Epidemic Algorithms for Replicated Database Maintenance

Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry

Xerox Palo Alto Research Center

Abstract

When a database is replicated at many sites, maintaining mutual consistency among the sites in the face of updates is a significant problem. This paper describes several randomized algorithms for distributing updates and driving the replicas toward consistency. The algorithms are very simple and require few guarantees from the underlying communication system, yet they ensure that the effect of every update is eventually reflected in all replicas. The cost and performance of the algorithms are tuned by choosing appropriate distributions in the randomization step. The algorithms are closely analogous to epidemics, and the epidemiology literature aids in understanding their behavior. One of the algorithms has been implemented in the Clearinghouse servers of the Xerox Corporate Internet, solving long-standing problems of high traffic and database inconsistency.

August 4, 1987

0. Introduction

Considering a database replicated at many sites in a large, heterogeneous, slightly unreliable and slowly changing network of several hundred or thousand sites, we examine several methods for achieving and maintaining consistency between the sites. Each database update is injected at a single site and must be propagated to all the other sites or supplanted by a later update. The sites can become fully consistent only when all updating activity has stopped and the system has become quiescent. On the other hand, assuming a reasonable update rate, most information at any given site is current. This relaxed form of consistency has been shown to be quite useful in practice [Bi]. Our goal is to design algorithms that are efficient and robust and that scale gracefully as the number of sites increases.

Important factors to be considered in examining algorithms for solving this problem include

- the time required for an update to propagate to all sites, and
- the network traffic generated in propagating a single update. Ideally network traffic is proportional to the size of the update times the number of servers, but some algorithms create much more traffic.

In this paper we present analyses, simulation results and practical experience using several strategies for spreading updates. The methods examined include:

1. *Direct mail:* each new update is immediately mailed from its entry site to all other sites. This is timely and reasonably efficient, but not entirely reliable since individual sites do not always know about all other sites and since mail is sometimes lost.
2. *Anti-entropy:* every site regularly chooses another site at random and by exchanging database contents with it resolves any differences between the two. Anti-entropy is extremely reliable but requires examining the contents of the database and so cannot be used too frequently. Analysis and simulation show that anti-entropy, while reliable, propagates updates much more slowly than direct mail.
3. *Rumor mongering:* sites are initially "ignorant"; when a site receives a new update it becomes a "hot rumor"; while a site holds a hot rumor, it periodically chooses another site at random and ensures that the other site has seen the update; when a site has tried to share a hot rumor with too many sites that have already seen it, the site stops treating the rumor as hot and retains the update without propagating it further. Rumor cycles can be more frequent than anti-entropy cycles because they require fewer resources at each site, but there is some chance that an update will not reach all sites.

Anti-entropy and rumor mongering are both examples of epidemic processes, and results from the theory of epidemics [Ba] are applicable. Our understanding of these mechanisms benefits greatly from the existing mathematical theory of epidemiology, although our goals differ (we would be pleased with the rapid and complete spread of an update). Moreover, we have the freedom to design the epidemic mechanism, rather than the problem of modeling an existing disease. We adopt the terminology of the epidemiology literature and call a site holding an update it is willing to share "infective." A site that has not yet received an update is called "susceptible" and a site that has received an update but is no longer willing to share it is called "removed." Anti-entropy is an example of a "simple epidemic": one in which sites are always either susceptible or infective.

Choosing partners uniformly results in fairly high network traffic, leading us to consider spatial distributions in which the choice tends to favor nearby servers. Analyses and simulations on the actual topology of the Xerox Corporate Internet reveal distributions for both anti-entropy and rumor mongering that converge nearly as rapidly as the uniform distribution while reducing the average and maximum traffic per link. The resulting anti-entropy algorithm has been installed on the Xerox Corporate Internet and has resulted in a significant performance improvement.

We should point out that extensive replication of a database is expensive. It should be avoided whenever possible by hierarchical decomposition of the database or by caching. Even so, the results of our paper are interesting because they indicate that significant replication can be achieved, with simple algorithms, at each level of a hierarchy or in the backbone of a caching scheme.

0.1 Motivation

This work originated in our study of the Clearinghouse servers [Op] on the Xerox Corporate Internet (CIN). The worldwide CIN comprises several hundred Ethernets connected by gateways (on the CIN these are called *internetwork routers*) and phone lines of many different capacities. Several thousand workstations, servers and computing hosts are connected to CIN. A packet enroute from a machine in Japan to one in Europe may traverse as many as 14 gateways and 7 phone lines.

The Clearinghouse service maintains translations from three-level. hierarchical names to machine addresses, user identities, etc. The top two levels of the hierarchy partition the name space into a set of *domains*. Each domain may be stored (replicated) on as few as one or as many as all of the Clearinghouse servers, of which there are several hundred.

Several domains are in fact stored at all Clearinghouse servers in CIN. In early 1986, many of the network's observable performance problems could be traced to traffic created in trying to achieve consistency on these highly replicated domains. As the network size increased, updates to domains stored at even just a few servers propagated very slowly.

When we first approached the problem, the Clearinghouse servers were using both direct mail and anti-entropy. Anti-entropy was run on each domain, in theory, once per day (by each server) between midnight and 6 a.m. local time. In fact, servers often did not complete anti-entropy in the allowed time because of the load on the network.

Our first discovery was that anti-entropy had been followed by a remailing step: the correct database value was mailed to all sites when two anti-entropy participants had previously disagreed. More disagreement among the sites led to much more traffic. For a domain stored at 300 sites, 90,000 mail messages might be introduced each night. This was far beyond the capacity of the network, and resulted in breakdowns in all the network services: mail. file transfer, name lookup, etc.

Since the remailing step was clearly unworkable on a large network our first observation was that it had to be disabled. Further analysis showed that this would be insufficient: certain key links in the network would still be overloaded by anti-entropy traffic. Our explorations of spatial distributions and rumor mongering arose from our attempt to further reduce the network load imposed by the Clearinghouse update process.

0.2 Related Work

The algorithms in this paper are intended to maintain a widely-replicated directory, or name look-up, database. Rather than using transaction based mechanisms that attempt to achieve "one-copy serializability" (for example [Gi]), we use mechanisms that drive the replicas towards eventual agreement. Such mechanisms were apparently first proposed by Johnson et al. [Jo] and have been used in Grapevine [Bi] and Clearinghouse [Op]. Experience with these systems has suggested that some problems remain; in particular, that some updates (with low probability) do not reach all sites. Lampson [La] proposes a hierarchical data structure that avoids high replication, but still requires some replication of each component, say by six to a dozen servers. Primary-site update algorithms for replicated databases have been proposed that synchronize updates by requiring them to be applied to a single site; the update site then takes responsibility for propagating updates to all replicas. The DARPA domain system, for example, employs an algorithm of this sort [Mo]. Primary-site update avoids problems of update distribution addressed by the algorithms described in this paper, but suffers from centralized control.

Two features distinguish our algorithms from previous mechanisms. First, the previous mechanisms depend on various guarantees from underlying communications protocols and on maintaining consistent distributed control structures. For example, in Clearinghouse the initial distribution of updates depends on an underlying guaranteed mail protocol, which in practice fails from time to time due to physical queue overflow, even though the mail queues are maintained on disk storage. Sarin and Lynch [Sa] present a distributed algorithm for discarding obsolete data that depends on guaranteed, properly ordered, message delivery, together with a detailed data structure at each server (of size $O(n^2)$) describing all other servers for the same database. Lampson et al. [La] envision a sweep moving deterministically around a ring of servers, held together by pointers from one server to the next. These algorithms depend upon various mutual consistency properties of the distributed data structure, e.g., in Lampson's algorithm the pointers must define a ring. The algorithms in this paper merely depend on eventual delivery of repeated messages, and do not require data structures at one server describing information held at other servers.

Second, the algorithms described in this paper are randomized; that is, there are points in the algorithm at which each server makes an independent random choice [Ra, Be85]. In distinction, the previous mechanisms are deterministic. For example, in both the anti-entropy and the rumor mongering algorithms, a server randomly chooses a partner. In some versions of the rumor mongering algorithm, a server makes a random choice to remain infective, or become removed. The use of random choice prevents us from making such claims as: "the information will converge in time proportional to the diameter of the network." The best that we can claim is that in the absence of further updates, the probability that the information has not converged is exponentially decreasing with time. On the other hand, we believe that the use of randomized protocols makes our algorithms straightforward to implement correctly using simple data structures.

0.3 Plan of this paper

Section 1 formalizes the notion of a replicated database and presents the basic techniques for achieving consistency. Section 2 describes a technique for deleting items from the database; deletions are more complicated than other changes because the deleted item must be represented by a surrogate until the news of the deletion has spread to all the sites. Section 3 presents simulation and analytical results for non-uniform spatial distributions in the choice of anti-entropy and rumor-mongering partners.

1. Basic Techniques

This section introduces our notion of a replicated database and presents the basic direct mail, anti-entropy and complex epidemic protocols together with their analyses.

1.1 Notation

Consider a network consisting of a set S of n sites, each storing a copy of a database. The database copy at site $s \in S$ is a time-varying partial function

$$s.\text{ValueOf} : K \rightarrow (v : V \times t : T)$$

where K is a set of keys (names), V is a set of values, and T is a set of timestamps. V contains the distinguished element NIL but is otherwise unspecified. T is totally ordered by $<$. We interpret $s.\text{ValueOf}[k] = (\text{NIL}, t)$ to mean that the item identified by k has been deleted from the database. That is, from a database client's perspective, $s.\text{ValueOf}[k] = (\text{NIL}, t)$ is the same as "$s.\text{ValueOf}[k]$ is undefined."

The exposition of the distribution techniques in Sections 1.2 and 1.3 is simplified by considering a database that stores the value and timestamp for only a single name. This is done without loss of generality since the algorithms treat each name separately. So we will say

$$s.\text{ValueOf} \in (v : V \times t : T)$$

i.e., $s.\text{ValueOf}$ is just an ordered pair consisting of a value and a timestamp. As before, the first component may be NIL, meaning the item was deleted as of the time indicated by the second component.

The goal of the update distribution process is to drive the system towards

$$\forall s, s' \in S : s.\text{ValueOf} = s'.\text{ValueOf}$$

There is one operation that clients may invoke to update the database at any given site, s:

$$\text{Update}[v : V] \equiv s.\text{ValueOf} \leftarrow (v, \text{Now}[])$$

where Now is a function returning a globally unique timestamp. One hopes that the timestamps returned by Now[] will be approximately the current Greenwich Mean Time—if not, the algorithms work formally but not practically. The interested reader is referred to the Clearinghouse[Op] and Grapevine[Bi] papers for a further description of the role of the timestamps in building a usable database. For our purposes here, it is sufficient to know that a pair with a larger timestamp will always supersede one with a smaller timestamp.

1.2 Direct Mail

The direct mail strategy attempts to notify all other sites of an update soon after it occurs. The basic algorithm, executed at a site s where an update occurs is:

```
FOR EACH $s' \in S$ DO
    PostMail[to : $s'$, msg : ("Update", $s$.ValueOf)]
    ENDLOOP
```

Upon receiving the message ("Update", (v, t)) site s executes

```
IF $s$.ValueOf.$t < t$ THEN
    $s$.ValueOf $\leftarrow (v, t)$
```

The operation PostMail is expected to be nearly, but not completely, reliable. It queues messages so the sender isn't delayed. The queues are kept in stable storage at the mail server so they are unaffected by server crashes. Nevertheless, PostMail can fail: messages may be discarded when queues overflow or their destinations are inaccessible for a long time. In addition to this basic fallibility of the mail system, the direct mail may also fail when the source site of an update does not have accurate knowledge of S, the set of sites.

In the Grapevine system [Bi] the burden of detecting and correcting failures of the direct mail strategy was placed on the people administering the network. In networks with only a few tens of servers this proved adequate.

Direct mail generates n messages per update; each message traverses all the network links between its source and destination. So in units of (links $\cdot$ messages) the traffic is proportional to the number of sites times the average distance between sites.

1.3 Anti-entropy

The Grapevine designers recognized that handling failures of direct mail in a large network would be beyond people's ability. They proposed *anti-entropy* as a mechanism that could be run in the background to recover automatically from such failures [Bi]. Anti-entropy was not implemented as part of Grapevine, but the design was adopted essentially unchanged for the Clearinghouse. In its most basic form anti-entropy is expressed by the following algorithm periodically executed at each site s:

```
FOR SOME $s' \in S$ DO
```

```
        ResolveDifference[$s, s'$]
        ENDLOOP
```

The procedure ResolveDifference[s, s'] is carried out by the two servers in cooperation. Depending on its design, its effect may be expressed in one of three ways, called *push*, *pull* and *push-pull*:

```
ResolveDifference : PROC[$s, s'$] = { -- push
    IF $s$.ValueOf.$t > s'$.ValueOf.$t$ THEN
        $s'$.ValueOf $\leftarrow$ $s$.ValueOf
    }

ResolveDifference : PROC[$s, s'$] = { -- pull
    IF $s$.ValueOf.$t < s'$.ValueOf.$t$ THEN
        $s$.ValueOf $\leftarrow$ $s'$.ValueOf
    }

ResolveDifference : PROC[$s, s'$] = { -- push-pull
    SELECT TRUE FROM
        $s$.ValueOf.$t > s'$.ValueOf.$t$ $\Rightarrow$ $s'$.ValueOf $\leftarrow$ $s$.ValueOf;
        $s$.ValueOf.$t < s'$.ValueOf.$t$ $\Rightarrow$ $s$.ValueOf $\leftarrow$ $s'$.ValueOf;
        ENDCASE $\Rightarrow$ NULL;
    }
```

For the moment we assume that site s' is chosen uniformly at random from the set S, and that each site executes the anti-entropy algorithm once per period.

It is a basic result of epidemic theory that simple epidemics, of which anti-entropy is one, eventually infect the entire population. The theory also shows that starting with a single infected site this is achieved in expected time proportional to the log of the population size. The constant of proportionality is sensitive to which ResolveDifference procedure is used. For *push*, the exact formula is $\log_2(n) + \ln(n) + O(1)$ for large n [Pi].

It is comforting to know that even if mail fails completely, leaving an update known at only a single site, anti-entropy will eventually distribute it throughout the network. Normally, however, we expect anti-entropy to distribute updates to only a few sites, assuming most sites receive them by direct mail. Thus, it is important to consider what happens when only a few sites remain susceptible. In that case the big difference in behavior is between *push* and *pull*, with *push-pull* behaving essentially like *pull*. A simple deterministic model predicts the observed behavior. Let p_i be the probability of a site's remaining susceptible after the i^{th} cycle of anti-entropy. For *pull*, a site remains susceptible after the $i+1^{st}$ cycle if it was susceptible after the i^{th} cycle and it contacted a susceptible site in the $i+1^{st}$ cycle. Thus, we obtain the recurrence

$$p_{i+1} = (p_i)^2$$

which converges very rapidly to 0 when p_i is small. For *push*, a site remains susceptible after the $i+1^{st}$ cycle if it was susceptible after the i^{th} cycle and no infectious site chose to contact it in the $i+1^{st}$ cycle. Thus, the analogous recurrence relation for *push* is

$$p_{i+1} = p_i \left(1 - \frac{1}{n}\right)^{n(1-p_i)}$$

which also converges to 0, but much less rapidly, since for very small p_i (and large n) it is approximately

$$p_{i+1} = p_i e^{-1}$$

This strongly suggests that in practice, when anti-entropy is used as a backup for some other distribution mechanism such as direct mail, either *pull* or *push-pull* is greatly preferable to *push*, which behaves poorly in the expected case.

As expressed here the anti-entropy algorithm is very expensive, since it involves a comparison of two complete copies of the database, one of which is sent over the network. Normally the copies of the database are in nearly complete agreement, so most of the work is wasted. Given this observation, a possible performance improvement is for each site to maintain a checksum of its database contents, recomputing the checksum incrementally as the database is updated. Sites performing anti-entropy first exchange checksums, comparing their full databases only if the checksums disagree. This scheme saves a great deal of network traffic, assuming the checksums agree most of the time. Unfortunately, it is common for a very recent update to be known by some but not all sites. Checksums at different sites are likely to disagree unless the time required for an update to be sent to all sites is small relative to the expected time between new updates. As the size of the network increases, the time required to distribute an update to all sites increases, so the naive use of checksums described above becomes less and less useful.

A more sophisticated approach to using checksums defines a time window τ large enough that updates are expected to reach all sites within time τ. As in the naive scheme, each site maintains a checksum of its database. In addition, the site maintains a *recent update list*, a list of all entries in its database whose ages (measured by the difference between their timestamp values and the site's local clock) are less than τ. Two sites s and s' perform anti-entropy by first exchanging recent update lists, using the lists to update their databases and checksums, and then comparing checksums. Only if the checksums disagree do the sites compare their entire databases.

Exchanging recent update lists before comparing checksums ensures that if one site has received a change or delete recently, the corresponding obsolete entry does not contribute to the other site's checksum. Thus, the checksum comparison is very likely to succeed, making a full database comparison unnecessary. In that case, the expected traffic generated by an anti-entropy comparison is just the expected size of the recent update list, which is bounded by the expected number of updates occurring on the network in time τ. Note that the choice of τ to exceed the expected distribution time for an update is critical; if τ is chosen poorly, or if growth of the network drives the expected update distribution time above τ, checksum comparisons will usually fail and network traffic will rise to a level slightly higher than what would be produced by anti-entropy without checksums.

A simple variation on the above scheme, which does not require *a priori* choice of a value for τ, can be used if each site can maintain an inverted index of its database by timestamp. Two sites perform anti-entropy by exchanging updates in reverse timestamp order, incrementally recomputing their checksums, until agreement of the checksums is achieved. While it is nearly ideal from the standpoint of network traffic, this scheme (which we will hereafter refer to as *peel back*) may not be desirable in practice because of the expense of maintaining an additional inverted index at each site.

1.4 Complex Epidemics

As we have seen already, direct mailing of updates has several problems: it can fail because of message loss, or because the originator has incomplete information about other database sites, and it introduces an $O(n)$ bottleneck at the originating site. Some of these problems would be remedied by a broadcast mailing mechanism, but most likely that mechanism would itself depend on distributed information. The epidemic mechanisms we are about to describe do avoid these problems, but they have a different, explicit probability of failure that must be studied carefully with analysis and simulations. Fortunately this probability of failure can be made arbitrarily small. We refer to these mechanisms as "complex" epidemics only to distinguish them from anti-entropy which is a simple epidemic; complex epidemics still have simple implementations.

Recall that with respect to an individual update, a database is either *susceptible* (it does not know the update), *infective* (it knows the update and is actively sharing it with others), or *removed* (it knows the update but is not spreading it). It is a relatively easy matter to implement this so that a sharing step does not require a complete pass through the database. The sender keeps a list of infective updates, and the recipient tries to insert each update into its own database and adds all new updates to its infective list. The only complication lies in deciding when to remove an update from the infective list.

Before we discuss the design of a "good" epidemic, let's look at one example, usually called rumor spreading in the epidemiology literature.

Rumor spreading is based on the following scenario: There are n individuals, initially inactive (susceptible). We plant a rumor with one person who becomes active (infective), phoning other people at random and sharing the rumor. Every person hearing the rumor also becomes active and likewise shares the rumor. When an active individual makes an unnecessary phone call (the recipient already knows the rumor), then with probability $1/k$ the active individual loses interest in sharing the rumor (the individual becomes removed). We would like to know how fast the system converges to an inactive state (a state in which no one is infective) and the percentage of people who know the rumor (are removed) when this state is reached.

Following the epidemiology literature, rumor spreading can be modeled deterministically with a pair of differential equations. We let s, i, and r represent the fraction of individuals susceptible, infective, and removed respectively, so that $s + i + r = 1$:

$$\begin{aligned} \frac{ds}{dt} &= -si \\ \frac{di}{dt} &= +si - \frac{1}{k}(1-s)i \end{aligned} \qquad (*)$$

The first equation suggests that susceptibles will be infected according to the product si. The second equation has an additional term for loss due to individuals making unnecessary phone calls. A third equation for r is redundant.

A standard technique for dealing with equations like (*) is to take the ratio [Ba]. This eliminates t and lets us solve for i as a function of s:

$$\frac{di}{ds} = -\frac{k+1}{k} + \frac{1}{ks}$$

$$i(s) = -\frac{k+1}{k}s + \frac{1}{k}\log s + c$$

where c is a constant of integration that can be determined by the initial conditions: $i(1-\epsilon) = \epsilon$. For large n, ϵ goes to zero, giving:

$$c = \frac{k+1}{k}$$

and a solution of

$$i(s) = \frac{k+1}{k}(1-s) + \frac{1}{k}\log s.$$

The function $i(s)$ is zero when

$$s = e^{-(k+1)(1-s)}.$$

This is an implicit equation for s, but the dominant term shows s decreasing exponentially with k. Thus increasing k is an effective way of insuring that almost everybody hears the rumor. For example, at $k = 1$ this formula suggests that 20% will miss that rumor, while at $k = 2$ only 6% will miss it.

Variations

So far we have seen only one complex epidemic, based on the rumor spreading technique. In general we would like to understand how to design a "good" epidemic, so it is worth pausing now to review the criteria used to judge epidemics. We are principally concerned with:

1. **Residue.** This is the value of s when i is zero, that is, the remaining susceptibles when the epidemic finishes. We would like the residue to be as small as possible, and, as the above analysis shows, it is feasible to make the residue arbitrarily small.
2. **Traffic.** Presently we are measuring traffic in database updates sent between sites, without regard for the topology of the network. It is convenient to use an average, the number of messages sent from a typical site:

$$m = \frac{\text{Total update traffic}}{\text{Number of sites}}$$

In section 3 we will refine this metric to incorporate traffic on individual links.

3. **Delay.** There are two interesting times. The average delay is the difference between the time of the initial injection of an update and the arrival of the update at a given site, averaged over all sites. We will refer to this as t_{ave}. A similar quantity, t_{last}, is the delay until the reception by the last site that will receive the update during the epidemic. Update messages may continue to appear in the network after t_{last}, but they will never reach a susceptible site. We found it necessary to introduce two times because they often behave differently, and the designer is legitimately concerned about both times.

Next, let us consider a few simple variations of rumor spreading. First we will describe the practical aspects of the modifications, and later we will discuss residue, traffic, and delay.

Blind vs. Feedback. The rumor example used feedback from the recipient; a sender loses interest only if the recipient already knows the rumor. A blind variation loses interest with probability $1/k$ regardless of the recipient. This obviates the need for the bit vector response from the recipient.

Counter vs. Coin. Instead of losing interest with probability $1/k$ we can use a counter, so that we lose interest only after k unnecessary contacts. The counters require keeping extra state for elements on the infective lists. Note that we can combine counter with blind, remaining infective for k cycles independent of any feedback.

A surprising aspect of the above variations is that they share the same fundamental relationship between traffic and residue:

$$s = e^{-m}$$

This is relatively easy to see by noticing that there are nm updates sent and the chance that a single site misses all these updates is $(1 - 1/n)^{nm}$. (Since m is not constant this relationship depends on the moments around the mean of the distribution of m going to zero as $n \rightarrow \infty$, a fact that we have observed empirically, but have not proven.) Delay is the the only consideration that distinguishes the above possibilities: simulations indicate that counters and feedback improve the delay, with counters playing a more significant rôle than feedback.

Table 1. Performance of an epidemic on 1000 sites using feedback and counters.

Counter	Residue	Traffic	Convergence	
k	s	m	t_{ave}	t_{last}
1	0.176	1.74	11.0	16.8
2	0.037	3.30	12.1	16.9
3	0.011	4.53	12.5	17.4
4	0.0036	5.64	12.7	17.5
5	0.0012	6.68	12.8	17.7

Table 2. Performance of an epidemic on 1000 sites using blind and coin.

Coin	Residue	Traffic	Convergence	
k	s	m	t_{ave}	t_{last}
1	0.960	0.04	19	38
2	0.205	1.59	17	33
3	0.060	2.82	15	32
4	0.021	3.91	14.1	32
5	0.008	4.95	13.8	32

Push vs. Pull. So far we have assumed that the all sites would randomly choose destination sites and *push* infective updates to the destinations. The *push* vs. *pull* distinction made already for anti-entropy can be applied to rumor mongering as well. *Pull* has some advantages, but the primary practical consideration is the rate of update injection into the distributed database. If there are numerous independent updates a *pull* request is likely to find a source with a non-empty rumor list, triggering useful information flow. By contrast, if the database is quiescent the *push* algorithm ceases to introduce traffic overhead, while the *pull* variation continues to inject fruitless requests for updates. Our own CIN application has a high enough update rate to warrant the use of *pull.*

The chief advantage of *pull* is that it does significantly better than the $s = e^{-m}$ relationship of *push.* Here the blind vs. feedback and counter vs. coin variations are important. Simulations indicate that the counter and feedback variations improve residue, with counters being more important than feedback. We have a recurrence relation modeling the counter with feedback case that exhibits $s = e^{-\Theta(m^3)}$ behavior.

Table 3. Performance of a pull epidemic on 1000 sites using feedback and counters†.

Counter	Residue	Traffic	Convergence	
k	s	m	t_{ave}	t_{last}
1	3.1×10^{-2}	2.70	9.97	17.63
2	5.8×10^{-4}	4.49	10.07	15.39
3	4.0×10^{-6}	6.09	10.08	14.00

† If more than one recipient pulls from a site in a single cycle then at the end of the cycle the effects on the counter are as follows: if any recipient needed the update then the counter is reset; if all recipients did not need the update then one is added to the counter.

Minimization. It is also possible to make use of the counters of both parties in an exchange to make the removal decision. The idea is to use a push and a pull together, and if both sites already know the update, then only the site with the smaller counter is incremented (in the case of equality both must be incremented). This requires sending the counters over the network, but it results in the smallest residue we have seen so far.

Connection Limit. It is unclear whether connection limitation should be seen as a difficulty or an advantage. So far we have ignored connection limitations. Under the *push* model, for example, we have assumed that a site can become the recipient of more than one push in a single cycle, and in the case of *pull* we have assumed that a site can service an unlimited number of requests. Since we plan to run the rumor spreading algorithms frequently, realism dictates that we use a connection limit. The connection limit affects the *push* and *pull* mechanism differently: *pull* gets significantly worse, and, paradoxically, *push* gets significantly better.

To see why *push* gets better, assume that the database is nearly quiescent, that is, only one update is being spread, and that the connection limit is one. If two sites contact the same recipient then one of the contacts is rejected. The recipient still gets the update, but with one less unit of traffic. (We have chosen to measure traffic only in terms of updates sent. Some network activity arises in attempting the rejected connection, but this is less than that involved in transmitting the update. We have, in essence, shortened a connection that was otherwise useless). How many connections are rejected? Since an epidemic grows exponentially, we assume most of the traffic occurs at the end when nearly everybody is infective and the probability of rejection is close to e^{-1}. So we would expect that *push* with connection limit one would behave like:

$$s = e^{-\lambda m} \qquad \lambda = \frac{1}{1 - e^{-1}}$$

Simulations indicate that the counter variations are closest to this behavior (counter with feedback being the most effective). The coin variations do not fit the above assumptions, since they do not have most of their traffic occurring when nearly all sites are infective. Nevertheless they still do better than $s = e^{-m}$. In all variations, since *push* on a nearly quiescent network works best with a connection limit of 1 it seems worthwhile to enforce this limit even if more connections are possible.

Pull gets worse with a connection limit, because its good performance depends on every site being a recipient in every cycle. As soon as there is a finite connection failure probability δ, the asymptotics of *pull* changes. Assuming, as before, that almost all the traffic occurs when nearly all sites are infective, then the chance of a site missing an update during this active period is roughly:

$$s = \delta^m = e^{-\lambda m} \qquad \lambda = -\ln \delta$$

Fortunately, with only modest sized connection limits, the probability of failure becomes extremely small, since the chance of a site having j connections in a cycle is $e^{-1}/j!$.

Hunting. If a connection is rejected then the choosing site can "hunt" for alternate sites. Hunting is relatively inexpensive and seems to improve all connection limited cases. In the extreme case, a connection limit of 1 with infinite hunt limit results in a complete permutation. Push and pull then become equivalent, and the residue is very small.

1.5 Backing Up a Complex Epidemic with Anti-entropy

We have seen that a complex epidemic algorithm can spread updates rapidly with very low network traffic. Unfortunately, a complex epidemic can fail; that is, there is a nonzero probability that the number of infective sites will fall to zero while some sites remain susceptible. This event can be made extremely unlikely; nevertheless, if it occurs, the system will be in a stable state in which an update is known by some, but not all, sites. To eliminate this possibility, anti-entropy can be run infrequently to back up a complex epidemic, just as it is used in the Clearinghouse to back up direct mail. This ensures with probability 1 that every update eventually reaches (or is superseded at) every site.

When anti-entropy is used as a backup mechanism, there is a significant advantage in using a complex epidemic such as rumor mongering rather than direct mail for initial distribution of updates. Consider what should be done if a missing update is discovered when two sites perform anti-entropy. A conservative response would be to do nothing (except make the databases of the two sites consistent, of course), relying on anti-entropy eventually to deliver the update to all sites. A more aggressive response would be to redistribute the update, using whatever mechanism is used for the initial distribution; e.g., mail it to all other sites or make it a hot rumor again. The conservative approach is adequate in the usual case that the update is known at all but a few sites. However, to deal with the occasional complete failure of the initial distribution, a redistribution step is desirable.

Unfortunately, the cost of redistribution by direct mail can be very high. The worst case occurs when the initial distribution manages to deliver an update to approximately half the sites, so that on the next anti-entropy cycle each of $O(n)$ sites attempts to redistribute the update by mailing it to all n sites, generating $O(n^2)$ mail messages. The Clearinghouse originally did redistribution, but we were forced to eliminate it because of its high cost.

Using rumor mongering instead of direct mail would greatly reduce the expected cost of redistribution. Rumor mongering is ideal for the simple case in which only a few sites fail to receive an update, since a single hot rumor that is already known at nearly all sites dies out quickly without generating much network traffic. It also behaves well in the worst-case situation mentioned above: if an update is distributed to approximately half the sites, then $O(n)$ copies of the update will be introduced as hot rumors in the next round of anti-entropy. This actually generates less network traffic than introducing the rumor at a single site.

This brings us to an interesting way of combining rumor mongering and anti-entropy. Recall that we described earlier a variation of anti-entropy called *peel back* that required an additional index of the database in time stamp order. The idea was to have sites exchange updates in reverse timestamp order until they reached checksum agreement on their entire databases. *Peel back* and rumor mongering can be combined by keeping the updates in a doubly linked list rather than a search tree. Whereas before we needed a search tree to maintain reverse timestamp order, we now use a doubly linked list to maintain a local activity order: sites send updates according to their local list order, and they receive the usual rumor feedback that tells them when an update was useful. The useful updates are moved to the front of their respective lists, while the useless updates slip gradually deeper in the lists. This combined variation is better than *peel back* alone because: 1) it avoids the extra index, and 2) it behaves well when a network partitions and rejoins. It is better than rumor mongering alone because it has no failure probability. In practice one would probably not send one update at a time, but batch together a collection of updates from the head of the list. This set is analogous to the hot rumor list of rumor mongering, but membership is no longer a binary matter since if the first batch fails to reach checksum agreement, then more batches are sent. If necessary, any update in the database can become a hot rumor again.

2. Deletion and Death Certificates

Using either anti-entropy or rumor mongering, we cannot delete an item from the database simply by removing a local copy of the item, expecting the *absence* of the item to spread to other sites. Just the opposite will happen: the propagation mechanism will spread old copies of the item from elsewhere in the database back to the site where we have deleted it. Unless we can simultaneously delete all copies of an obsolete item from the database, it will eventually be "resurrected" in this way.

To remedy this problem we replace deleted items with *death certificates*, which carry time stamps and spread like ordinary data. During propagation, when old copies of deleted items meet death certificates, the old items are removed. If we keep a death certificate long enough it eventually cancels all old copies of its associated item. Unfortunately, this does not completely solve the problem. We still must decide when to delete the death certificates themselves or they will ultimately consume all available storage at the sites.

One strategy is to retain each death certificate until it can be determined that every site has received it. Sarin and Lynch [Sa] describe a protocol for making this determination, based on the distributed snapshot algorithm of Chandy and Lamport [Ch]. Separate protocols are needed for adding and removing sites (Sarin and Lynch do not describe the site addition protocol in any detail). If any site fails permanently between the creation of a death certificate and the completion of the distributed snapshot, that death certificate cannot be deleted until the site removal protocol has been run. For a network of several hundred sites this fact can be quite significant. In our experience, there is a fairly high probability that at any time some site will be down (or unreachable) for hours or even days, preventing the distributed snapshot or site deletion algorithm from completing.

A much simpler strategy is to hold death certificates for some fixed time, such as 30 days, and then discard them. With this scheme, we run the risk of obsolete items older than the threshold being resurrected, as described above. There is a clear tradeoff between the amount of space devoted to death certificates and the risk of obsolete items being resurrected: increasing the time threshold reduces the risk but increases the amount of space consumed by death certificates.

2.1 Dormant Death Certificates

There is a distributed way of extending the time threshold back much further than the space on any one server would permit. This scheme, which we call *dormant death certificates*, is based on the following observation. If a death certificate is older than the expected time required to propagate it to all sites, then the existence of an obsolete copy of the corresponding data item anywhere in the network is unlikely. We can delete very old death certificates at most sites, retaining "dormant" copies at only a few sites. When an obsolete update encounters a dormant death certificate, the death certificate can be "awakened" and propagated again to all sites. This operation is expensive, but it will occur infrequently. In this way we can ensure that if a death certificate is present at *any* site in the network, resurrection of the associated data item will not persist for any appreciable time. Note the analogy to an immune reaction, with the awakened death certificates behaving like antibodies.

The implementation uses two thresholds, τ_1 and τ_2, and attaches a list of r retention site names to each death certificate. When a death certificate is created, its retention sites are chosen at random. The death certificate is propagated by the same mechanism used for ordinary updates. Each server retains all death certificates timestamped within τ_1 of the current time. Once τ_1 is reached, most servers delete the death certificate, but every server on the death certificate's retention site list saves a dormant copy. Dormant death certificates are discarded when $\tau_1 + \tau_2$ is reached.

(For simplicity we ignore the differences between sites' local clocks. It is realistic to assume that the clock synchronization error is at most $\epsilon << \tau_1$. This has no significant effect on the arguments.)

Dormant death certificates will occasionally be lost due to permanent server failures. For example, after one server half-life the probability that all servers holding dormant copies of a given death certificate have failed is 2^{-r}. The value of r can be chosen to make this probability acceptably small.

To compare this scheme to a scheme using a single fixed threshold τ, assume that the rate of deletion is steady over time. For equal space usage, assuming $\tau > \tau_1$, we obtain

$$\tau_2 = (\tau - \tau_1)n/r$$

That is, there is $O(n)$ improvement in the amount of history we can maintain. In our existing system, this would enable us to increase the effective history from 30 days to several years.

At first glance, the dormant death certificate algorithm would appear to scale indefinitely. Unfortunately, this is not the case. The problem is that the expected time to propagate a new death certificate to all sites, which grows with n, will eventually exceed the threshold value τ_1, which does not grow with n. While the propagation time is less than τ_1 it is seldom necessary to reactivate old death certificates; after the propagation time grows beyond τ_1 reactivation of old death certificates becomes more and more frequent. This introduces additional load on the network for propagating the old death certificates, thereby further degrading the propagation time. The ultimate result is catastrophic failure. To avoid such a failure, system parameters must be chosen to keep the expected propagation time below τ_1. As described above, the time required for a new update to propagate through the network using anti-entropy is expected to be $O(\log n)$. This implies that for sufficiently large networks τ_1, and hence the space required at each server, eventually must grow as $O(\log n)$.

2.2. Anti-Entropy with Dormant Death Certificates

If anti-entropy is used for distributing updates, dormant death certificates should not normally be propagated during anti-entropy exchanges. Whenever a dormant death certificate encounters an obsolete data item, however, the death certificate must be "activated" in some way, so it will propagate to all sites and cancel the obsolete data item.

The obvious way to activate a death certificate is to set its timestamp forward to the current clock value. This approach might be acceptable in a system that did not allow deleted data items to be "reinstated." In general it is incorrect, because somewhere in the network there could be a legitimate update with a timestamp between the original and revised timestamps of the death certificate (e.g. an update reinstating the deleted item). Such an update would incorrectly be cancelled by the reactivated death certificate.

To avoid this problem, we store a second timestamp, called the *activation timestamp*, with each death certificate. Initially the ordinary and activation timestamps are the same. A death certificate still cancels a corresponding data item if its ordinary timestamp is greater than the timestamp of the item. However, the activation timestamp controls whether a death certificate is considered dormant and how it propagates. Specifically, a death certificate is deleted (or considered dormant by a site on its site list) when its activation timestamp grows older than τ_1; a dormant death certificate is deleted when its activation timestamp grows older than $\tau_1 + \tau_2$; and a death certificate is propagated by anti-entropy only if it is not dormant. To reactivate a death certificate, we set its activation timestamp to the current time, leaving its ordinary timestamp unchanged. This has the desired effect of propagating the reactivated death certificate without cancelling more recent updates.

2.3. Rumor Mongering with Dormant Death Certificates

The activation timestamp mechanism described above for anti-entropy works equally well if rumor mongering is used for distributing updates. Each death certificate is created as an active rumor. Eventually it propagates through the network and becomes inactive at all sites. Whenever a dormant death certificate encounters an obsolete data item at some site, the death certificate is activated by setting its activation timestamp to the current time. In addition, it is made an active rumor again. The normal rumor mongering mechanism then distributes the reactivated death certificate throughout the network.

3. Spatial Distributions

Up to this point we have regarded the network as uniform, but in reality the cost of sending an update to a nearby site is much lower than the cost of sending the update to a distant site. To reduce communication costs, we would like our protocols to favor nearby neighbors, but there are drawbacks to too much local favoritism. It is easiest to begin exploring this tradeoff on a line.

Assume, for the moment, that the database sites are arranged on a linear network, and that each site is one link from its nearest neighbors. If we were using only nearest neighbors for anti-entropy exchanges, then the traffic per link per cycle would be $O(1)$, but it would take $O(n)$ cycles to spread an update. At the other extreme, if we were using uniform random connections, then the average distance of a connection would be $O(n)$, so that even though the convergence would be $O(\log n)$ the traffic per link per cycle would be $O(n)$. In general, let the probability of connecting to a site at distance d be proportional to d^{-a}, where a is a parameter to be determined. Analysis shows that the expected traffic per link is:

$$T(n) = \begin{cases} O(n), & a < 1; \\ O(n/\log n), & a = 1; \\ O(n^{2-a}), & 1 < a < 2; \\ O(\log n), & a = 2; \\ O(1), & a > 2 \end{cases}$$

Convergence times for the d^{-a} distribution are much harder to compute exactly. Informal equations and simulations suggest that they follow the reverse pattern: for $a > 2$ the convergence is polynomial in n, and for $a < 2$ the convergence is polynomial in $\log n$. This strongly suggests using a d^{-2} distribution for spreading updates on a line, since both traffic and convergence would scale well as n goes to infinity.

A realistic network bears little resemblance to the linear example used above (surprisingly, small sections of the CIN are in fact linear, but the bulk of the network is more highly connected), so it is not immediately obvious how to generalize the d^{-2} distribution. The above reasoning can be generalized to higher dimensional rectilinear meshes of sites, suggesting good convergence (polynomial in $\log n$) with distributions as tight as d^{-2D}, where D is the dimension of the mesh. Moreover, the traffic drops to $O(\log n)$ as soon as the distribution is $d^{-(D+1)}$, so we have a broader region of good behavior, but it is dependent on the dimension of the mesh, D. This observation led us to consider letting each site s independently choose connections according to a distribution that is a function of $Q_s(d)$, where $Q_s(d)$ is the cumulative number of sites at distance d or less from s. On a D-dimensional mesh, $Q_s(d)$ is $\Theta(d^D)$, so that a distribution like $1/Q_s(d)^2$ is $\Theta(d^{-2D})$, regardless of the dimension of the mesh. We conjectured that the $Q_s(d)$ function's ability to adapt to the dimension of a mesh would make it work well in an arbitrary network, and that the asymptotic properties would make distributions between $1/dQ_s(d)$ and $1/Q_s(d)^2$ have the best scaling behavior. The next section describes further practical considerations for the choice of distribution, and our experience with $1/Q_s(d)^2$.

3.1 Spatial Distributions and Anti-Entropy

We argued above that a nonuniform spatial distribution can significantly reduce the traffic generated by anti-entropy without unacceptably increasing its convergence time. The network topologies considered were very regular: D-dimensional rectilinear grids.

Use of a nonuniform distribution becomes even more attractive when we consider the actual CIN topology. In particular, the CIN includes small sets of critical links, such as a pair of transatlantic links that are the only routes connecting n_1 sites in Europe to n_2 sites in North America. Currently n_1 is a few tens, and n_2 is several hundred. Using a uniform distribution, the expected number of conversations on these transatlantic links per round of anti-entropy is $2n_1n_2/(n_1 + n_2)$. This is about 80 conversations, shared between the two links. By comparison, when averaged over all links, the expected traffic per link per cycle is less than 6 conversations. It is the unacceptably high traffic on critical links, rather than the average traffic per link, that makes uniform anti-entropy too costly for use in our system. This observation originally inspired our study of nonuniform spatial distributions.

To learn how network traffic could be reduced, we performed extensive simulations of anti-entropy behavior using the actual CIN topology and a number of different spatial distributions.

Preliminary results indicated that distributions parameterized by $Q_s(d)$ adapt well to the "local dimension" of the network as suggested in Section 3, and perform significantly better than distributions with any direct dependence on d. In particular, $1/Q_s(d)^2$ outperforms $1/dQ_s(d)$. The results using spatial distributions of the form $Q_s(d)^{-a}$ for anti-entropy were very encouraging. However, early simulations of rumor mongering uncovered a few problem spots in the CIN topology when spatial distributions were used.

After examining these results, we developed a slightly different class of distributions that are less sensitive to sudden increases in $Q_s(d)$. These distributions proved to be more effective for both anti-entropy and rumor mongering on the CIN topology. Informally, let each site s build a list of the other sites sorted by their distance from s, and then select anti-entropy exchange partners from the sorted list according to a function $f(i)$. This function gives the probability of choosing a site as a function of its position i in the list. For f we can use the spatial distribution function that would be used on a uniform linear network. Of course, two sites at the same distance from s in the real network (but at different positions in the list) should not be selected with different probabilities. We can arrange for this by averaging the probabilities of selecting equidistant sites; i.e., by selecting each of the sites at distance d with probability proportional to

$$p(d) = \frac{\sum_{i=Q(d-1)+1}^{Q(d)} f(i)}{Q(d) - Q(d-1)}$$

Letting $f = i^{-a}$, where a is a parameter to be determined, and approximating the summation by an integral, we obtain

$$p(d) \approx \frac{Q(d-1)^{-a+1} - Q(d)^{-a+1}}{Q(d) - Q(d-1)} \qquad (3.1.1)$$

Note for $a = 2$ the right side of (3.1.1) reduces to

$$1/(Q_s(d-1)Q_s(d)),$$

which is $O(d^{-2D})$ on a D-dimensional mesh, as discussed in Section 3.

Simulation results reported in this and the next section use either uniform distributions or distributions of the above form for selected values of a.

Table 4 summarizes results for the CIN topology using push-pull anti-entropy with no connection limit. This table represents 250 runs, each propagating a single update introduced at a randomly-chosen site. The "Compare Traffic" figures represent number of anti-entropy comparisons per cycle, averaged over all network links and separately for the transatlantic link to Bushey, England. "Update Traffic" represents the total number of anti-entropy exchanges in which the update had to be sent from one site to the other. (The distinction between compare traffic and update traffic can be significant if checksums are used for database comparison, as discussed in Section 1.3).

Table 4. Simulation results for anti-entropy, no connection limit.

Spatial Distribution	t_{last}	t_{ave}	Compare Traffic		Update Traffic	
			Average	Bushey	Average	Bushey
uniform	7.81	5.27	5.87	75.74	5.85	74.43
$a = 1.2$	10.04	6.29	2.00	11.19	2.61	17.52
$a = 1.4$	10.31	6.39	1.93	8.77	2.49	14.10
$a = 1.6$	10.94	6.70	1.71	5.72	2.27	10.88
$a = 1.8$	11.97	7.21	1.52	3.74	2.07	7.68
$a = 2.0$	13.32	7.76	1.36	2.38	1.89	5.87

Two points are worth mentioning:

1. Comparing the $a = 2$ results with the uniform case, convergence time t_{last} degrades by less than a factor of 2, while average traffic per round improves by a factor of more than 4. Arguably, we could afford to perform anti-entropy twice as frequently with the nonuniform distribution, thereby getting an equivalent convergence rate with a factor of two improvement in average traffic.
2. Again comparing the $a = 2$ results with the uniform case, the compare traffic on the transatlantic link falls from 75.74 to 2.38, a factor of more than 30. Traffic on this link is now less than twice the mean. We view this as the most important benefit of nonuniform distributions for the CIN topology. Using this distribution, anti-entropy exchanges do not overload critical network links.

The results in Table 4 assume no connection limit. This assumption is quite unrealistic – the actual Clearinghouse servers can support only a few anti-entropy connections at once. Table 5 gives simulation results under the most pessimistic assumption: connection limit of 1 and hunt limit 0. As above, the table represents 250 runs, each propagating a single update introduced at a randomly-chosen site.

Table 5. Simulation results for anti-entropy, connection limit 1.

Spatial Distribution	t_{last}	t_{ave}	Compare Traffic		Update Traffic	
			Average	Bushey	Average	Bushey
uniform	11.00	6.97	3.71	47.54	5.83	75.17
$a = 1.2$	16.89	9.92	1.14	6.39	2.69	18.03
$a = 1.4$	17.34	10.15	1.08	4.68	2.55	13.68
$a = 1.6$	19.06	11.06	0.94	2.90	2.32	10.20
$a = 1.8$	21.46	12.37	0.82	1.68	2.12	7.03
$a = 2.0$	24.64	14.14	0.72	0.94	1.94	4.85

Note:

3. The Compare Traffic figures in Table 5 are significantly lower than those in Table 4, reflecting the fact that fewer successful connections are established per cycle. The convergence times in Table 5 are correspondingly higher than those in Table 4. These effects are more pronounced with the less uniform distributions.

4. The total compare traffic (which is the product of convergence time and Compare Traffic) does not change significantly when the connection limit is imposed; the compare traffic per cycle decreases, while the number of cycles increases.

To summarize: using a spatial distribution with anti-entropy can significantly reduce traffic on critical links that would be "hot spots" if a uniform distribution were used. The most pessimistic connection limit slows convergence but does not significantly change the total amount of traffic generated in distributing the update; it just slows down distribution of the update somewhat.

Based on our early simulation results, a nonuniform anti-entropy algorithm using a $1/Q(d)^2$ distribution was implemented as part of an internal release of the Clearinghouse service. The new release has now been installed on the entire CIN and has produced dramatic improvements both in network load generated by the Clearinghouse servers and in consistency of their databases.

3.2 Spatial Distributions and Rumors

Because anti-entropy effectively examines the entire database on each exchange, it is very robust. For example, consider a spatial distribution such that for every pair (s, s') of sites there is a nonzero probability that s will choose to exchange data with s'. It is easy to show that anti-entropy converges with probability 1 using such a distribution, since under those conditions every site eventually exchanges data directly with every other site.

Rumor mongering, on the other hand, runs to quiescence – every rumor eventually becomes inactive, and if it has not spread to all sites by that time it will never do so. Thus it is not sufficient that the site holding a new update eventually contact every other site; the contact must occur "soon enough," or the update can fail to spread to all sites. This suggests that rumor mongering might be less robust than anti-entropy against changes in spatial distribution and network topology. In fact we have found this to be the case.

Rumor mongering has a parameter that can be adjusted: as the spatial distribution is made less uniform, we can increase the value of k in the rumor mongering algorithm to compensate. For *push-pull* rumor mongering on the CIN topology, this technique is quite effective. We have simulated the (Feedback, Counter, *push-pull*, No Connection Limit) variation of rumor mongering described in Section 1.4, using increasingly nonuniform spatial distributions. We found that once k was adjusted to give 100% distribution in each of 200 trials, that the traffic and convergence times were nearly identical to the results in Table 4. That is, there is a small finite k that achieves the same results as $k = \infty$.

The fact that rumor mongering achieves the same results as Table 4 is a stronger fact than it might seem at first. The convergence times in Table 4 are given in cycles; however, the cost of an anti-entropy cycle is a function of the database size, while the cost of a rumor mongering cycle is a function of the number of active rumors in the system. Similarly, the compare traffic figures in Table 4 have different meanings when interpretted as pertaining to anti-entropy or rumor mongering. In general rumor mongering comparisons should be cheaper than anti-entropy comparisons, since they need to examine only the list of hot rumors.

We conclude that a nonuniform spatial distribution can produce a worthwhile improvement in the performance of *push-pull* rumor mongering, particularly the traffic on critical network links.

The *push* and *pull* variants of rumor mongering appear to be much more sensitive than *push-pull* to the combination of nonuniform spatial distribution and arbitrary network topology. Using (Feedback, Counter, *push*, No Connection Limit) rumor mongering and the spatial distribution (3.1.1) with $a = 1.2$, the value of k required to achieve 100% distribution in each of 200 simulation trials was 36; convergence times were correspondingly high. Simulations for larger a values did not complete overnight, so the attempt was abandoned.

We do not yet fully understand this behavior, but two simple examples illustrate the kind of problem that can arise. Both examples rely on having isolated sites, fairly distant from the rest of the network.

Figure 1

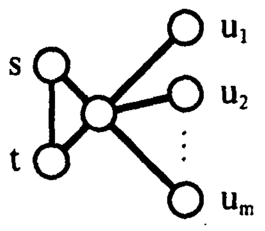
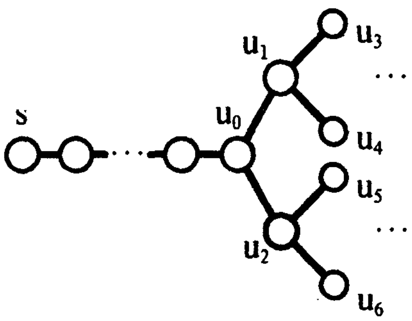


Figure 2



First, consider a network like the one shown in Figure 1, in which two sites s and t are near each other and slightly farther away from sites $u_1, \ldots, u_m$, which are all equidistant from s and equidistant from t. (It is easy to construct such a network, since we are not required to have a database site at every network node). Suppose s and t use a $Q_s(d)^{-2}$ distribution to select partners. If m is larger than k there is a significant probability that s and t will select each other as partners on k consecutive cycles. If *push* rumor mongering is being used and an update has been introduced at s or t, this results in a catastrophic failure - the update is delivered to s and t; after k cycles it ceases to be a hot rumor without being delivered to any other site. If *pull* is being used and the update is introduced in the main part of the network, there is a significant chance that each time s or t contacts a site u_i, that site either does not yet know the update or has known it so long that it is no longer a hot rumor; the result is that neither s nor t ever learns of the update.

As a second example, consider a network like the one shown in Figure 2. Site s is connected to the root u_0 of a complete binary tree of $n - 1$ sites, with the distance from s to u_0 greater than the height of the tree. As above, suppose all sites use a $Q_s(d)^{-2}$ distribution to select rumor mongering partners. If n is large relative to k, there is a significant probability that in k consecutive cycles no site in the tree will attempt to contact s. Using *push* rumor mongering, if an update has been introduced at some node of the tree, the update may fail to be delivered to s until it has ceased to be a hot rumor anywhere.

More study will be needed before the relation between spatial distribution and irregular network topology is fully understood. The examples and simulation results above emphasize the need to back up rumor mongering with anti-entropy to guarantee complete coverage. Given such a guarantee, however, *push-pull* rumor mongering with a spatial distribution appears quite attractive for the CIN.

4. Conclusions

It is possible to replace complex deterministic algorithms for replicated database consistency with simple randomized algorithms that require few guarantees from the underlying communication system. The randomized anti-entropy algorithm has been implemented on the Xerox Corporate Internet providing impressive performance improvements both in achieving consistency and reducing network overhead traffic. By using a well chosen spatial distribution for selecting anti-entropy partners, the implemented algorithm reduces average link traffic by a factor of more than 4 and traffic on certain critical links by a factor of 30 when compared with an algorithm using uniform selection of partners.

The observation that anti-entropy behaves like a simple epidemic led us to consider other epidemic-like algorithms such as rumor mongering, which shows promise as an efficient replacement for the initial mailing step for distributing updates. A backup anti-entropy scheme easily spreads the updates to the few sites that do not receive them as a rumor. In fact, it is possible to combine a *peel back* version of anti-entropy with rumor mongering, so that rumor mongering never fails to spread an update completely.

Neither the epidemic algorithms nor the direct mail algorithms can correctly spread the absence of an item without assistance from death certificates. There is a trade-off between the retention time for death certificates, the storage space consumed, and the likelihood of old data undesirably reappearing in the database. By retaining dormant death certificates at a few sites we can significantly improve the network's immunity to obsolete data at a reasonable storage cost.

There are more issues to be explored. Pathological network topologies present performance problems. One solution would be to find algorithms that work well with all topologies; failing this, one would like to characterize the pathological topologies. Work still needs to be done on the analysis and design of epidemics. So far we have avoided differentiating among the servers; better performance might be achieved by constructing a dynamic hierarchy, in which sites at high levels contact other high level servers at long distances and lower level servers at short distances. (The key problem with such a mechanism is maintaining the hierarchical structure.)

5. Acknowledgments

We would like to thank Mark Gealy, Sue Manning, and Don Woods who stimulated our early interest in this problem, implemented randomized anti-entropy in the Xerox Clearinghouse Service, and supervised the distribution of the new code to over 300 sites in the Xerox Corporate Internet. We would also like to thank Mike Paterson for his help with parts of the analysis. Finally, many thanks to Subhana Menis for threading the arcane production maze required to produce the camera ready copy of this paper.

References

[Ab] Karl Abrahamson, Andrew Addler, Lisa Higham, David Kirkpatrick
Probabilistic Solitude Verification on a Ring.
Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing.
Calgary, Alberta, Canada. 1986, Pages 161-173.

[Aw] Baruch Awerbuch and Shimon Even.
Efficient and Reliable Broadcast is Achievable in an Eventually Connected Network.
Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing.
Vancouver, B.C., Canada. 1984, Pages 278-281.

[Ba] Norman T. J. Bailey.
The Mathematical Theory of Infectious Diseases and its Applications (second edition).
Hafner Press, Second Edition, 1975.

[Be83] M. Ben-Or.
Another Advantage of Free Choice.
Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing.
Montreal, Quebec, Canada. 1983.

[Be85] M. Ben-Or
Fast Asynchronous Byzantine Agreement.
Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing.
Minaki, Ontario, Canada. 1985, Pages 149-151.

[Bi] A. D. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder.
Grapevine, An Exercise in Distributed Computing.
Communications of the ACM 25(4):260-274, 1982.

[Ch] K. M. Chandy and L. Lamport.
Distributed Snapshots: Determining Global States of Distributed Systems.
ACM Transactions on Computing Systems 3(1):63-75 1985

[Fr] J. C. Frauenthal.
Mathematical Modeling in Epidemiology, Pages 12-24.
Springer-Verlag, 1980.

[Gi] D. K. Gifford.
Weighted Voting for Replicated Data.
Proceedings of the Seventh Symposium on Operating Systems Principles ACM SIGOPS.
Pacific Grove, California. 1979, Pages 150-159.

[Jo] P. R. Johnson and R. H. Thomas.
The Maintenance of Duplicate Databases.
Bolt Beranek and Newman Inc., Arpanet Request for Comments (RFC) 677, 1975.

[La] Butler W. Lampson.
Designing a Global Name Service.
Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing.
Calgary, Alberta, Canada. 1986, Pages 1-10.

[Mo] P. Mockapetris.
The domain name system.
Proceedings IFIP 6.5 International Symposium on Computer Messaging,
Nottingham, England, May 1984.
Also available as:
USC Information Sciences Institute,
Report ISI/RS-84-133, June 1984.

[Op] Derek C. Oppen and Yogen K. Dalal.
The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment.
Xerox Technical Report: OPD-T8103, 1981.

[Pi] Boris Pittel.
On Spreading a Rumor.
SIAM Journal of Applied Mathematics 47(1):213–223, 1987.

[Ra] Michael O. Rabin.
Randomized Byzantine Generals.
24th Annual Symposium on Foundations of Computer Science.
IEEE Computer Society, 1983, Pages 403-409.

[Sa] S. K. Sarin and N. A. Lynch.
Discarding Obsolete Information in a Replicated Database System.
IEEE Transactions on Software Engineering SE-13(1):39-47 1987.