# OpenDist - Incremental Software Distribution

Peter W. Osel and Wilfried Gnsheimer
Siemens AG, Mnchen, Germany

# OpenDist – Incremental Software Distribution

*Peter W. Osel and Wilfried Gänsheimer* – Siemens AG, München, Germany


## ABSTRACT

OpenDist provides efficient procedures and tools to synchronize our software file servers. This simple goal becomes challenging because of the size and complexity of supported software, the diversity of platforms, and because of network constraints.

Our current solution is based on *rdist*(1) [1]. However, it is not possible anymore to synchronize file servers nightly, because it takes several days just to compare distant servers.

We have analyzed the update process to find bottlenecks in the current solution. We measured the effects of network bandwidth and latency on *rdist*. We created statistics on the number of files and file sizes within all software packages.

We found that not only the line speed, but also the line delay contributes substantially to the overall update time. Our measurements revealed that adding a compression mode to *rdist* would not have solved our problem, so we decided to look for a new solution.

We have compiled a list of requirements for evaluating software distribution solutions. Based on these requirements, we evaluated both commercial and freely available tools. None of the tools fulfilled our most important requirements, so we implemented our own solution.

In the following we will describe the overall architecture of the toolset and present performance figures for the distribution engine that replaces *rdist*. The results of the prototype implementation are promising. We conclude with a description of the next steps for enhancing the OpenDist toolset.


### Our Environment

The CAD Support Group of the Semiconductor Division of Siemens AG installs, integrates and distributes all software needed to develop Integrated Circuits. We have development sites in Germany (München and Düsseldorf), Austria (Villach), the United States (Cupertino, CA), and Singapore. The development sites are connected by leased lines with a speed of 64 to 128 kBit/s. At each site, a central file server stores all software. Client workstations mount software from these servers. Software is installed and integrated in München and distributed to all other development sites. System administrators of the development sites initiate the transfer on the master server in München.

The CAD Support Group takes care of the CAD software and tools, only. A separate department is responsible for system administration, i.e., maintenance of the operating system and system tools, backups, etc.

Our software distribution problem differs in many ways from the one solved by traditional software distribution tools. Most software distribution tools we looked at are designed to distribute a moderate number of fairly static software packages of moderate size to many clients.

In contrast, we have to synchronize few file servers (under a dozen), which store many (about 200) packages of sizes ranging from tiny (a couple of kilobytes) to huge (1.8 GBytes). The total size of the software we store is currently 25 GBytes, 10-15 GBytes are currently being kept up-to-date at all sites. Many packages are changed each day. A change might update only a single file of a few bytes or could change up to 50,000 files for a total of 1 GBytes per day. Every month about 10 % of the software change. Most changes are small, but many files are constantly updated. The installation of a huge patch or a new software package changes many files at once.

There is no separate installation- or test-server, all changes are applied to the systems while our clients are using them. The changes are tested in München and, ideally, copied to all slave file servers within one day. Synchronizing or cloning file servers is the best way to describe our setup.

### Our Current Solution

Our current software distribution process uses *rdist*(1) to find changed files and to update slave software servers. It is no longer possible to compare two software servers in one night. A complete check of all software packages on the slave file server in Singapore would take several days which is not acceptable nor feasible. During that time, software packages would be in inconsistent states, and changes of the master software server could take up to a week to be transferred to the slave file server. Though it is possible to apply different update schedules – updating small packages daily, some weekly – the setup is not satisfactory. With an ever-increasing number of software packages and an ever-growing size of each software package, the distribution process using *rdist* is not acceptable any more.

### Searching The Bottleneck

We have analyzed the update process to find bottlenecks in our current solution. We analyzed our lines and measured bandwidth, latency and compression rate (all leased lines are equipped with datamizers – devices that compress all traffic). We created statistics on the number of files and their size for more than 200 software and data packages. Commercial software packages, technology data and cell libraries, as well as many free packages like X11 and gnu tools were analyzed. We were also interested in the compression rate and time of software packages and how much the compression rate differs when software packages are compressed file by file or as a complete archive. We analyzed where *rdist* spends its time during updates. Compared to the installed software, our change rate is small, so finding changed files must be efficient. Changes can be rather huge, so the transmission of changed files must be efficient, too.

### The Benchmark

We wrote a benchmark suite that measures the elapse time needed to perform typical software distribution operations such as installing, comparing, deleting, and updating files of different sizes, installing symbolic and hard links. All operations were executed many thousand times to equalize differences of the link performance.

The benchmark measures *ping* (1), *rcp* (1), and *rdist* (1) performance and times. Each *rdist* test runs on a directory with an appropriate number of random files of the same size. Each test contains an add, check, update and delete sequence. The file size is increasing from 1 Byte to 1 MBytes. Thus the effect of transfer rate and *rdist* protocol can be separated. The *rdist* part of the benchmark source tree contains approximately 5,000 files. This sums up to 10,000 transferred files, 5,000 check actions, 5,000 delete actions and 30 MBytes transferred data per test run. *rcp* (1) times are measured for a text, a binary and a compressed file of 1 MBytes each. This shows the achieved on-line compression.

The leased lines (except the dialup ISDN link) are shared by many users. So it is not astonishing that the benchmark results varied a lot, sometimes by more than a factor of three. To make our benchmark of the line performance more comparable, we calculated the average value for the best results of several runs of the benchmark. Some of the small numbers are within the magnitude of time resolution and must be interpreted cautiously.

### The Results

*Size and Composition*

Software packages vary substantially in size and composition of file types, however bigger packages don't necessarily have bigger files, they have a few huge files, but the average file size is more or less independent of the total size of the package (Diagram 1).

| | LAN | MÜNCHEN | DÜSSELDORF | VILLACH | CUPERTINO | SINGAPORE |
|---|---|---|---|---|---|---|
| Line Type | Ethernet | ISDN | X.25 | leased | X.25 | leased |
| Nominal Line Speed [kBit/s] | 10,000 | 64 | 64 | 128 | 64 | 64 |
| Transfer rate [kByte/s] | 90-100 | 6-7 | 4-5 | 7-12 | 2-3 | 3-4 |
| Ping Response Time [ms] | <1 | 33-88 | 188-372 | 81-311 | 530-1083 | 617-1375 |
| *rdist* file create [s] | 0.2 | 0.2 | 1.2 | 0.6 | 2.1 | 4.5 |
| *rdist* file check [s] | 0.02 | 0.06 | 0.5 | 0.2 | 1. | 2.1 |
| *rdist* file delete [s] | 0.1 | 0.13 | 0.5 | 0.3 | 1. | 2.3 |
| 10 kBytes transfer rate [kByte/s] | - | 5.9 | 2.5 | 4.9 | 1.5 | 1.4 |
| Run Benchmark [h] | 1 | 2.5 | 7 | 4 | 12 | 24 |
| *rdist* check SW subset [h] | - | - | 16 | 8 | - | >80[2] |
| OpenDist check SW subset [h][3] | 0.5 | - | 2[1] | .5 | .75 | .75 |
| *rdist* check all SW [h][2] | 3 | - | 69 | 27 | 140 | 290 |
| OpenDist check all SW [h] | 1.5 | - | 5[1] | 1.5 | 2 | 3 |

[1]Increased time, because software pools in Düsseldorf are accessed via NFS not UFS.

[2]Estimated.

[3]This subset consists of technology data and is changed and distributed daily. The subset contains approximately 150,000 files with a total of 1.1 GBytes.

**Table 1**: Line Characteristics

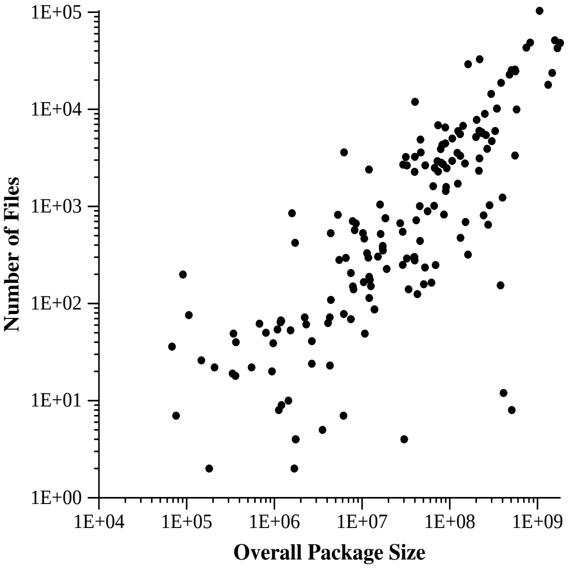


**Diagram 1**: Package Size vs. File Count

*Compression Factor*

The average compression factor of our software packages is three. Most of our software packages were compressed by this factor, though we observed compression factors between two and five.

When using *gzip*(1), you can regulate the compression speed between fast (less compression) and slow (best compression). For our software packages, increasing the compression quality reduces the compressed file size by less than 5 %, the compression time however sometimes increased by more than 200 % (Diagram 2). The default compression level of 6 is a good compromise, so we decided to use it.

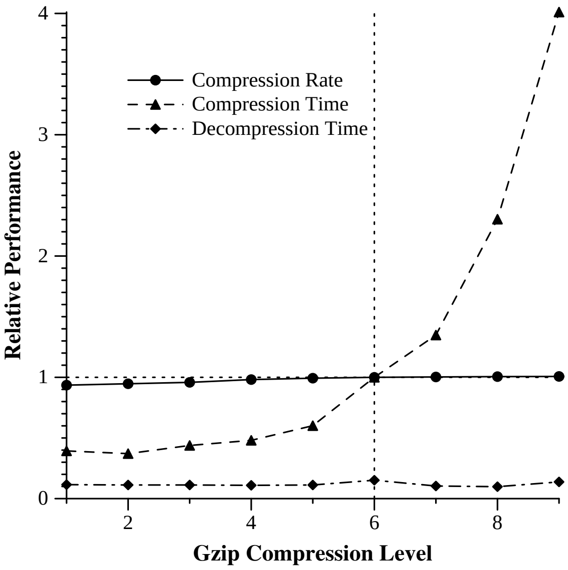


**Diagram 2**: Gzip compression Quality and Speed

Though our leased lines are equipped with datamizers that compress network traffic, it is worthwhile to compress archives before transmission. Datamizers increased the transmission rate of uncompressed data by 10 .. 15 %, whereas *gzip* reduced the data to a third of their original size.

*Compression Rate*

On a SPARCstation 10/41 (Solaris 2.4, 128 MBytes memory) *gzip* created compressed data at a rate of 65 kByte/s, many times faster than the speed of our leased lines. This figure is important to know when you want to pipeline the creation, compression, and transmission of update archives. In case the throughput of the lines is in the same order of magnitude as the *gzip* output rate, it would be advisable to decrease the compression level.

*Decompression*

Decompressing the archives with *gunzip*(1) is usually six times faster than compressing the data. Decompression time does not depend significantly on the compression quality chosen for compression (Diagram 2).

*Compression and Archives*

It is better to compress an archive of files than to archive compressed files. Compressing complete packages is significantly faster and creates smaller archives than compressing each file separately and archiving the compressed files. For example, archiving and compressing X11R6 was completed in three minutes elapse time, and the overall size was reduced by 55 %. Compressing each individual file and archiving the compressed files in a second step took five minutes elapse time and reduced the overall file size by only 45 %. All tests were performed several times on an unloaded machine. Compressing individual files and archiving them needs many more file and disk operations compared to archiving the uncompressed files and compressing the archive. Compressing several small files (or small network packets) is not as efficient as compressing the files in a single run.

*Transmission and Archives*

It is better to transmit an archive of files than to transmit each file individually. Depending on the file transfer protocol used, the latency of the line has a high impact on transfer rates. The smaller the files and the higher the latency, the higher is the delay caused by inefficient protocols.

The latency increases the time *rdist* needs to check or create files. If you have many files, *rdist* needs a long time to compare master and slave server. If many or all files changed (e.g. when installing a new software package), *rdist* will need much more time to transfer all files. The average file size of our software packages is 30 kBytes (Diagram 3). To our Singapore site, we need about 10 seconds (3 kByte/s) to transfer a file of this size. However, *rdist* needs more than 4 seconds to create the new file, for a total transmission time of 14 seconds (40 % increase), a 30 % decrease in transfer rate. The transfer rate for 10 kBytes files is only half of the normally achievable transfer rate (See Table 1).

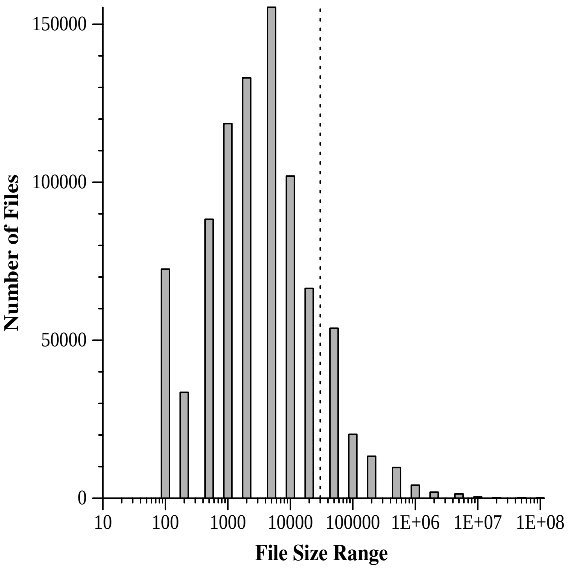


**Diagram 3**: File size range (All packages)

Besides avoiding protocol overhead, the transmission of archives has additional advantages. By first transferring all changed files to a holding disk, and installing changes locally on the remote server from the holding disk, the time during which the software package is in an inconsistent state is significantly reduced. Moreover, we can use the same tools to archive and roll-back changes. The installation of changes can be done asynchronously, so a system administrator at the remote site can easily postpone updates. The advantages compensate the disadvantage of needing holding disks to temporarily store the file archives.

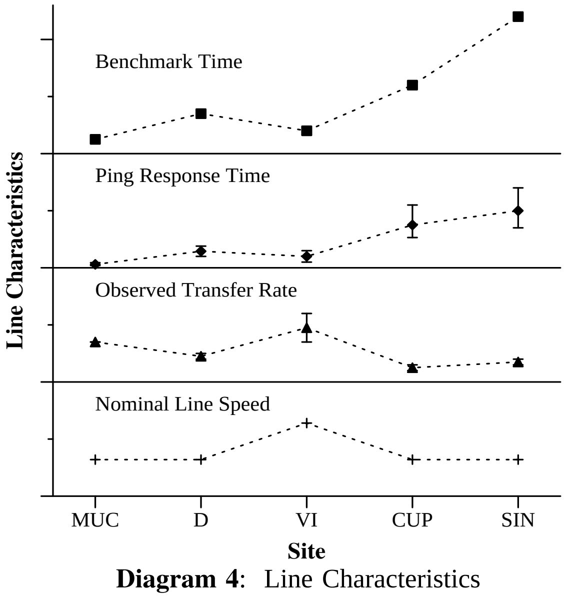


**Diagram 4**: Line Characteristics

*rdist and Latency*

Although the line speed from München to Villach and to Singapore differs by only a factor of two, the time needed to run the *rdist* benchmark differs by a factor of six (see Table 1 and Diagram 4). The ping response time (and therefore latency) has a greater impact on the time *rdist* needs to create or compare files than the line speed.

**Benchmark Summary**

Our measurements have revealed that the line speed is not the only bottleneck: the latency also plays an important role. *rdist* compares source and target directory file by file. Because the time for this is proportional to the latency, and because our change rate is small compared to the installed software, adding compression to *rdist* would not have solved our problem. *rdist* spent most of its time trying to figure out what to update, and not actually updating files. On the other hand, if a new version of our biggest software package is installed, we have to transmit 1.8 GBytes, so transmission must be optimized, too. The transmission of single files is another bottleneck as in our environment, the protocol overhead and transmission time are in the same order of magnitude, which reduces the average actual transfer rate by up to 30 %. For an efficient solution in our environment, files that have to be updated must be archived first and then be transmitted in one large file.

Upgrading our lines would not solve our problem, because the latency would not get small enough. It is also a very costly solution.

We found that we had to tackle two problems: making the finding of changed files more efficient, and making the transmission of data more efficient. We began to look for a new solution.

## Requirements for Software Maintenance

We compiled a long list of requirements that a new solution should fulfill. Here are some of the more important ones:

**Optimal Support of Incremental Distribution**

We do not want to trace changes as they are applied and re-apply them at a later date on slave file servers. Changes should be found by comparing the status of the master and the slave file server. Comparison should be stateless – it should not depend on update history. Each file server is administrated by independent system administrator groups, so we don't want to rely on what we think the status is, but we rather have to check the actual status of the remote file server. We have to detect changes applied by remote administrators.

**Update Programs Currently Executing**

Files that are updated may not be overwritten. The old file has to be moved and unlinked, then the new file has to be moved to it's final destination.

### Do Not Require Root Permission To Run

We install all software using unprivileged accounts and try to avoid using root permissions as much as possible. Synchronizing software file servers should be done using an unprivileged account, too. If root permission are required (e.g., to update entries in system files, or programs that require an user or group s-bit with a system owner- or group-ship), a script should be created, that is executed separately by the system administrator of the slave file server.

### Support Mapping

To allow localization, a flexible mapping of, e.g., file- and path-names, permissions, and owner should be supported. Software packages might be owned by different accounts on different servers. Symbolic links replace files to implement site specific changes (e.g., for configuration files).

### Support Execution of Scripts Before and After Updates

Before and after an update or roll-back it should be possible to execute scripts on the server and client. You might want to shutdown a database server and restart it after the update has finished. License servers might have to be re-started, if license files were updated.

### Transfer Data Efficiently and Reliably

The data transfer must be efficient, because our links are slow and have a high latency. If the link fails for a short period of time and the data transfer is aborted, transmit only the missing data, do not re-transmit all data.

### Be Humble

Do not require special installation of software packages. We don't want to change the installation of commercial software and have to support a variety of different package types.

### Should Support Roll-back

It should be possible to undo at least one update. If an update of a software package introduces problems, it should be possible to go back to the previous state of the software package. Also, if an update fails, roll-back the already applied changes to return to a consistent state.

### Minimize Inconsistent States

The time that a software package is in an inconsistent state (the time between the first and the last change that is applied) should be as small as possible. The time between applying changes to software packages that depend on each other should be as small as possible, too. Roll-back changes, if an update did not complete successfully.

### Avoid Errors Pro-actively

Try to verify in advance, whether an update is likely to succeed, e.g. check whether the target server has enough disk space to store new or changed files.

### Should Be Flexible

It should be easy to choose alternative distribution media: e.g., tape, email, direct network link. Comparing the status of master and slave file servers should be possible, even if no direct network link exists between the servers. The tool should be modular and extensible. Tool interfaces should exist and be well-documented.

### Should Use Standards

Use well-known existing standards and standard tools as much as possible. Do not re-invent wheels.

### Should Be Cost Effective

The cost for the product, its installation and customization, and its maintenance must be acceptable.

## Evaluation of Alternatives

We took a look at freely available tools, as well as commercial tools, proposed standards, and papers dealing with software management ([14], [17], [20], [21]).

### Freely Available Tools

The tools that we looked at can be categorized as follows: Tools that help to maintain source code and install software in a heterogeneous platform environment, like rtools [22]; Tools whose primary focus is network and disk space efficient installation and an unified setup and access by users in a campus network – tools like "The depot" [3], [18], depot-lite [7], opt_depot [24], ldd [4], lude [8], and beam [19] fall in this category; and tools that are designed to distribute software, like *rdist* [1], fdist [6], mirror [25], track [9], sup [2], and SPUDS [5].

All tools lack efficient incremental software distribution over slow WAN links. Bad assumptions include the set-up of software packages which imposes too tight restrictions. This won't work in our multi-vendor system environment. None cares about controlling the transmission in terms of media, scheduling, interruption or measurement and self-adoption. No roll-back support exists.

### Commercial Tools

Some commercial data distribution tools exist, as well as software management tools, that provide additional functions to cover a broader range of the software life-cycle, e.g., packaging, installation and de-installation. XFer from ViaTech, MLINK/ACM & DistribuLink from Legent, Tivoli/Courier from Tivoli, and DSM-SAX from SNI fall more into the data distribution category, whereas HP OpenView Software Distributor from Hewlett-Packard, and Sun-DANS from Sun Microsystems [13] fall into the latter category.

GUI and object-oriented methods and policies ease software packaging and automate distribution and gathering tasks. Typical application fields for these commercial tools are large companies with diverse offices with many client machines like financial or insurance companies.

All commercial tools claim to comply to standards, although it is sometimes hard to tell which standard they mean. Only one commercial tool purports to be compliant to the draft of the POSIX standard 1387.2 (formerly 1003.7.2) *Software Administration*.

We found it difficult to explain exactly what we mean by *incremental update* to some tool vendors. No package had proper mechanisms built-in. Incremental updates can be added to most commercial tools by writing scripts. Price is also a problem. Truly powerful tools won't start below $50,000 just for the licenses. Add an equivalent amount for installation, customization, maintenance, and updates. One benefit of commercial tools is to reduce the required skill and cost of the personnel at remote sites. This will not work for our few, demanding development sites.

Usage of commercial tools is problematic if you want to establish links to external companies. You need to buy licenses, so has your partner, too. All evaluated distribution software tools require that on both target and source locations daemons are running, and you have to pay a license fee per master and per client.

It's foreseeable that not all companies (esp. small consulting groups) are willing or able to spend the extra money and the extra effort of installation. Therefore it's very important for us to have a tool that can be used without any restrictions at least at the client side.

Two software packages looked promising, though:

*Tivoli/Courier*

Tivoli Systems implemented an extensive collection of system administration tools. One of them, Tivoli/Courier, allows automatic software distribution and control of server and workstation configuration. Tivoli/Courier is embedded into the Tivoli Management Framework. Together with other tools this forms a complete management environment. A graphical user interface and an object oriented approach allow easy maintenance of large systems. Tivoli/Courier allows to define software packages, different styles of scheduling, to define which files are updated at what time. Scripts can be added to customize the management environment to special requirements.

Tivoli/Courier does not fit to our requirements in respect to incremental distribution as we need it (It could be implemented by external scripts). It was not clear if we could run Tivoli/Courier stand alone without the framework or the other system administration tools. A direct network link is mandatory. All hosts involved in the update process need licenses and the software has to be installed as root. Reference customers seem to have a different profile (many hosts to update, static package design, smaller volume) than we have.

If we already had Tivoli Management Environment in productive use as the basis of our system administration, it would make sense to evaluate the performance of Tivoli/Courier. For the time being it would be too costly to implement the Management Environment to just use the Tivoli/Courier part.

*XFer*

XFer from ViaTech Technology was the second tool we evaluated very closely. XFer is optimized to solve the standard software distribution task: Update many hosts spotted over the globe with packages of (in our opinion) modest size and well-known structure. Compression and packaging of updates are standard. Packages, hosts and other resources are objects and can be managed efficiently. Machines can be grouped in "profiles". These profiles allow to send updates to machines which require certain software, regardless of type and location.

But at the time of the evaluation there was no support built in for incremental distribution (in our terms). High entry costs would pay off for many client hosts, but not for the few servers we run. Database and protocol overhead are not known. XFer would profit if installed together with cooperative network, user administration and configuration management tools, which are not available on our sites.

### POSIX 1387.2: Software Administration

P1387.2 provides the basis for standardized software administration. It includes commands to install, remove, configure, and verify software. A distribution format (install image) of software is defined along with commands to create and to merge distribution images. Provision is also made for tracking what software is installed and what its level is. Commands may be directed on one system to occur on any number of systems throughout your network.

There is a set of concessions to operating systems not based on POSIX.1 and POSIX.2, and there are exception conditions, so that systems such as DOS can conform to P1387.2.

*Status*

P1387.2 has passed its ballots within IEEE and has become the first POSIX system administration project to complete its work. Now P1387.2 has to be approved by the IEEE Standards Board, registered by ISO as a Committee Draft, and soon will be balloted as a Draft International Standard.

Copies of the current draft, P1387.2/Draft 14a, April 1995, are available from the IEEE Computer Society and from the IEEE Standards Office. A previous version of the draft (P1387.2/Draft 13) is also available by anonymous ftp [16]. See [15] and [23] for a more detailed discussion of the status of P1387.2 as of April 1995.

Suggestions for follow-on activity include a guide to best use of the current standard, a profile for DOS (and related) systems, version and patch/fix management, policies in distributed management (especially related to the definition of the "success" of an operation) and associated recovery policies, file sharing and client management, hierarchical distribution, scheduling, and queuing and queue management.

Because P1387.2 does not specify the means by which distributed functions occur, the System Management Working Group at X/Open are working to provide the necessary specifications to permit distributed interoperability.

*Relevance*

P1387.2 focuses on the distribution of software by installation. Software Service (patching software) has explicitly been left out of the standard, because existing schemes currently in use were too diverse. A possible solution is described in the rationale, though.

Once ISVs and all our internal developers of software, technology and library data use P1387.2 for their products, initial installation of software might become easier (or at least more similar). Hopefully even software service (i.e., applying patches) will eventually become standardized.

However, unless all our changes to all our software is done using standard procedures, we will have to clone file servers. Even if we were able to install all changes in a standard way, we would need some kind of queuing, so that we first can test the changes, before all servers are updated. Updates would have to be scheduled for night time. We don't believe that there will be a standard or a commercial product for cloning file servers any time soon.

## OpenDist

No available tool solved all important requirements, so we decided to implement our own tool set. OpenDist consists of administrative tools taking care of scheduling updates, statistical tools to report changes and performance of updates, and distribution tools do the actual update. Design goals are modularity and flexibility. The tools should be independently usable entities, easily exchangeable and should work together in changeable configurations. All tools are implemented in Perl [12], Version 5. Wherever possible, existing standard tools are used: e.g., GNU tar (*gtar*(1)) *gzip*, etc.

All tools currently use a command line interface. A more convenient graphical user interface for casual users will be added using TkPerl or a HTML browser.

**Administrative Tools**

These manage the scheduling of updates and call the distribution tools. Software administrators can subscribe and unsubscribe to software packages, query the software package database for information about each software package, temporarily postpone the update of selected packages, force an immediate update of selected packages, or roll-back updates. There are tools to browse the update history and to retrieve information about the status of each file server and software package.

Information about our software packages is stored in a flat (ASCII) file database, and includes: name and purpose of the package, status (test, old, current), dependency between packages, grouping of packages into bundles, recommended update frequency, contact information of package maintainer, etc.

**Statistical Tools**

These display performance information. Transfer rates and update duration are indicators of bottlenecks and problems with WAN lines. We need early indicators to be able to upgrade our network in a timely manner. The update history can be shown, as well as the current and historic free disk status on the file servers.

**Distribution Tools**

These distribute software packages, replacing *rdist* in our environment. The tools are optimized for low speed links with high latency. Software updates are transmitted in a compressed format.

They try hard to never leave a package in an inconsistent state, i.e. an update must be completed or has to be rolled back. Pre- and post-processing scripts are supported to save files before updating, or restart a license server after updating.

## The OpenDist Distribution Engine

The OpenDist distribution engine is implemented in several independent stages. Each stage has clearly defined input and output data formats. Tools can be easily combined and exchanged as long as the interface does not change. The different steps can be pipelined and performed in parallel when updating several software packages to further speed up the update and to optimized resource usage. For lines that do not support independent data transfers in both directions, the sending of updates and the retrieving of index files should not be done in parallel.

For our important packages we run the update once a day on the distribution server. The distribution process is controlled and mainly run on this server. We do not use the master file server but a dedicated machine as the distribution server. This server must have a direct access to all file servers. The index files and archives are temporarily stored on a holding disk.

### The Update Flow

A package is updated in the following steps:

- INDEX
  Create an index of the software package on the master and all slave servers. Compress and transfer to the distribution server's data-base.
- COMPARE
  Compare the master index against each slave index. Output a list of changed file attributes. Exceptions are handled here.
- BUILD-ARCHIVE
  Build a compressed archive with all changed files.
- BUILD-INSTALL
  Build an installation script which does the actual update on the slave server and takes care of changed attributes.
- BUILD-RESTORE
  Build a rollback script, which allows to undo the update in case of failure or in case the last status should be restored.
- TRANSMIT
  Transmit the archive and the scripts to the slave server.
- INSTALL
  Execute the installation script on the slave server and notify the administrator about success or failure.

### The Update Stages in Detail

*The INDEX Stage*

Every time an update cycle is started, first a sorted index of the software package is retrieved. The index is not stored on the remote system, but immediately compressed and piped to the master server. It is plain ASCII and sorted alphabetically by filename to make comparison of indices easier. Each line in the index file describes one object of the filesystem, e.g., a file, a directory, a FIFO, etc. The *perl* script *od_getindex* is started on the master and the slave servers from the distribution server. *od_getindex* scans the complete tree beneath a given directory. Symbolic links are not traversed. For each entry the returned index contains the attributes "name", "type & permissions", "size" and "modification time" by default. For symbolic links the link target is appended. For files with a link-count greater than one, the link count and the inode number are appended. Optionally more attributes like owner, group-id or checksum can be reported. The core of *od_getindex* is very simple and efficient (Figure 1). Of course, for the final version, we added more error checks.

The attributes are separated by a tab. Files with funny characters in their name (e.g., tabs, new-lines) are ignored and *od_getindex* will issue a warning message. A leading hash sign marks extra data or comment lines. Besides these characters no restrictions are imposed on filenames or package names. Usage of these and other non-printable characters should be avoided, anyway.

```
sub scan_it {
    my($Name) = @_; my(@filenames);

    ...

    ($dev, $ino, $mode, $nlink, $uid, $gid, $rdev, $size,
     $atime, $mtime, $ctime, $blksize, $blocks) = lstat($Name);

    ... if file not readable, print warning and return ...

    printf("%s\t0%o\t%d\t%d\t%d\t%d%s\n", $Name, $mode, $size, $mtime,
                 $nlink, $ino, -l $Name ? "\t->\t" . readlink($Name) : "");

    if (-d $Name && (! -l $Name)) {
        opendir(DIR, $Name);
        @filenames = sort(grep(!/^\.\.?$/, readdir(DIR)));
        closedir(DIR);

        for (@filenames) {
            &scan_it(sprintf("%s/%s", $Name, $_));
            }
        }
    }
```

**Figure 1**: Excerpt from *od_getindex*

Extra data like a time-stamp, the remote host name, blocksize in this file system or the mapping of user id to user name are added case by case. The format of the entries is described in “#FORMAT” lines.

The output from *od_getindex* is run through an ignore filter, compressed and finally transmitted to the distribution server. The index file uses complete filenames relative to the root of the software package and gets quite large (about 100 MBytes for our 25 GBytes of software), on the other hand, the compression rate is obviously very good (about a factor of 10).

The ignore filter is an optional script, which, e.g., allows to avoid the transfer of unwanted data like huge working directories. Ignore filters can be implemented site and/or package specific. The appropriate ignore filters will be transferred from the distribution server in advance.

Other output filters (e.g., for encryption) can easily be added to the stream. At this stage there is no difference between master and slave servers. The very same program is run on all locations where the package is found.

*The COMPARE Stage*

As soon as the index for a master and a slave package is available, differences can be calculated. *od_compare* opens both index files and reads them line by line. As the index files are sorted by entry name, missing or extra entries are recognized easily. An ADD or DEL tag followed by name and type is printed. Following are a list of attributes and values relevant for this type.

If an entry is in both index files, all significant attributes of this entry are compared. A list of changed attributes is printed.

A post processing filter cares about well-known exceptions. These exception filters can be defined per package and/or per site. Several possible actions like “notify administrator”, “ignore this” or “run script” can be triggered. An typical case is to exclude printer configuration files from being updated.

To allow for mapping of specific attributes, an optional mapping filter can be applied on the master index before comparison. Examples are mapping of user-id or names of entries.

*The BUILD Stage*

The remaining changes are handed over to the *od_build* script. This script reads in the changes and decides what action should be triggered, based on the type attribute. The build script packs all files, which need an update, into an archive. Currently we are using *gtar* to create the archive.

The build-update script counts the number of changes and the size of these changes. If specific built-in limits are reached different actions might be started. E.g. if the overall size of an update archive is very large, a tape might be written and sent instead of transmitting it over the direct link. If the

```
#Start "od_getindex /pool3/gnu", host: server, time-stamp: 806615319
#FORMAT_FILE     NAME    TYPM      MTIM    SIZE
#FORMAT_HARDLINK NAME    TYPM      MTIM    SIZE   NLNK   INOD
#FORMAT_SOFTLINK NAME    TYPM      MTIM    SIZE   ->     TARG
#BLOCKSIZE: 1024

... Directory:

gnu/sun4.1/bin         040775 806615225    1024

... File with linkcount > 1:

gnu/sun4.1/bin/gzcat  0100755 806614252   65536 2  75248
gnu/sun4.1/bin/gzip   0100755 806614252   65536 2  75248

... Symbolic link:

gnu/sun4.1/bin/gzcmp  0120777 806615225       6 -> gzdiff
gnu/sun4.1/bin/gzdiff 0100755 746115780    2008

... Plain file:

gnu/sun4.1/bin/gzexe  0100755 746115781    3864
gnu/sun4.1/bin/gzgrep 0100755 746115781    1341
...
#End "od_getindex /pool3/gnu", host: server, time-stamp: 806615321
```

**Figure 2**: Excerpt from *od_getindex* output

number of changes is above a given percentage, the administrator is notified.

Besides the archive, an installation and a roll-back script are generated. The installation script contains appropriate code for every change. It starts pre- and post-installation scripts. It checks for available disk-space on the target system and tries to avoid inconsistent states. The roll-back script is able to undo the latest update, even if the update failed half-way.

*The TRANSMIT Stage*

As soon as the archive, the install and the roll-back script are ready, they are scheduled for transmission. The actual transmission can run on a direct link, on tape or by email. The current implementation relies on a direct link.

The update package is transmitted in fragments suitable for the line speed. Each fragment gets a checksum and will be retransmitted in case of an error. The fragments might be encrypted. In case the package is going to be transmitted over a leased line, a flag can be set to postpone the transmission until non-busy hours, to avoid resource conflicts with other users. The fragments are decrypted and concatenated on the target system.

*The INSTALL Stage*

After the complete update package is on the slave server, the install script is started. At first the install script checks if the necessary disk space is available. Next all files which will be updated or deleted are written to a roll-back archive on the slave distribution holding disk.

Care is taken, that files are not overwritten. Old files are moved and unlinked, before new files are installed. This way, we can update programs that are currently executing.

Then the changes are applied. The install script runs all pre- and post-installation scripts at the right time.

If something fails with the update, the roll-back script is called to restore the latest consistent state.

Through the five independent steps changes can be implemented more easily. We could implement

```
#Start "od_compare gnu.stat.2 gnu.stat", host: od_host, time-stamp: 806616691
#MASTER #Start "od_getindex /pool3/gnu", host: server, time-stamp: 806615319
#CLIENT #Start "od_getindex /pool2/gnu", host: client, time-stamp: 806614037

... Changed attributes of file:
Change-MTIM: gnu/sun4.1/bin/gzcat    FILE  806614252 753610106
Change-PERM: gnu/sun4.1/bin/gzcat    FILE  0755      0777

... Replace a file with a symbolic link:
Change-TYPE: gnu/sun4.1/bin/gzcmp    SYML  SYML      FILE
Change-SIZE: gnu/sun4.1/bin/gzcmp    SYML  6         2008
Change-PERM: gnu/sun4.1/bin/gzcmp    SYML  0777      0755
Change-MTIM: gnu/sun4.1/bin/gzcmp    SYML  806615225 746115780
Change-TARG: gnu/sun4.1/bin/gzcmp    SYML  gzdiff    <UK>

... Added a symbolic link:
ADD:         gnu/sun4.1/bin/zcat     SYML  5
Change-PERM: gnu/sun4.1/bin/zcat     SYML  0777      <UK>
Change-TARG: gnu/sun4.1/bin/zcat     SYML  gzcat     <UK>
Change-MTIM: gnu/sun4.1/bin/zcat     SYML  806614148 <UK>

... Deleted a directory + files in it:
DEL:         gnu/sun5.2/lib          DIR   1024
Change-PERM: gnu/sun5.2/lib          DIR   <UK>      0775
Change-MTIM: gnu/sun5.2/lib          DIR   <UK>      806612281
DEL:         gnu/sun5.2/lib/libfl.a  FILE  1328
Change-PERM: gnu/sun5.2/lib/libfl.a  FILE  <UK>      0644
Change-MTIM: gnu/sun5.2/lib/libfl.a  FILE  <UK>      801569606

... Change the target of a symbolic link:
Change-SIZE: gnu/sun5.4/man          SYML  16      11
Change-TARG: gnu/sun5.4/man          SYML  ./share/man/man3 ./share/man
#End "od_compare gnu.stat.2 gnu.stat", host: od_host, time-stamp: 806616694
```

**Figure 3**: Excerpt from compare output

different INSTALL backends and use the most appropriate case by case.

The separation of finding and applying changes gives us more flexibility in trying not to run into disk space problems: We could remove all files that are going to be changed at the beginning of the update, or we could delete and add files simultaneously. We could even try to allocate extra disk space to prevent someone else from "stealing" disk space while we are installing the changes. However as long as several changes are applied simultaneously, it is not possible to guarantee that we will have enough disk space to install the changes, nor that we can roll-back to the previous status, though this is unlikely to happen.

### Results

OpenDist performs much faster than our previous, *rdist* based solution. Comparing a subset of 10 % of all software on the software servers in Villach and München can be completed in half an hour instead of 8 hours.

Creating the archive of files to be distributed must be done carefully. There is no backup or archive program that can cope with every special situation (files with holes, excessive pathname or symbolic link length, unreadable files and directories, or special (device) files) [10]. Currently we are using *gtar* and will document known limitations.

Distributing an archive of files helps to guarantee a consistent state of the target server. We do not start to change the slave server before the complete set of changed files has been transmitted to the slave server's holding disk. During the actual change of the slave server's software package, the network link might be down without affecting the update process.

Roll-backs of (at least one) update can be more easily supported than with *rdist*. The installation script can archive all files to be deleted or changed. The distribution engine creates a script that replaces the changed files with the saved ones.

A link (currently a direct network link) between master and slave server is needed only in stages INDEX and TRANSMIT. The COMPARE and the BUILD stage perform locally on the distribution server, independent of the line to the remote host.

Now instead of line speed and latency, index creation and compression became the bottleneck: the transfer rate from our Singapore site would allow to transfer the complete compressed index (5.5 MBytes) in about 25 minutes, whereas it takes about three hours now.

When you access the software over NFS, index creation slows down by a factor of 3 - 5. Therefore index creation should always be done on the file server that actually stores the software.

Small changes are very likely to be bug-fixes to software officially released and in use. Therefore these have to be found and applied as soon as possible. If a package is changed substantially, it's very likely to be new, not yet officially released. Changes to such a package are not as urgent and can be postponed to weekends and non-busy hours. OpenDist is very efficient in finding differences and there is enough bandwidth available to apply small changes on the fly. This allows us to run OpenDist once every day on all packages to apply small changes and postpone major changes to weekends.

The index files can be stored and the same tools can be used not only to find differences between different sites, but also to find changes to software packages in a specific period of time. Thus the same tools can be used to track changes applied to our software packages over time.

OpenDist needs no special installation on the remote systems. We can run the OpenDist procedure with any external company without forcing them to buy licenses or to install a huge tool package. All the tools used in OpenDist are freely available and can be distributed. Thus we could deliver, e.g., perl and gnu-software to the client, too.

OpenDist uses *rsh*(1) and *rcp*(1) which requires trusted hosts. We would not use this over public networks. However within our corporate network this is acceptable. It is also possible to replace *rsh* usage by a more secure mechanism, if required.

## Conclusion

### Is OpenDist Better Then rdist?

Are apples better than oranges? It depends.

The OpenDist toolset has more functionality than a distribution engine like *rdist*; in fact, we could use *rdist* as one of OpenDist's distribution engines. OpenDist fulfills all our important requirements and compared to our previous *rdist*-based solution, wins especially in the categories: efficient transfer over low speed, high latency lines; support to roll-back changes; and time period in which software packages are in an inconsistent state. Initial installation and configuration needs more effort, though. You will also need additional disk space for the holding disk on all file servers.

*rdist* is easier to setup and configure and is a standard part of most Unix systems (if your vendor ships an old version of *rdist*, upgrade to *rdist* Version 6 and complain at you vendor about the old version.) We still use *rdist* for many smaller tasks in the local network as well as to copy few files to remote servers.

Both tools have their limitations, and OpenDist depends on the tool used to create archives, which has its own set of limitations [10].

Though we started to just replace *rdist* with a better distribution tool, we realized that software distribution is closely coupled with other software maintenance tasks, like installing software, keeping information about installed software, de-installation, making software accessible by users. All stages of the software life cycle interact with each other. Changes made in one stage can help to solve problems in related stages.

## Future Work

The first implementation of the OpenDist distribution engine has proven to be superior to *rdist* in our environment, so we will further enhance it. Enhancements include reduction of re-transmitted bytes in case of line failures, optional transfer media (e.g., use tapes to transmit large low priority packages). We will also add more functions to administrative and statistics tools. Scheduling of updates and tools to query the status of slave software servers will be implemented next.

We will fine tune the parallelization of the update to further speed up the update. We plan to implement some of the needed update functions (move file before updating) within *gtar*, which will make the installation script simpler and more robust.

## Availability

At the time of writing, OpenDist is only available for use within Siemens AG. We hope to make the source for OpenDist available on the Internet at a later time. The paper and the slides for this talk are available by anonymous ftp from `ftp.ConnectDE.NET` in the directory /pub /sysadmin/sw-distribution/OpenDist/.

## Acknowledgments

Special thank goes to Tom Christiansen for reviewing early drafts of this paper, to all our system administrators for valuable discussions and encouragement, to Gernot Babin for his many contributions, and to SAM for all the support you have given. We thank the companies Interchip and Opis for their support during the evaluation of their products. We also thank Connect! for providing ftp and web space.

## Author Information

Peter W. Osel received his diploma in electrical engineering from the Technische Universität München (TUM) in 1985. For three years he worked at central research and development of Siemens AG, where he developed tools for ECAD of Integrated Circuits. Since 1988 he is working for the Semiconductor Division of Siemens. He is responsible for worldwide integration and distribution of the CAD system, as well as the development of central tools, and the coordination of the development sites' system environment. Reach Peter at Siemens AG, HL CAD, Postfach 801709, D-81617 München, Germany; or by e-mail at pwo@HL.Siemens.DE, or see his Web page at URL: http://www.ConnectDE.NET/people/pwo/.

Wilfried Gänsheimer received his diploma in electrical engineering from the Technische Universität München (TUM) in 1990. Since then he is working for the Semiconductor Division of Siemens. He started at the application support group for RISC microprocessors and Siemens microcontrollers. There he was responsible for definition, specification and test of development tools, customer support, benchmarking and simulation of system performance. He ran a small heterogeneous network (PC, X-Terminals, Workstations) inside the Siemens network. Since end of 1994 he is working in the CAD department. Management of licenses, installation of software, software distribution and user support are the main topics. Trouble shooting printer/plotter problems is his favorite time waster. Reach Wilfried at Siemens AG, HL CAD, Postfach 801709, D-81617 München, Germany; or by e-mail at wig@HL.Siemens.DE.

## References

[1] Michael A. Cooper, "Overhauling Rdist for the '90s", *Proceedings of the Sixth Systems Administration Conference (LISA VI)*, pp.175-188, Long Beach, CA, October 19-23, 1992

[2] Stephen Shafer and Mary Thompson, "The SUP Software Upgrade Protocol", Carnegie Mellon University, School of Computer Science, 1988. Available from mach.cs.cmu.edu in /usr/mach/public/doc/sup.ps.

[3] Wallace Colyer and Walter Wong, "Depot: A Tool for Managing Software Environments", *Proceedings of the Sixth Systems Administration Conference (LISA VI)*, pp.153-162, Long Beach, CA, October 19-23, 1992

[4] Walter C. Wong, "Local Disk Depot – Customizing the Software Environment", *Proceedings of the Seventh Systems Administration Conference (LISA VII)*, pp.51-55, Monterey, CA, November 1-5, 1993

[5] Ola Ladipo, "A Subscription-Oriented Software Package Update Distribution System (SPUDS)", *Proceedings of the Workshop on Large Installation Systems Administration*, pp.75-77, Monterey, CA, November 17-18, 1988

[6] Bjorn Satdeva and Paul M. Moriarty, "Fdist: A Domain Based File Distribution System for a Heterogeneous Environment", *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, pp.109-125, San Diego, CA, September 30 - October 3, 1991

[7] John P. Rouillard and Richard B. Martin, "Depot-Lite: A Mechanism for Managing Software", *Proceedings of the Eighth Systems Administration Conference (LISA VIII)*, pp.83-91, San Diego, CA, September 19-23, 1994

[8] Michel Dagenais, Stéphane Boucher, Robert Gérin-Lajoie, Pierre Laplante, and Pierre Mailhot, "LUDE: A Distributed Software Library", *Proceedings of the Seventh Systems Administration Conference (LISA VII)*, pp.25-32, Monterey, CA, November 1-5, 1993

[9] The track package is available at URL: ftp://ftp.cs.toronto.edu/pub/track.tar.Z

[10] Elizabeth D. Zwicky, "Torture-testing Backup and Archive Programs: Things You Ought to Know But Probably Would Rather Not", *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, pp.181-189, San Diego, CA, September 30 - October 3, 1991

[12] Larry Wall and Randal L. Schwartz, *Programming perl*, O'Reilly & Associates, Inc., Sebastopol, CA, 1991

[13] SunDANS (SoftDist) Manual, alpha draft, Sun Microsystems Computer Corporation, Mountain View, CA, September 1993

[14] Ram R. Vangala, Michael J. Cripps, and Raj G. Varadarajan, "Software Distribution and management in a Networked Environment", *Proceedings of the Sixth Systems Administration Conference (LISA VI)*, pp.163-170, Long Beach, CA, October 19-23, 1992

[15] Barrie Archer, "Towards a POSIX Standard for Software Administration", *Proceedings of the Seventh Systems Administration Conference (LISA VII)*, pp.67-79, Monterey, CA, November 1-5, 1993

[16] "POSIX 1387 System Administration Standard (draft 13)", available at URL: ftp://dcdmjw.fnal.gov/posix/

[17] John Sellens, "Software Maintenance in a Campus Environment: The Xhier Approach", *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, pp.21-28, San Diego, CA, September 30 - October 3, 1991

[18] Kenneth Manheimer, Barry A. Warsaw, Stephen N. Clark, and Walter Rowe, "The *Depot*: A Framework for Sharing Software Installation Acros Organizational and UNIX Platform Boundaries", *Proceedings of the Fourth Large Installation System Administrator's Conference*, pp.37-46, Colorado Springs, CO, October 18-19, 1990

[19] Thomas Eirich, "Beam: A Tool for Flexible Software Update", *Proceedings of the Eighth Systems Administration Conference (LISA VIII)*, pp.75-82, San Diego, CA, September 19-23, 1994

[20] D. Nachbar, "When Nework File Systems Aren't Enough: Automatic Software Distribution Revisited", USENIX Conference Proceedings, Summer 1986, pp.159-171

[21] Steven W. Lodin, "The Corporate Software Bank", *Proceedings of the Seventh Systems Administration Conference (LISA VII)*, pp.33-42, Monterey, CA, November 1-5, 1993

[22] Helen E. Harrison, Stephen P. Schaefer, terry S. Yoo, "Rtools: Tools for Software management in a Distributed Computing Environment", Proceedings of the Summer 1988 USENIX Conference, pp.85-94, San Fancisco, CA, 1988

[23] Nicholas M. Stoughton <nick@usenix.org>, "Standards Update, POSIX System Administration: Software Administration", Newsgroups: comp.std.unix, Message-ID: <3rsf12$nqu@cygnus.com>, 16 Jun 1995 10:29:06 -0700, available at URL: ftp://ftp.ConnectDE.NET/pub/sysadmin/sw-distribution/OpenDist/P1387.2-status-9504

[24] URL: http://www.arlut.utexas.edu/opt_depot/opt_depot.html

[25] URL: ftp://src.doc.ic.ac.uk/packages/mirror/