

A Tool for Massively Replicating Internet Archives: Design, Implementation, and Experience

Katia Obraczka
University of Southern California
Information Science Institute
4676 Admiralty Way
Marina del Rey, CA 90292, USA
katia@isi.edu

Peter Danzig, Dante DeLucia, Erh-Yuan Tsai
University of Southern California
Computer Science Department
Los Angeles, CA 90089-0781
{danzig, dante, erhyuant}@usc.edu

Abstract

This paper reports the design, implementation, and performance of a scalable and efficient tool to replicate Internet information services. Our tool targets replication degrees of tens of thousands of weakly-consistent replicas scattered throughout the Internet's thousands of autonomously administered domains. The main goal of our replication tool is to make existing replication algorithms scale in today's exponentially-growing, autonomously-managed internetworks.

1. Introduction

Internet services provide large, rapidly evolving, highly accessed, autonomously managed information spaces. To achieve adequate performance, services such as WWW [1] will have to replicate their data in thousands of autonomous networks. As an example of a highly replicated service, take Internet news [5]. Although it manages a dynamic, flat, gigabyte database, it responds to queries in seconds. In contrast, popular WWW and FTP servers are constantly too overloaded to provide reasonable response time to users. As WWW, FTP and other Internet information services become more popular, their databases must be highly replicated for performance.

Existing replication mechanisms will not scale in today's exponentially growing, autonomously managed internets. We implemented a tool for efficient replication of Internet

information services. We target replication degrees of thousands or even tens of thousands of weakly consistent replicas scattered throughout the Internet's thousands of administrative domains.

Our tool consists of two components, *mirror-d* and *flood-d*. *Mirror-d* propagates updates according to the logical topologies computed by *flood-d*. *Flood-d* aggregates replicas into multiple *replication groups* analogous to the way the Internet partitions itself into autonomous routing domains. Having multiple replication groups preserves autonomy, since administrative decisions of one group, such as its connectivity or when it should be split in two, do not affect other groups. Also, it insulates groups from topological rearrangements of their neighboring groups and from most of the network traffic associated with group membership.

For each replication group, *flood-d* automatically builds the logical topology over which updates travel. Unlike Lampson's Global Name Service (GNS) [6], *flood-d*'s logical update topologies are not restricted to a Hamiltonian cycle. By automating the process of building update topologies among replicas of a service, *flood-d* offloads system administrators from having to make logical topology decisions.

We argue that efficient replication algorithms flood data between replicas. Note that the flooding scheme that we propose differs from network-level flooding as used by routing algorithms: flooding at the network level simply follows the network's physical topology and flood updates throughout all physical links of the network. Instead, the replicas flood data to their logical neighbor or peer replicas. Although the word "flooding" sounds inefficient, we claim that the application-level flooding scheme that we propose does use network bandwidth efficiently.

Flood-d employs a network topology estimator and a logical topology calculator. Every group member measures available bandwidth and propagation delay to the other

This research was funded in part by ARPA contract number DABT63-93-C-0052, AFOSR award number F49620-93-1-0082, NSF NYI award NCR-9457518, and NSF small-scale infrastructure grant number CDA-9216321. Katia Obraczka is currently supported by ARPA contract number J-FBI-95-204.

group members. Based on these estimates, the logical topology calculator builds topologies for the group that are k -connected for resilience, have low total edge-cost for efficient use of the underlying network, and low diameter to limit update propagation delays.

Figure 1 illustrates the relationship between logical topologies and the underlying physical network. The left-hand figure shows three *overlapping* replication groups and their logical update topologies. The right-hand figure shows the physical network topology and the logical update topology built on top of it for the three replication groups in the left-hand figure.

1.1. What Current Algorithms Lack

As existing naming services and distributed file systems have demonstrated, the problem of replicating data that can be partitioned into autonomously managed subspaces has well-known solutions. Naming services such as the Domain Name Service (DNS) [8] and Grapevine organize their name space hierarchically according to well-defined administrative boundaries. They also use these administrative boundaries to partition their name space into several *domains*, which only need to be replicated in a handful of servers to meet adequate performance. In fact, according to the results we report in [3], over 85% of second level domains in the DNS hierarchy are replicated at most three times, while 100% of these domains use at most 7 replicas. The same study also shows that more than 90% of DNS's second-level domains store less than 1,000 entries. Because of the limited domain sizes and small number of replicas, DNS's primary-copy replication scheme performs quite adequately.

Similarly, distributed file systems organize their file space hierarchically, where intermediate nodes are directories and leaf nodes are files. Like LOCUS [11], Andrew-AFS¹ [4], and Coda [12], distributed file systems use locality of reference to partition their file space into directory subtrees. File servers replicate a subset of files in a directory subtree. Both LOCUS and Andrew provide read-only file replication, while Coda uses distributed updates to keep its writable replicas weakly consistent.

Because layered network protocols hide the network topology from application programs, replicas themselves cannot select their flooding peers to optimize use of the network. Both Grapevine and its commercial successor, the Clearinghouse [10] ignore network and update topology. GNS assumes the existence of a single administrator who hand-configures the topology over which updates travel. The GNS administrator places replicas in a Hamiltonian cy-

cle, and reconfigures the ring when replicas are added or removed. As the number of replicas grows and replicas spread beyond single administrative boundaries, frequently reconfiguring the ring gets prohibitively expensive.

Netnews employs flooding to distribute updates among its thousands of replicas. Like GNS, NNTP site administrators hand-configure their logical flooding topology. Since obtaining current physical topology information is difficult in today's Internet, system administrators frequently confer with one another to plan changes in the logical flooding topology. They try to keep up with the dynamics of the underlying physical topology, specially as the Internet's scale and complexity increase.

The main contribution of the replication tool we built is to make GNS-like services scale in today's exponentially growing, autonomously managed internetworks. In this paper, we describe our experience designing, implementing, deploying and measuring our replication tool's performance.

2. Design

The Harvest resource discovery system [2] has been designed and implemented to solve the scalability and efficiency problems of early resource discovery services. For availability and response time, Harvest relies on massive replication of its servers. In particular, Harvest's directory, which stores information about all available servers, is expected to be highly accessed and must be massively replicated for adequate performance.

Although flood-d was designed to support Harvest servers' replication, it was built as an independent service to allow its use by other applications. Indeed, several existing Internet information services such as Network News, FTP archives, and WWW servers could use flood-d to propagate their updates more efficiently and timely, yet reducing the burden on system administrators.

The replication tool we implemented consists of two separate services: flood-d and mirror-d, whose name expresses its reliance on the well-known FTP-mirror package [7]. Flood-d estimates the underlying physical network load by measuring available bandwidth and round-trip time (RTT) between members of a replication group. Based on these estimates, flood-d computes a fault-tolerant, low-cost, low-diameter logical update topology for the group. Mirror-d, a weakly-consistent, replicated file archiver uses these logical topologies to propagate updates timely and efficiently.

To simplify replicating Harvest brokers, mirror-d uses a master copy replication scheme. All updates are performed at the master site, with replicas being read-only copies. Each replica has a version number that determines if a replica needs to be updated. Replicas pull data from neighbors rather than having a neighbor push data. This

¹Andrew is the name of the research project at Carnegie-Mellon University. AFS is based on Andrew, and has become a product marketed and supported by Transarc Corporation.

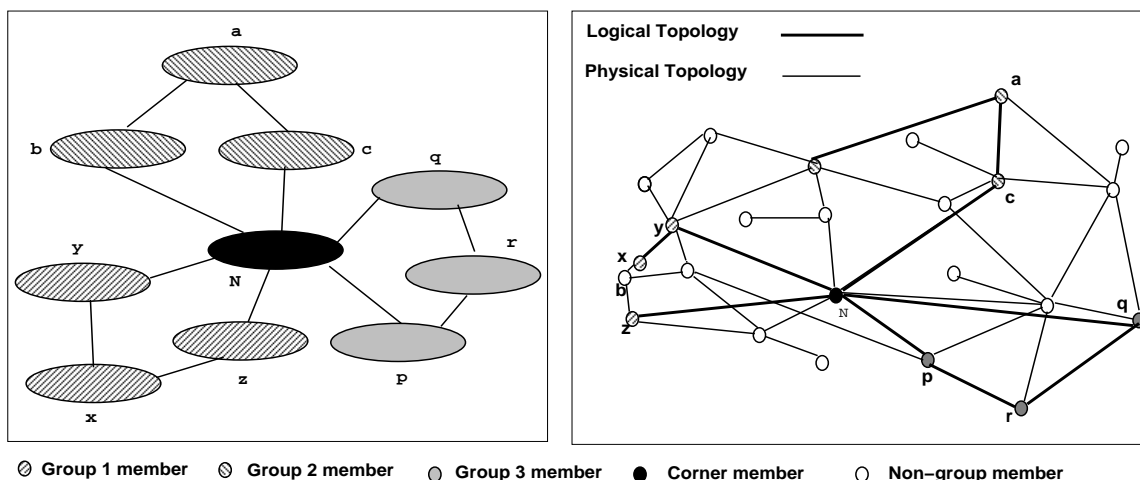


Figure 1. Replication groups showing logical versus physical topologies.

avoids the problem of multiple concurrent updates.

When a replica completes an update, it sends out a notification to its neighbors with the new version number. When a neighbor receives a notification it checks if its local version is out of date. If so, it then invokes mirror-d to update its local copy. Since it is relatively cheap to send out update notifications, mirror-ds can request them periodically to ensure that their local copies are up to date. This works well when replicas crash and lose update notifications, or are interrupted midway in an update process.

Another design decision was the use anonymous-FTP as the file propagation mechanism. While this complicates the installation of mirror-d, it has the advantage that most system administrators are familiar with anonymous-FTP. Additionally, anonymous-FTP is widely supported, and much work has been done to eliminate its security holes.

2.1. Consistency Between Groups

Consistency between replication groups is maintained as easily as it is between members of a group. Representative individual replicas, or *corner replicas* belong to multiple groups. Since replicas flood updates to neighbors in the logical topology, updates in one group make their way to all groups.

Although network node and link failures may result in network partitions and permanent node failures and group membership changes may introduce temporary inconsistencies, they are eventually resolved as long as flood-d topologies keep the nodes connected.

2.2. Updating Logical Topologies

Network nodes and links may fail temporarily, or may be permanently removed from service. Replicas may also join

and leave a flooding group. The group membership protocol and physical topology estimation will eventually detect these changes, which will be reflected in the new topology graph computed for the group.

Our replication tool uses flooding to propagate topology updates to all members of a replication group. Topology update messages carry a sequence number corresponding to the topology identifier, which replicas use to order topology updates, and detect duplicates. Topology update messages also contain the new group membership. When a replica receives a topology update, it floods the new topology according to the current topology before committing the new topology.

The topology update process generates additional traffic associated with propagating topology update messages to the participating replicas. The resulting overhead in terms of the total number of messages generated is proportional to the number of participating replicas, and the frequency with which topology updates occur. In a highly replicated service whose copies are spread throughout large internets, the topology update overhead may become prohibitively expensive. As the number of replicas increases, the overhead associated with maintaining group membership information, and estimating communication costs becomes excessive. Our hierarchical approach helps limit this overhead. Grouping replicas located physically close to one another restricts the scope of the changes of topology updates. It also limits the scope of the resulting topology updates to the local group, and therefore restricts topology update traffic on the more expensive, long-haul physical links.

The aggregate cost of topology estimation grows as $O(n^2)$ where n is the size of the group. On one hand, the more estimates collected, the more adaptive to network and server load changes the group is. On the other hand, the

higher the estimation frequency, the greater the impact on network utilization. Section 5 presents estimation traffic measurements collected from replication groups of different sizes.

3. The Flood Daemon

A flood-d replica computes bandwidth and RTT estimates to other replicas in its group. A single designated site, the *group master*, generates topologies for the group. Flood-d handles group membership of one or more replication groups.

The first topology the group master generates after a new member joins will not be very good since the new member has not had time to perform bandwidth and RTT estimates between group members. The topology will get better as estimates improve.

Flood-d was written to be fast and robust. Consequently it is written as a Unix daemon that does not fork, but instead uses non-blocking I/O for all communication. The interface to flood-d is via an interface that can be queried with either ‘telnet’ or more conveniently with a WWW browser that speaks HTTP [1].

3.1. Membership and Multiple Groups

When a new site joins a group, it sends a join request to an existing group member. As soon as a replica receives a join request from a site, it adds the new site to the list of known sites and starts collecting network estimates for that site. The replica that receives the join request also floods it out to all replicas in the group. A site is not part of the the group until the master distributes a new topology that contains the site. This naturally gives the master control over group membership.

There is no protocol for leaving the group. Sites leave a replication group silently; after a pre-determined period of time, if a site has not been heard from, it is simply dropped from the group. This silence period is configurable and is currently set to 1 hour. Setting the silence period should take into account other group parameters such as the RTT time and bandwidth estimation periods, as well as the estimate reporting period.

A flood-d replica can be a member of more than one group. This is the case of Figure’s 1’s replica N.

3.2. Estimate Collection

A flood-d replica periodically performs RTT and bandwidth estimation between itself and other members of the group. To avoid synchronous group behavior, we add a random offset to the estimation frequency. Over time, a site

builds estimates of RTT and available bandwidth to all other members of the group.

For RTT estimates, a replica sends a UDP packet containing a timestamp to a randomly-selected group member. When a flood-d receives such a packet it simply sends the packet back to the originator. The returned packet is then used to estimate the RTT between flood-ds. Similarly to RTT estimation, a flood-d replica estimates bandwidth by periodically choosing a random site and sending a block of data to that site. The default block size is 32 KBytes. Available bandwidth is defined as the

$$bandwidth = bytes_sent / (time_{last_byte} - time_{first_byte})$$

The times at which the destination replica received the first and last bytes are given by $time_{first_byte}$ and $time_{last_byte}$ respectively. In order to build up a base of statistics more quickly, bandwidth is measured both when data is sent or received. Bandwidth estimation is performed whenever the data transfer meets or exceeds the bandwidth block size. While these means of collecting statistics are admittedly not very precise, they serve our purposes well enough. In Section 5, we report available bandwidth estimates using different data block sizes.

When computing the actual estimates to report to the group master, previous history is taken into account. This prevents adapting to transient changes. The damping effect is computed as follows:

$$new_estimate = \alpha * old_estimate + (1 - \alpha) * current_estimate$$

where $old_estimate$ is the previously reported estimate, and $current_estimate$ is the estimate just measured. The damping rate α is set according to how much weight is given to past history. Currently we set α to 0.90.

To build up group estimates as quickly as possible, the periodic estimation algorithm “fast-starts” by performing more frequent estimates when flood-d first starts. Over time, the estimation process slows down to reduce the impact of bandwidth and RTT estimates on network utilization.

There is an obvious tradeoff between the ability of the system to adapt to network conditions and the amount of overhead incurred by bandwidth and RTT estimation. Large groups will need to perform more estimation than small groups. This difference is not linear since a n -replica group performs $O(n^2)$ estimates.

The end-to-end bandwidth and RTT estimates take into account the actual load on the servers involved. Measurements done at the network level might be more accurate in terms of the actual network statistics, but they do not reflect the actual delay and bandwidth seen by the application. For instance, the fact that a server tends to be heavily loaded is

just as important as network congestion as far as applications are concerned.

The master collects estimates reported by group members into a cost matrix for the group, which is then used to compute the group’s current logical update topology. Each entry C_{ij} in the cost matrix corresponds to the communication cost between nodes i and j , which is given by B_{ij}/D_{ij} , where B_{ij} and D_{ij} are the estimated bandwidth and RTT between i and j , respectively.

3.3. Topology Calculation

Flood-d generates logical update topologies by invoking a topology generator. The current topology generator uses as input the estimated cost matrix and the connectivity requirement k . It computes a minimum cost spanning tree with extra edges to provide redundancy in the event of link failure and to decrease the graph’s diameter. The algorithm first computes a minimum spanning tree connecting all the nodes, and then, for each node whose degree d is less than the required connectivity k , adds the current cheapest edge until $d = k$. We are currently using $k = 2$.

The original topology generator [9] produced low-cost, low-diameter, k -connected topologies for a group using simulated annealing as its optimization technique. Our experiments demonstrated that this sophisticated algorithm only produced moderate reductions in total edge cost. Consequently, in practice we use a simpler, faster, less optimal algorithm.

Because the simulated annealing algorithm tries to lower both total edge cost and diameter, it may select more expensive links over cheaper ones. For instance, our testing environment contained several 28K PPP links. Frequently, a replica would attempt to replicate across a slow link, when there was another replica available via a local ethernet. We do not notice such phenomenon in the topologies generated by the minimum spanning tree algorithm.

4. The Mirror Daemon

A mirror-d replicated archive consists of a *master copy* and read-only replicas. Each replica contains the file system hierarchy being replicated and an associated version number. When the master site is updated, it issues a command to its mirror-d, which creates a new version number. It then sends out update notifications to neighbors according to the logical topology flood-d generates.

An update notification contains a version number, the name of the host on which the replica sending the update resides, information required to access the archive via FTP, and a template containing parameters to be used by the FTP-mirror package.

A read-only replica’s mirror-d is responsible for determining if the replica’s local copy is out of date. When a replica receives an update notification, it checks if the update’s version is more recent than the local version. If it is, it places the notification in a notification set. After receiving several notifications, mirror-d selects from the notification set the update notification that came from the neighbor with the best bandwidth/delay metric according to the local flood-d. Using this update notification, mirror-d builds an FTP-mirror configuration file from the FTP-mirror template, and then starts an FTP-mirror process. If the mirror process succeeds, the local version is updated and any redundant notifications are purged from the notification set. If the update fails, mirror-d selects another notification, and the FTP-mirror process repeats.

A mirror-d determines its neighbors by querying the local flood-d, and extracting a list of all neighbors and their corresponding logical link cost metrics. The fact that these neighbors might be in different replication groups is transparent to mirror-d. The mirror-d applications know nothing about the existing replication groups. Indeed, mirror-d could be easily hand-configured with pre-defined neighbors. One would lose the elegance of automatic topology calculation, but for some applications it might be useful that mirror-d be independent of flood-d.

Since a mirror-d will have several neighbors, it is often the case that it will receive update notifications from several of them. The mirror-d implementation never acts immediately on update notifications, but instead waits at least a minute to allow time for multiple update notification to arrive. It then orders the update notifications with the highest bandwidth, lowest delay neighbor first. This avoids the situation where a mirror-d will mirror over a 28K link, when it could do it over a local ethernet.

5. Performance Measurements

We have conducted preliminary performance measurement experiments with our replication tool. The goal of the first set of experiments is to evaluate the overhead flood-d generates, while the second experiment tries to assess the impact of the message size on the estimated available bandwidth.

5.1. Flood-d Overhead

Recall that replicas in a group periodically estimate RTT and available bandwidth to the other group members. For this experiment, the time between RTT and bandwidth estimation was set to 15 minutes and 1 hour, respectively. From time to time, replicas report their estimates, which the group master uses to compute a new logical topology

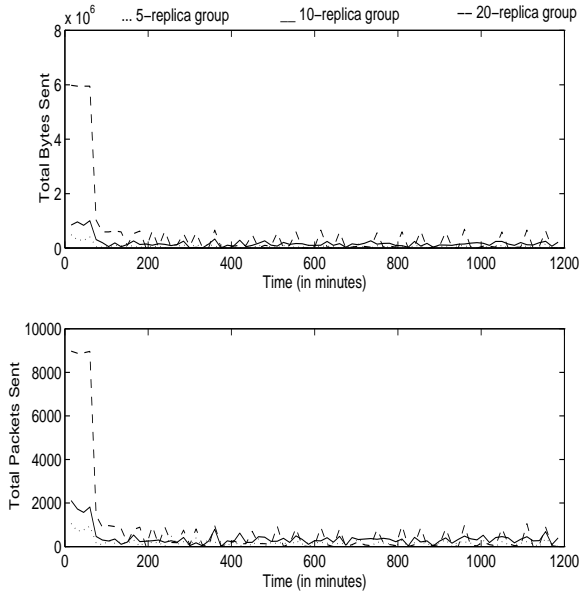


Figure 2. Total Estimation traffic for 5-, 10-, and 20-replica groups.

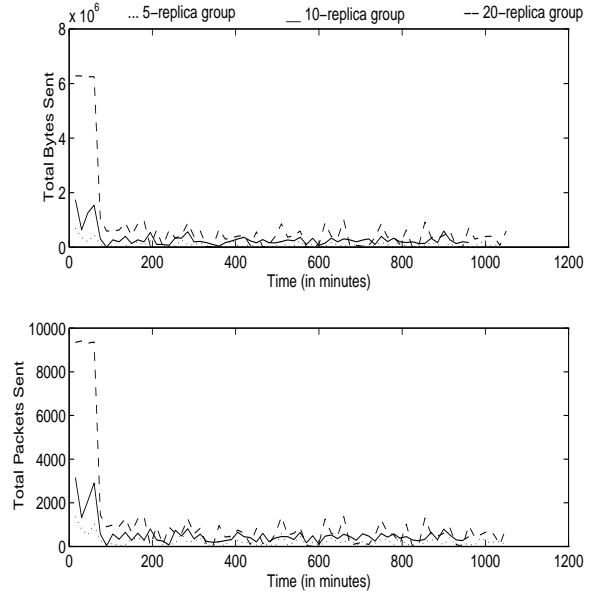


Figure 3. Total Estimation traffic for 5-, 10-, and 20-replica groups with dynamic membership.

for the group. The estimate reporting period for this experiment was set to 30 minutes. The master then floods the topology to all group members. The time between topology updates was set to 1 hour.

Replicas send RTT probes and RTT probe replies using UDP. For estimating bandwidth, sending estimates to the master, and propagating topology, they use TCP. Figure 2 shows the total overhead traffic in bytes and packets generated by a 5-, 10-, and 20-replica group. In these experiments, groups were created by having replicas join one by one. We recorded 20-hour traffic traces from replicas running locally in our laboratory. The measurements were taken every 15 minutes.

The reason we observe more traffic during the first 90 minutes is due to the fact that replicas “fast-start” when they start execution. In other words, when a replica starts, it performs estimates more frequently so that the replica’s view of the group builds up faster. Notice that the amount of traffic stabilizes around a significantly lower value once the group membership becomes stable. For the 20-replica group, flood-d generates 6 MBytes of overhead traffic during approximately the first 90 minutes. From then on, flood-d’s overhead traffic oscillates around 1 MByte.

Figure 3 shows how dynamic membership affects the overhead traffic flood-d generates. In this experiment, replicas join one by one in the beginning, and then, every hour a replica leaves and re-joins the group. We observe that the additional traffic generated when a replica re-joins the group is not considerable. The additional traffic generated

is due to the fast-start of the re-joining replica.

5.2. Message Size and Available Bandwidth

For this experiment, we set up a wide-area, 6-replica group so that we could evaluate the impact of the bandwidth estimation message size on the estimated available bandwidth. Table 1 shows bandwidth estimates taken from a replica at USC, *ventura.usc.edu* to the other members of the group. Notice that for all sites the 16 KByte message size resulted in lower bandwidth estimates. This is probably due to the fact that the 16 KByte message was not large enough to open up the TCP window.

6. Experience

The building of flood-d and mirror-d has been a very enlightening experience. Much of the experience we gained was of the practical sort and we hope to pass some of it along to others who might be contemplating this type of work. The items below attempt to highlight some of the lessons we learned.

- Some of the problems encountered were directly related to the sheer volume of data and the large number of objects that had to be replicated. *Flood-d* was originally designed for small objects with relatively small updates (1 to 2 MBytes per update). Fortunately the

Replica Name	Available Bandwidth (Kbytes/sec)			
	16 Kbytes	32 Kbytes	48 Kbytes	64 Kbytes
buzios.usc.edu	455.4	503.6	514.3	457.2
sunw0.cse.psu.edu	12.6	18.3	18.5	19.4
powell.cs.colorado.edu	9.2	17.3	27.7	16.8
casino.cchs.su.edu.au	1.8	2.0	1.4	1.5
aloe.cs.arizona.edu	7.2	9.4	20.0	8.7

Table 1. Bandwidth estimates collected at ventura.usc.edu using different message sizes.

system was designed to be modular with the ability to interface with a variety of components to handle data management. Consequently it was relatively easy to redesign the data management portion of the project and handle the archives required by Harvest.

- There needs to be a widely implemented standard for threads. Flood-d has been written entirely in C using standard UNIX system calls. This allows flood-d to use a very portable lightweight process and gives us great flexibility in dealing with network problems. A drawback is that is that non-blocking I/O is fiendishly difficult. The obvious solution to this is to use a threads package. Unfortunately, no portable threads packages were discovered that could deal with a wide variety of I/O on multiple platforms.
- Interpreters are nice for rapid prototyping. The mirror-d application is written in Perl. While Perl is not the most elegant scripting language, one can generate a portable prototype very quickly.

On the down side, Perl rarely seems to be installed correctly, which one might construe as a portability problem. Fortunately, this is relatively simple to solve.

A major drawback of Perl is the lack of precise memory management. Perl has a tendency to grow over time. Try as we might, we could not prevent Perl from growing. When writing a daemon process, this is very problematic. Perl tends to grow very large and eventually swamp the machine. The incredibly ugly hack solution to this is to have the Perl application restart itself periodically. Not elegant, but effective.

- FTP is not good for small file transfers. Ordinarily this is not a problem, but when transferring 16000 files of 500 bytes in size, the overhead is non-trivial. The obvious solution is to aggregate the smaller files into a much large files so that TCP can get through its slow-start phase and send data at reasonable rates.
- When moving files across multiple timezones with mirror site chosen dynamically, the issue of time becomes important. Some existing FTP daemons return

file timestamps in local time, while others return time in GMT. Despite our best efforts, some host/software combinations refuse to return time in GMT. It would be helpful some notion of time were incorporated into anonymous FTP so that file timestamps were always returned in a standard timezone, or at least allow a user to discover the timezone utilized at the remote site.

- Memory is **NOT** cheap. Your application will at some point have to run on a “normal” machine. If one goes on the assumption that memory is cheap, perspective is quickly lost and memory utilization is not accounted for. If a machine has n MBytes of memory, it is always easy to use $2n$ MBytes of memory if one assumes memory is cheap. Is is best to assume that memory is a limited resource. The same can be said of disk space. Always aim for efficiency. Let someone else figure out how to use all that space, since you can always be sure that they will.
- Many assumptions are made about network connectivity. In our experience the only assumption one can make about internet connectivity is that the probability that any two sites are disconnected is non-zero. When working with a live internet, connectivity problems are rampant. At one period in time, network connectivity between USC and U Colorado Boulder network was lost with depressing regularity. When moving large amounts of data around (60 MBytes to 100 MBytes), these sort of problems can cause major headaches. It is very helpful to be able to start the transfer over from any point. When distributing these large archives, one cannot assume atomic updates, and must instead assume that update will be incremental.
- As for user interfaces, by far the best that we have found yet is HTTP. HTTP is an incredibly easy way to write a lightweight remote interface to a program. Once this has been done, it is a simple matter to find a WWW browser and use it as the front end. While HTTP may constrain the type of interaction with the application, it provides a good portable interface for many purposes.

- Care must be taken when generating data to be used for bandwidth estimation. Modern modems have the wonderful ability to automatically compress data. If textual data, or non-random data is used, the modem will in all likelihood do a reasonable compression job. We were getting reports of 6K per second over 28K dialup links when using non-random data, and 2k when using random data.

7. Conclusions and Future Work

In this paper we reported the design, implementation, and performance of a scalable and efficient replication tool for Internet information services. The primary goal of our replication tool is to make existing replication algorithms scale in today's exponentially growing, autonomously managed internetworks.

Our replication tool consists of two independent services: flood-d and mirror-d. Flood-d generates fault tolerant, low cost, low delay logical flooding topologies, which mirror-d uses to maintain weakly consistent FTP archives. Flood-d was designed as an independent service to allow its use by other Internet services, such as WWW servers, FTP archives, and an Internet cache hierarchy.

Our preliminary performance measurements indicate that flood-d's overhead behaves as expected, that is, it stabilizes over time. However, only after measuring the network and server resources flood-d's logical update topologies save, can we fully evaluate the benefits of using flood-d. This is the goal of the wide-area experiment we are currently setting up to replicate commonly accessed Internet archives. In this experiment we hope to demonstrate how flood-d's logical update topologies can reduce the mirror load on popular Internet archives, and at the same time, make more efficient use of the network, and reduce update propagation time.

In the longer term, we will investigate the use of IP multicast for bulk data distribution. Since IP multicast is a best effort protocol, a reliable transport mechanism must be built on top of it for applications that require reliable delivery. There are several efforts building reliable multicast transport protocol. We are focusing our efforts on building a flow control mechanism for bulk data distribution that can be used in conjunction with these protocols.

References

[1] T. Berners-Lee, R. Cailliau, J-F. Groff, and B. Pollermann. World-Wide Web: The information universe. *Electronic Networking: Research, Applications and Policy*, 1(2), Spring 1992.

[2] C.M. Bowman, P.B. Danzig, D.R. Hardy, U. Manber, and M.F. Schwartz. The Harvest information discovery and access system. Proceedings of the Second International World Wide Web Conference, October 1994.

[3] P.B. Danzig, K. Obraczka, and A. Kumar. An analysis of wide-area name server traffic: A study of the domain name system. *Proc. of the ACM SIGCOMM '92, Baltimore, Maryland*, August 1992.

[4] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[5] B. Kantor and P. Lapsley. Network news transfer protocol - a proposed standard for the stream-based transmission of news. Internet Request for Comments RFC 977, February 1986.

[6] B. Lampson. Designing a global name service. *Proceedings of the 5th. ACM Symposium on the Principles of Distributed Computing*, pages 1–10, August 1986.

[7] L. McLoughlin. The FTP-mirror software. Available from <ftp://src.doc.ic.ac.uk/computing/archiving/mirror>, January 1994.

[8] P. Mockapetris. Domain names - concepts and facilities. Internet Request for Comments RFC 1034, November 1987.

[9] K. Obraczka. *Massively Replicating Services in Wide-Area Internetworks*. PhD thesis, Computer Science Department, University of southern California, December 1994.

[10] D. Oppen and Y. Dalal. The Clearinghouse: A decentralized agent for locating named objects in a distributed environment. *ACM Transactions on Office Information Systems*, 1(3):230–253, July 1983.

[11] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, and G. Rudisin and G. Thiel. LOCUS: A network transparent, high reliability distributed system. *Proc. of the 8th. Symposium on Operating Systems Principles*, pages 169–177, December 1981.

[12] M. Satyanarayanan. Scalable, secure, and highly available distributed file access. *Computer Magazine*, 23(5):9–21, May 1990.