Group membership in the epidemic style

Richard A. Golding and Kim Taylor

UCSC–CRL–92–13

May 4, 1992

Concurrent Systems Laboratory
Computer and Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064

Existing group membership mechanisms provide consistent views of membership changes. However, they require heavyweight synchronous multicast protocols. We present a new lightweight group membership mechanism that allows temporary inconsistencies in membership views. This mechanism uses *epidemic communication* techniques to ensure that all group members eventually converge to a consistent view of the membership. Members can join or leave groups, and we show that the mechanism is resilient to $k \leq n - 2$ members failing by crashing, where n is the number of members in the group.

Keywords: Group membership, distributed systems, weak consistency replication

1 Introduction

Group membership mechanisms are an integral part of implementing many distributed services. Such mechanisms allow individual processes to dynamically join and leave a set of processes to facilitate some task. For example, consider a replication service in which each data item is stored at k sites, for a variable parameter k. If k increases or some of the sites fail, other sites will need to dynamically join the group responsible for maintaining the item in order to achieve the required resilience.

We present a new definition and implementation of the group membership problem based upon *epidemic communication* [Alon87, Demers88, Demers89]. Epidemic communication allows temporary inconsistencies in shared data in exchange for low-overhead implementation. More specifically, information changes are spread gradually throughout the process "population" without incurring the latency and bursty communication typically used to achieve a high degree of consistency. This is particularly important for wide-area systems, where failure is common, communication latency is high, and groups may contain hundreds or thousands of members.

Of course, applications must be willing to tolerate temporary inconsistencies. One example is a name service [Oppen81, Schroeder84, Terry85], in which location information can be treated as a *hint*. Hints provide good performance as long as they are usually correct and the cost of identifying and recovering from out-of-date hints is low. For similar reasons, distributed load balancing is an ideal application for epidemic techniques.

A necessary question is exactly what "correctness" means when groups are dynamic, communication is epidemic, and processes can fail. Before giving our implementation, we first formally specify the correctness and resilience conditions to be achieved by the group membership protocols. These have the flavor of epidemic communication guarantees [Alon87, Demers88, Demers89], namely eventual convergence of group views in a probabilistically bounded time. However, we also guarantee k-resilience of group communication and we guarantee that processes temporarily joining a group see past updates. These conditions are much stronger than would be guaranteed by simply propagating group information via known epidemic protocols.

In [Demers88, Demers89] three epidemic communication methods are specified: *direct mail*, *rumor mongery*, and *anti-entropy*. Each propagates the update at different rates, requiring different amounts of message traffic and providing different guarantees on eventual consistency. Direct mail propagates an update to other replicas using a single unreliable multicast datagram. A replica can use rumor mongery to send recent updates to other replicas, again using unreliable datagrams. Pairs of replicas perform a reliable exchange of database contents in an anti-entropy session. The techniques can be combined, using direct mail or rumor mongery for fast, unreliable propagation, while anti-entropy provides a reliable backup if the other methods fail.

Our group membership mechanism can use direct mail and rumor mongery, as described above. In addition, it utilizes a variant of anti-entropy termed *timestamped anti-entropy* [Golding91a]. In contrast to the earlier anti-entropy protocol, processes using it can determine when all other replicas have observed an update. This forms the basis of a group consensus mechanism which is used to loosely synchronize group membership information.

In contrast to our work, previous group membership mechanisms ensure greater consistency of group views at the expense of latency and communication overhead. Both the ISIS system [Birman87, Birman91] and a group membership mechanism by Cristian [Cristian89] are built on top of atomic broadcast protocols, and hence provide each process with the same sequence of group views. Ricciardi [Ricciardi91] is investigating an alternative group membership mechanism for Isis which does not use the underlying atomic broadcast. However, it still uses two- and three-phase commit protocols to maintain consistent group views. Finally, the Arjuna system [Little90] maintains a logically centralized group view via atomic transactions.

The remainder of our paper is organized as follows. In Section 2, we give our system model. In Section 3, we give some background on epidemic communication techniques and the properties of the underlying protocols used in our implementation. Section 4 contains a description of our group membership mechanism and its correctness and resilience conditions. Specific protocols for implementing this mechanism are in Section 5, and in Section 6 we discuss some experimental results on the fault tolerance and performance of those protocols. We conclude with a summary and future work in Section 7.

2 System model

Throughout this paper, the word *time* refers to real time, as opposed to any virtual time measure. When an event is said to happen *eventually* after time t, the probability that the event will not happen during the period $(t, t+\delta]$ goes to zero as $\delta \to \infty$.

The system includes a set P of processes, some of which are group members. A process can fail by crashing. Each process has a unique identity, and once a process has permanently failed it ceases to exist for all time. No assumptions are made about the rates at which processes progress, and it is impossible to distinguish between a slow process and one that has exhibited a temporary performance failure.

Every process p has a clock, denoted clock(p); the value of the clock at real time t is clock(p, t). This clock allows every important event in the process to be assigned a distinct, monotonically increasing timestamp. The clocks are loosely synchronized,[1] and differ by at most a constant ϵ :

$$(\forall t)(\forall p, q \in P)|\text{clock}(p,t) - \text{clock}(q,t)| < \epsilon.$$

We assume that processes have a mean time-to-failure (MTTF) much longer than the time required to perform the anti-entropy protocol. Such processes can be constructed from less-reliable processes if stable storage is available; this is the approach often taken on Unix systems.

The transmission time of messages is not bounded. The communication network does not always deliver messages in FIFO order, and it may lose or duplicate messages from time to time. It does not spontaneously create messages. No part of the network fails permanently, though temporary partitions can occur. Any process p can eventually send a message to any other process q if it continually tries to send until it receives an acknowledgment. Recent studies show that the Internet behaves in much this way [Golding91b, Long91].

3 Epidemic communication

Epidemic communication schemes provide communication between processes in a group, with weak guarantees on the consistency between them. In general, there is a non-zero probability that two processes agree on the state of a system, and eventually all processes are guaranteed to agree if no further updates occur. Most operations performed by a group of processes are first performed at only one process, and the results are later propagated to the others.

We have developed new epidemic communication protocols that guarantee, first, that every process will eventually observe an update, and second, that any process can determine whether every process has observed some update [Golding91a]. The first guarantee is provided by existing protocols, while the second is not.

[1] A version of the timestamped anti-entropy protocol works with unsynchronized clocks that need only to be strictly increasing at each process and to satisfy Lamport's happens-before condition. Our group membership protocols work with asynchronous clocks as well.

The first guarantee, which ensures that every process eventually observes every group operation, can be stated formally:

Condition 1 (Eventual consistency) *If an operation occurs at a process p at (real) time t, the operation will eventually be observed at any other process q, as long as the update is not superseded by another update, and q is a member of the group at some time $t' \geq t$.*

This condition extends that in previous work only to handle processes that join the group after an update occurs. Process q can join the group at any time before, during, or after t. This guarantee ensures that processes cannot join and leave the group so often that some updates are lost.

The group join and leave protocols must be crafted so as not to compromise this guarantee. The **leave** protocol we will present ensures that every process delays leaving long enough that ordinary epidemic communication propagates all its updates to every other process.

This guarantee is not met in one important case: when a process permanently fails and loses data. Our original formulation of the guarantee assumed that processes never failed permanently. However, in a real system many different problems can cause a process to lose data. No epidemic replication scheme can be free from this vulnerability, since operations are only performed at one process. A window of vulnerability must exist while the data is being sent to other processes. In practice the duration can often be reduced by disseminating the new updates rapidly, but real networks do not allow complete certainty. We will return to this topic when we discuss failure recovery.

The second guarantee provided by timestamped anti-entropy protocols is as follows:

Condition 2 (Detectable consistency) *Assume an update occurs at (real) time t, and let M be the set of processes that were members at time t. A group member process p can eventually determine that all processes in M have either observed the update or failed.*

This guarantee allows processes to perform consistent operations, if necessary, on distributed data even though the communication protocols do not enforce consistency. Existing epidemic protocols rely on probabilistic methods to detect consistency, and hence must either accept erroneous results, or delay consistent operations for long periods. The group membership protocols we will present use this guarantee to detect when all group members have observed a membership change.

Note that we have not included any guarantees on message delivery order. Epidemic communication protocols can deliver messages in many different orders, many of which do not respect causality. This allows the protocols to use efficient communication mechanisms.

3.1 Timestamped anti-entropy

We have developed an epidemic communication protocol, *timestamped anti-entropy*, that provides both these guarantees [Golding91a]. The protocol works, in general, by maintaining information on what updates and acknowledgments it has received from which processes. Pairs of processes occasionally exchange updates until they are mutually consistent. We have also developed a similar protocol that requires $O(n^2)$ state per process but allows unsynchronised clocks. This protocol was discovered independently by Agrawal and Malpani [Agrawal91].

Processes maintain a *summary timestamp vector* to record what updates they have observed [Mattern88]. Process A records a timestamp t for process B, if A has observed all updates generated at B up to time t on B's clock. Each process maintains one such timestamp in its timestamp vector for every process in the group. The vector provides a fast mechanism for transmitting summary information about the state of a process.

Each process also maintains an *acknowledgment timestamp vector* to record what messages have been acknowledged by other processes. If process A holds a timestamp t for process B, A knows that B has received every message from any sender with timestamp less than or equal to t. Process B periodically sets its entry in its acknowledgment vector to the minimum timestamp recorded in its summary vector; this will be viewed by A in later exchanges and become a part of its acknowledgment vector. This mechanism makes progress as long as process clocks are loosely synchronised and the acknowledgement vector is updated regularly. A process can determine that every other group member process has observed a particular update by looking at its local acknowledgment vector.

From time to time, a process will select another process and start an *anti-entropy session*. A session begins with the two processes exchanging their summary and acknowledgment vectors. Each process can determine if it has updates the other has perhaps not yet observed, if some of its summary timestamps are greater than the corresponding ones of its partner. These updates can be sent to the other process using a reliable stream protocol. The session ends with an exchange of acknowledgment messages. If any step of the exchange fails, either process can abort the session. If this mechanism is used as the only communication between processes, each update is sent exactly once to each process. We have proven that this mechanism guarantees that every process will eventually observe all updates [Golding91a].

4 Epidemic group membership

A *group* is defined as a set of members. It comes into existence when the first member initializes it, and ceases to exist when the last member leaves. Each group has a unique identity, so every newly-initialized group is distinct from all other groups, past or present. Processes become members by executing a **join** protocol, and stop either by failing or by executing a **leave** protocol.

The problem of *group membership* is determining the set of processes that make up a group. In our model, each member process maintains a *view* of the group membership. The view defines the set of processes a member believes are a part of the group at any given time. A member process will always have itself in its view.

The group view may also contain application-specific *shared state* information. A process updates the shared state by using epidemic communication to propagate changes to members currently in its group view. Our group views also contain internal information for use by the group membership protocols, such as status and timestamps. These will be discussed with our implementation in Section 5.

There are four group operations in the model of group membership discussed in this paper:

Initialization: create a new group.

Join: add a new process to a group. Transfer a copy of the shared state to the new process.

Leave: a group member voluntarily leaves. Coordinate state transfer from the leaving member.

Failure recovery: the group recovers from member failure and ejects failed member from group.

As noted earlier, existing group membership protocols provide *consistent* views of group membership; that is, every group member observes changes to the membership in the same order. We guarantee the following: if, at any time, all group membership changes cease, processes will then converge to a single consistent view. More specifically, given the last such change in membership at time t, the probability that two processes, p and q, disagree on group membership at time $t + \delta$ goes to zero as δ increases. The same holds with respect to each process' view of the membership status of individual processes.

The view information for one group can be represented as a *knows-about* directed graph $K = (P, E)$. For processes p and q, there will be an edge (p, q) in the graph if q is in p's view, and an edge (q, p) if p is in

q's view. There will be an edge (p, p) for every process that believes it is part of the group. The set of such vertices (processes) is the *member* set M.

The structure of the *knows-about* graph is *correct* at any time iff all processes can eventually converge to the same view. Since view information is propagated along edges in the graph, every process p that believes it is part of the group should be in the transitive closure of the edges directed away from every other process that believes itself a member.

Definition 1 (Correctness) *Let M be the set of processes $p \in P$ for which there are edges $(p, p) \in E$. Let E^* be the transitive closure of E. K is* correct *iff $(\forall p \in M)(\forall q \in M)((p, q) \in E^*)$.*

Every group operation must preserve the correctness of the *knows-about* relation. As part of this, whenever a process joins or leaves the group, it must find other processes to act as its *sponsor*. The sponsors are the source of the new member's state, and membership information propagates from the sponsors to other existing members.

The membership system should be able to withstand some number k of simultaneous permanent process failures without compromising the correctness of members' views. If processes never fail permanently, then the mechanisms in this paper will work correctly if every process obtains one sponsor when they join. However, if they can permanently fail then it is possible to obtain an incorrect *knows-about* graph. A *knows-about* graph that can withstand up to k simultaneous process failures and still be correct is called a *k-resilient* membership graph.

Definition 2 (k-resilience) *A* knows-about *graph K is k-*resilient *if it is correct after any k or fewer vertices (processes) are deleted from K.*

Note that this definition of resilience is only concerned with group membership information. As we noted in the last section, process failures can cause an update to shared state to be lost if that update has not propagated to other (non-faulty) processes. Therefore k-resilience implies that shared state updates will propagate from one non-faulty process to another in the face of k failures.

The membership protocols are correct if they transform one correct *knows-about* graph into another. They are k-resilient if they transform one correct, k-resilient graph into another. The proof of the following lemma is straightforward:

Lemma 1 *For any two members $m, m' \in M$, a correct* knows-about *digraph K can be viewed as a flow graph with source m and sink m'. K is k-resilient if, for all members $m, m' \in M$, the minimum vertex cut of the associated flow graph with source m, sink m' is at least $k + 1$.*

> *Proof:* If the minimum vertex cut of the flow graph from m to m' is at least $k + 1$, then up to k vertices can be removed from K while maintaining a path from m to m'. Call the set of failed processes F; $|F| \leq k$. There is an edge (m, m') in the transitive closure of $K - F$. Since this condition holds for all pairs of members, the *knows-about* graph is still correct after removing the failed processes F. □

The membership protocols defined in the next section ensure that this condition holds by constructing a $(k + 2)$-clique in the *knows-about* graph around every member as it joins, and ensures that every process stays part of a $(k + 2)$-clique as processes leave or fail.

5 Protocols for epidemic group management

In this section we will present the group membership protocols in detail. These protocols allow for creating a new group, joining and leaving a group, and handling member failure. We will discuss how these

protocols provide k-resilience, and why they do not interfere with the guarantees provided by epidemic communication.

5.1 Data structures

Each member process maintains a view of the group membership, defining the set of processes it believes are a part of the group. The view contains a list of each member, its status, and two timestamps:

view: list of (process, status, timestamp, aetimestamp)

Updates to the list are propagated between processes using normal epidemic communication methods. Status and timestamp are used for group membership information; aetimestamp is used by the timestamped anti-entropy protocol.

Every process in the list has a status. A process that is part of the group will have status member, while one that has voluntarily left the group will have status left. A failed process is marked as failed. The left and failed records are maintained for a period of time so the information can propagate to other processes – that is, until all processes have reached consensus on the status of the failed process. These records are often called *death certificates*, and are used in other systems that allow temporary inconsistency [Bloch87, Demers89]. The communication protocol allows a process to determine when consensus has been reached, so these records can safely be purged.

The timestamp for process p records the clock at p, clock(p, t) when the process entered its current state. For failed status the timestamp can either be some value of 'infinity', or an approximation obtained from another process' clock. The approximation must be distinct from any clock value from the failed process. This will be taken up in more detail in Section 5.5.

5.2 Initializing a new group

A process can create a new group by performing an **initialization** operation. This does two things. First, it creates a new group identity. Second, it sets up a membership view at the process. The view contains only the initializing process, with status member at time zero. Since there is only one process in the group, there are no concerns about consistency or failure resilience.

Until k additional members have joined the group, there is no possibility of k-resilience. The **join** protocol presented in Section 5.3 will form a complete *knows-about* graph when the membership is small (i.e. $n <= k + 2$).

5.3 Group join

A process joins a group by finding one or more group members, then contacting them until it has obtained enough *sponsors* among the current membership to ensure k-resilience. We assume the existence of a fault-tolerant method by which processes can find other processes, such as broadcasting, well-known addresses, or a location service. The method must always provide at least one location that is a member. Processes leaving the group will return forwarding addresses to other members.

To keep the *knows-about* graph k-resilient, a process joining the group must obtain $k + 1$ sponsors before becoming a full-fledged group member. If this is impossible because the group is too small, the graph is kept as resilient as possible by using all members as sponsors. These sponsors put the process in their view, meeting the fault-tolerance criterion. The basic protocol is as follows:

1. Obtain a list of possible group members using the location method. As noted above, this list must include at least one group member.

2. The new process constructs a view containing only itself, with status pending member.

3. Send a message to one possible member from the list, requesting sponsorship of that process. The message includes a copy of the joining process' view. If the possible member no longer exists, the message fails. If it has a forwarding address for a member, it sends back the address, which is added to the list of possible members. If the process accepts sponsorship, it replies with a copy of its view, which is merged into the joining process' view and the processes in it are added to the list of possible members.

4. The previous step is repeated until $k + 1$ processes have accepted sponsorship, or until all group members are sponsors.

5. The process changes its status to member.

6. (Optional.) The process performs anti-entropy with each of its sponsors.

The joining process is considered to be a group member after changing its status to member.

Theorem 1 *The above protocol preserves k-resilience in the* knows-about *graph.*

Proof: The process j is joining a group with members M. Assume that the set $S \subseteq M$ is the set of sponsors it obtains; $|S| = k + 1$. Denote the *knows-about* graph before this protocol is executed as K_M, which is assumed to be k-resilient.

After executing this protocol, up to k failures occur in the set $F = \{f_1, \ldots, f_k\}$. Assume that $j \notin F$. The failed processes can be divided into two sets: the members of $F_S = F \cap S$ are sponsors for j, and those of $F_M = F \cap (M - S)$ are not.

For every member m that has not failed ($m \in M - F$), there is a path $m, \ldots, m_s$ in the *knows-about* graph, where $m_s \in S$, since $|S| > |F|$ and the original *knows-about* graph is k-resilient. Since there is an edge (m_s, j) in the new *knows-about* graph, for every functioning member m there is a path $m, \ldots, m_s, j$.

The same argument holds in reverse for paths from j to all other members in $M - F$, so the new *knows-about* graph is correct after up to k failures. Therefore it is k-resilient. □

In addition, the protocol forms a n-clique when the number of members n is not more than $k + 2$. Once a $k + 2$-clique is reached, k-resilience is established.

5.4 Group leave

When a member leaves the group, it must not cause the *knows-about* relation to become less than k-resilient, or even incorrect. This can happen when removing the member from the group causes the minimum vertex-cut to drop below $k + 1$ vertices. Further, it must ensure that all updates are propagated to other processes.

To alleviate this problem, a process that wants to leave the group must do so in two steps.

1. The process declares its intent to leave by changing its status to left and performing anti-entropy with one or more other members.

2. The process then waits until all processes that were group members at the time it declared its intent have observed the status change. Then the member can destroy its state.

During this delay, the process cannot perform new updates, and so should not accept new operations. However, it must maintain all its state and actively participate in anti-entropy sessions with all comers.

The delay ensures that the *knows-about* graph cannot become vulnerable to k or fewer failures, and that no updates performed at the process will be lost. This also ensures that the leaving process does not compromise the correctness condition for epidemic communication. The delay completes when all members have observed that the process expressed its intent to leave. This requires a chain of anti-entropy events starting after time t between it and all other members.

When the last process leaves a group, the group ceases to exist. Name or location services should be informed.

Theorem 2 *The leave protocol preserves k-resilience in the* knows-about *graph as long as at least $k+1$ processes remain in the group.*

Proof: Consider any path $m_1, \ldots, m_n$ in the *knows-about* graph. Assume some subset $\{l_1, \ldots, l_m\} \subseteq \{m_2, \ldots, m_{n-1}\}$ of these have declared their intent to leave the group. Without loss of generality, assume that l_1 is the first to find that its change has been observed by all group members, and that this occurs at time t_1. At time $t_2 \geq t_1$, the second process l_2 will find that it has met its condition and will cease to exist. Assume that the *knows-about* graph is k-resilient for $t \leq t_1$. Then for the period $t_1 < t \leq t_2$ the part of the path m_{k-1}, l_1, m_{k+1} can be replaced by the sequence m_{k-1}, m_{k+1} since m_{k-1} must know about all members that l_1 knew about. These steps can be repeated for each member as it leaves the group.

Since the graph was originally k-resilient, after some k failures there must still be some path $m_1, \ldots, m_n$ between any two processes. This path will still exist after $l_1 \ldots l_m$ leave the group. □

As with the **join** protocol, this protocol forms a n-clique when there are fewer than $k+1$ group members.

5.5 Failure recovery

Finally, we discuss a method for failure recovery that is no more expensive than the other operations. Processes can exhibit either temporary performance failures or permanent failures. Host rebooting, transient load, and network router failure are typical temporary failures. A process that has permanently failed will never recover, or recovery will take so long that it might as well be forever. Extended repair, disaster, or unexpected removal from service are permanent failures. We assume a standard probabilistic failure detection mechanism such as timeouts.

Epidemic communication protocols already handle temporary failures. Soon after the process recovers, it will begin receiving rumors and anti-entropy sessions. Between its state on stable storage and the information preserved at other processes, these operations will catch it up to the current state of the system.

Permanent failures are a different problem. Most seriously, they compromise the guarantees provided by the replication protocols. Information that has not propagated out of the failed process may be trapped there and lost. Without remedy, this would be especially serious for group information, as it could create an incorrect *knows-about* relation. For example, a single process that has just joined the group with one sponsor can become isolated from the rest of the group if its sponsor fails.

The only way to solve this problem is to ensure that information is recorded at several processes before a group membership operation is complete. The group membership protocols as given maintain a k-resilient *knows-about* graph, where information is always recorded by at least $k+1$ processes.

When a process must be ejected from the group, the *knows-about* graph can become less than k-resilient, just as with group leaves. If we were only concerned with resilience to k failed processes over all time, the

loss of one failed process would not cause a loss of resilience; only $k-1$ failures would be possible and $k-1$ resilience still holds. However, if a group is large enough we can do better by re-obtaining k-resilience even though as many as k processes have already failed. As long as the number of members has not dropped below $k+1$, anti-entropy among the members will eventually restore k-resilience. In fact, anti-entropy will eventually generate a complete graph among the members if the membership is stable long enough.

Ejection is handled as follows: when process p finds that process f has failed, it marks f as status failed in its view, and sets the timestamp of the status change to infinity. This will prevent a process from recovering and claiming to be a group member, then polluting the data base with out-of-date information. Anti-entropy and rumor-mongery will spread this information to all other group members. Process f is ejected, and the failure recovered from, when all group members have observed its failure. As with group leaves, k-resilience is restored once all members have observed the ejection.

This approach leaves the group vulnerable for the time it takes to restore k-resilience. The length of time the group is vulnerable can be decreased by increasing the rate at which members perform anti-entropy.

6 Experimental results

We are evaluating the performance and fault tolerance of these protocols, and we are implementing them as part of two different systems.

The performance of the group membership protocols depends mostly on the performance of the epidemic communication protocols. We have modelled a system of n processes as a Markov system with $O(n^2)$ states, with each state labelled with the number of processes available and the number that have observed the update. Data loss due to permanent site failure appears to be negligible in systems like the Internet, where sites stay in service for several years. The usual approximations to stable storage, such as delayed writeback from volatile storage, is also a negligible problem. We have found that the time required for a system to reach agreement increases approximately as the log of the number of processes, and that under reasonable update and read rates information is likely to propagate to most sites before it is needed. We are extending this performance evaluation to include a more complex model of transient network and process failures, and to evaluate the message traffic induced by these protocols.

These protocols are being implemented in two systems. The *refdbms* system implements a distributed bibliographic database. It is being implemented by one of us as vehicle for gathering performance measures of a large-scale wide-area distributed application [Golding92]. The *tattler* system is a distributed performance monitor for the Internet that collects statistics on host availability. It is being built by Long [Long92] at UC Santa Cruz, as part of his research into distributed system reliability. These systems will provide insight on how difficult these group membership protocols are to implement.

7 Summary

Group membership operations can use semantics similar to those provided by epidemic communication protocols: that all processes will eventually see changes, but that only one or a few see the change initially. We have shown that basic epidemic communication protocols can maintain their correctness when used in a group with dynamically changing membership.

Our group membership protocols provide four operations: initialization, join, leave, and failure recovery. Each of these protocols proceeds immediately at only a few processes, providing better scalability than existing consistent group membership protocols. The operations maintain a *view* of group membership at each process that records the members that process knows about. A *knows-about* graph is defined by these views, and the protocols take care to maintain its correctness even in the presence of a few failures. In

particular, the graph is said to be k-resilient if up to k members can fail and not compromise the graph. The join and leave protocols have been shown to preserve k-resilience. These algorithms handle permanent failure – as distinguished from transient failure – though it reduces the resilience of the *knows-about* graph until it is restored by normal epidemic propagation.

An initial performance investigation indicates that these protocols propagate information rapidly among processes, and are reliable even in the face of high failure rates.

Acknowledgments

Richard Golding was supported in part by a fellowship from the Santa Cruz Operation, and by the Concurrent Systems Project at Hewlett-Packard Laboratories. Kim Taylor was supported in part by NSF Grant CCR–9111132. Darrell Long, Robert Ellefson, John Wilkes, and George Neville-Neil proofread this paper and made many helpful suggestions.

References

[Agrawal91] Divyakant Agrawal and Amr El Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Transactions on Computer Systems*, **9**(1):1–20 (February 1991).

[Alon87] Noga Alon, Amnon Barak, and Udi Manber. On disseminating information reliably without broadcasting. *Proceedings of 7th International Conference on Distributed Computing Systems* (Berlin, 21–25 September, 1987), pages 74–81, R. Popescu-Zeletin, G. Le Lann, and K. H. Kim, editors (1987). IEEE Computer Society Press.

[Birman87] Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, **5**(1):47–76 (February 1987).

[Birman91] Kenneth P. Birman, Robert Cooper, and Barry Gleeson. Programming with process groups: group and multicast semantics. Technical report TR–91–1185 (29 January 1991). Department of Computer Science, Cornell University.

[Bloch87] Joshua J. Bloch, Dean S. Daniels, and Alfred Z. Spector. A weighted voting algorithm for replicated directories. *Journal of the ACM*, **34**(4):859–909 (October 1987).

[Cristian89] Flaviu Cristian. A probabilistic approach to distributed clock synchronization. *Proceedings of 9th International Conference on Distributed Computing Systems* (Newport Beach, CA), pages 288–96 (1989). IEEE Computer Society Press.

[Demers88] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. *Operating Systems Review*, **22**(1):8–32 (January 1988).

[Demers89] Alan Demers, Mark Gealy, Dan Greene, Carl Hauser, Wes Irish, John Larson, Sue Manning, Scott Shenker, Howard Sturgis, Dan Swinehart, Doug Terry, and Don Woods. Epidemic algorithms for replicated database maintenance. Technical report CSL–89–1 (January 1989). Xerox Palo Alto Research Center, CA.

[Golding91a] Richard A. Golding. Distributed epidemic algorithms for replicated tuple spaces. Technical report HPL–CSP–91–15 (28 June 1991). Concurrent Systems Project, Hewlett-Packard Laboratories.

[Golding91b] Richard A. Golding. Accessing replicated data in a large-scale distributed systems. Master's thesis; published as Technical report UCSC–CRL–91–18 (June 1991). Computer and Information Sciences Board, University of California at Santa Cruz.

[Golding92] Richard A. Golding. *Weak-consistency group communication and membership*. PhD thesis (September 1992, expected). University of California at Santa Cruz.

[Little90] Mark C. Little and Santosh K. Shrivastava. Replicated k-resilient objects in Arjuna. *Proceedings of Workshop on Management of Replicated Data* (Houston, Texas), pages 53–8 (November 1990).

[Long91] Darrell D. E. Long, John L. Carroll, and C. J. Park. A study of the reliability of Internet sites. *Proceedings of 10th IEEE Symposium on Reliability in Distributed Software and Database Systems* (Pisa, Italy), pages 177–86 (September 1991). Institute of Electrical and Electronics Engineers.

[Long92] Darrell D. E. Long. A replicated monitoring tool. Technical report UCSC–CRL–92–14 (April 1992). Computer and Information Sciences Board, University of California at Santa Cruz.

[Mattern88] Friedemann Mattern. Virtual time and global states of distributed systems. *Proceedings of International Workshop on Parallel Algorithms* (Chateau de Bonas, France, October 1988), pages 215–26, M. Cosnard, Y. Robert, P. Quinton, and M. Raynal, editors (1989). Elsivier Science Publishers, North-Holland.

[Oppen81] D. C. Oppen and Y. K. Dahl. The Clearinghouse: a decentralized agent for locating named objects in a distributed environment. Technical report OPD–T8103 (1981). Xerox Office Products Division, Palo Alto, Ca.

[Ricciardi91] Aleta M. Ricciardi and Kenneth P. Birman. Using process groups to implement failure detection in asynchronous environments. Technical report TR91–1188 (7 February 1991). Department of Computer Science, Cornell University.

[Schroeder84] Michael D. Schroeder, Andrew D. Birrell, and Roger M. Needham. Experience with Grapevine: the growth of a distributed system. *ACM Transactions on Computer Systems*, **2**(1):3–23 (February 1984).

[Terry85] Douglas Brian Terry. *Distributed name servers: naming and caching in large distributed computing environments*. PhD thesis, published as Technical report CSL–85–1 (February 1985). Xerox Palo Alto Research Center, CA.