely, but only "just right." The command-line runs:

prot_sources [-R] [pathname] [pathname] ...

Each pathname argument may be a filename or a directory-name. The default is the current directory. If **-R** is given, any directories in the pathname-list will be scanned recursively. Symbolic links to directories are NOT descended. There are 4 classes of protection:

Classification	*uid.gid*	*mode*
WRITABLE	builder.builders	664
EXECUTABLE	builder.builders	775
LOCKED_SOURCES	*author*.builders	o-w
UNLOCKED_SOURCES	builder.builders	444

Prot_sources classifies and protects the source-tree as follows:

• WRITABLE: symlinks, object-files, vers.c files in kernel source-trees, **lex**- and **yacc**-generated C sources, everything in **config**'s kernel-directories, Makefile~, core, a.out, and other special cases.

• EXECUTABLE: directories, executable files.

• LOCKED_SOURCES: *.[chflys], Makefiles, and their RCS-files, unrecognized names.

• UNLOCKED-SOURCES: *.[chflys], Makefiles, and their RCS-files.

Our use of **prot_sources** has proven very successful, in that our **make** invocations are very seldom derailed by broken permissions.

Track is an **rdist**-like tool for updating filesystems which was developed at BellCore.[4] We ported it to the RT, added some features of our own, and rewrote it pretty completely for speed and maintenance. Compared to **rdist**, **track** offers the following (dis)advantages:

• Net-Performance: **Track** optimizes its use of the network for one-to-many distribution, by "compiling" the exporting filesystem's inodes' contents into a single file, the *statfile*. When an updating host runs **track** for an update, **track** compares the local file system against the *statfile*'s contents. This saves thousands of networked stat() calls.

• Net-Politics: While **rdist** pushes updates at its clients, **Track** offers a "librarian-subscriber" model of file-distribution. That is, **track** allows subscribers to be selective about what they import of the librarian's exports. Also, since a subscriber updates by running **track** himself, he decides when and whether to update.

• Flexibility: **Track** allows the export and/or import of symlinks to the librarian's files and directories. **Track** can update device nodes, so that we run **MAKEDEV** not on our workstations, but on the RVD-packs' master copy of /dev. We use these features to replace a lot of error-prone update-script code. Finally, **track**'s *statfile*-output is designed to be filtered through **awk**, allowing **find**-like selectivity. **Rdist** offers none of these features.

• Bad User-Interface: **Track**'s user-interface is even worse than **rdist**'s. The command-line, *subscription-list* syntax, and overall semantics are all user-hostile. We're rewriting that code now.

We use **track** to update workstations, servers, and even source-trees. Usually, we construct one or more **track** invocations into update-scripts. We also use **awk**'ed *statfile*s to compare different releases for directory and inode-differences, where **diff** would report irrelevant file-content differences. This automated checking of 12,000 files is important, because hand-checking and staff-testing just haven't ever sufficed.

A surprisingly high-leverage trick was to add entry and exit banners to **make**, which enable junior staff to read **make**'s logfiles. We also equipped **make** to read the environment variable MAKEOPT, so that recursive invocations can be forced to print these banners. Before we added these banners, it took a lot of expertise to diagnose and fix a broken build. Remember, we're talking about 1300 Makefiles!

Makesrv is our script-set for converting a standard public-workstation configuration to a server's. By "configuration", here, we mean root-contents; all servers run the same kernel version. The scripts use **track**, so that their maintenance is usually just a matter of changing various *subscription-list*s that the **track** invocations require. **Makesrv** currently configures servers of the following types:

NFS	Hesiod	Kerberos	Moira	On-Line-Consulting
RVD	Zephyr	Discuss	Printer	

[4]Daniel Nachbar, "When Network File Systems Aren't Enough: Automatic File Distribution Revisited." In *USENIX Conference Proceedings* (Summer 1986).

Enduring Problems

Service Interdependence (mentioned above) boils down to two problems: (1) parallel development always incurs a merging cost; (2) new technology has to bootstrap itself. Now, merging and bootstrapping are inherent to the R&D lifestyle; they're not going to go away. It may help, though, to build *all* libraries before we build *any* of our programs, even at the expense of complicating some Makefiles. This would relieve the release-cycle of some interdependence-spawned delays, since we would rebuild less code in each pass through the code-merge cycle.

RT executables are bigger than VAX executables. This comes partly from the RTs more prolix RISC architecture, but mostly comes from printf() calls. RTs are capable of running without hardware support for floating-point, so every program that calls printf() gets linked to a gung-ho floating-point library. Thus, even though all of our RTs have hardware FPPs, and even though very few of the programs use floats, executables bulk 30% larger on the RT than on the VAX. This makes for space problems, wherever RT executables live. Either we have to reformat a lot of disks, or we have to add a dynamic-linking scheme to the RT kernel. Other solutions are possible, but they're even less attractive.

Our developers' handoff to Release Engineering is broken; we'd like to retain control of our central source-trees, without actually installing bug-fixes ourselves. We haven't devised a workable balance between security and laziness. Also, developers can't fully test their own Makefiles for compatibility with our build-procedures, because their development environments are different from ours. For example, developers have their own access-control groups, which are mutually exclusive with Release Engineering's "builders" group. This often leads to permissions-breakage when we try to build their code.

Preparing installation media still requires expertise. Currently, to install (or re-install) a workstation, we use the boot floppies' kernel and scripts to give the workstation remote file access. The workstation then runs a remotely-held script that reformats the hard disk and updates it to the current workstation configuration. The hard part is to prepare a very compact, network-capable configuration that fits on our various removable media.[5] We currently have to hand-craft a kernel and boot-blocks to fit, each time we cut a release. To solve this problem, we're working on a stand-alone **tftp** boot program, which will fit on all of the removable media. At install-time, this primary boot will download a kernel that has a memory-resident root-partition.

Appendix: Trademarks

UNIX is a registered trademark of AT&T. The X Window System is a trademark of MIT. DECstation, PMAX, ULTRIX, VAX, and µVAX are trademarks of Digital Equipment Corporation. AOS, PS/2, RT/PC, and RVD are trademarks of International Business Machines Corporation. Sun and NFS are trademarks of Sun Microsystems. proNET-10 is a trademark of Proteon, Inc.

[5] RT-floppies, RX33's, RX50's, TK50's, and RL02's. The smallest of these allows 350K for file-contents.

Table of Contents