Probabilistic Broadcast *

Mark Hayden and Ken Birman

Computer Science Department, Cornell University

1 Introduction

We present a class of scalable and probabilisticly reliable communication protocols. The protocols are based on a probabilistic system model and thus their properties tend to be probabilistic in nature. The protocols are scalable in two senses. First, the message costs and latencies of the protocols grow slowly with the system size. Second, the reliability of the protocols, expressed in terms of the probability of a failed run of a protocol, approaches 0 exponentially fast as the number of processes is increased. This scalable reliability is achieved through a form of gossip protocol which is strongly self-stabilizing in a sense similar, although not identical to, the notion of self stabilizing systems proposed by Dijkstra[Dij74].

The protocols we present aim to resolve problems in using typical gossip protocols. Gossip protocols are broadcast protocols that propagate a message to all processes in the system through occasional and insistent retransmission, or *gossiping*, of messages[DGH+87, GT92]. A message is initially sent by some process to some subset of the other processes, all of which then retransmit the message to more processes, and so on until eventually all have received the message. Two useful characteristics of gossip protocols are their scalability and convergence properties. Gossip style communication is inherently scalable because gossiping only requires from each process a fixed amount of effort that generally does not increase as the system grows . Gossip messages propagate exponentially quickly, so messages can usually be expected to flood the network in a logarithmic number of gossip rounds. Although the actual rate is not guaranteed because of failures, gossip protocols usually guarantee that, given enough time, eventually either all or no correct processes will deliver a message. This property is called *eventual convergence*. Although eventual convergence is sufficient for many uses, we argue that it is not sufficiently strong for many others because it does not provide bounds on message latency or ordering properties.

The solution we present to problems of using eventual convergence with gossip protocols is to weaken the convergence property and introduce synchronization and ordering properties. This is achieved by taking a particular gossip protocol and submitting it to analysis in the context of a probabilistic system model. We arrive at a new broadcast primitive, called *pbcast*, with properties that can be demonstrated to be sufficient for building higher level constructs for synchronization and replicated data.

Our work can also be compared to timed asynchronous protocols [CASD85], which use gossip-like flood-fill mechanisms to broadcast messages. These protocols differ from ours primarily in the system model and resulting analysis. The timed asynchronous protocols are analyzed under

*This research was supported under ARPA/ONR order 9247, grant N00014-92-J-1866, and a grant by Siemens Corporate Research. Email addresses for authors: hayden@cs.cornell.edu, ken@cs.cornell.edu.

a standard bounded failure model and arrive at non-probabilistic properties; our protocols are analyzed under a probabilistic failure model and provide probabilistic properties. Similar to work with some timed asynchronous protocols, our system model makes assumptions that are not satisfied by typical communication networks, limiting the applicability of the protocols to special purpose.

2 System Model

The system model in which we analyze pbcast is a static set of processes communicating synchronously over a fully connected, point-to-point network. The processes have unique, totally ordered identifiers, and can toss weighted, independent random coins. Runs of the system proceed in a sequence of rounds in which messages sent in the current round are delivered in the next. There are two types of failures, both probabilistic in nature. The first are process failures. There is an independent, per-process probability of at most τ that a process has a crash failure during the finite duration of a protocol. Such processes are called faulty. The second type of failures are message omission failures. There is an independent, per-message probability of at most ϵ that a message between non-faulty processes experiences a send omission failure. The union of all message omission failure events and process failure events are mutually independent. There are no malicious faults, spurious messages, or corruption of messages. We expect that both ϵ and τ are small probabilities. (For example, unless otherwise stated, the values used in the graphs in this paper are $\epsilon = 0.05$ and $\tau = 0.001$.)

The impact of the failure model above can be described in terms of an adversary attempting to cause a protocol to fail by manipulating the system within the bounds of the model. Such an adversary has these capabilities and restrictions:

- An adversary cannot use knowledge of future probabilistic outcomes, interfere with random coin tosses made by processes, cause correlated (non-independent) failures to occur, or do anything not enumerated below.
- Has complete knowledge of the history of the current run of the protocol.
- At the beginning of a run of the protocol, ability to individually set process failure rates, within the bounds $[0..\tau]$.
- For faulty processes, ability to choose an arbitrary point of failure.
- For messages, ability to individually set send omission failure probabilities within the bounds of $[0..\epsilon]$.

Note that although probabilities may be manipulated by the adversary, the adversary may only manipulate the probabilities to make the system "more reliable" than the bounds, ϵ and τ.

On this system model, we will layer protocols with strong probabilistic convergence properties. The probabilistic analysis of these properties is necessarily only valid in runs of the protocol in which the system obeys the model. The independence properties of the system model are quite strong and are not likely to be continuously realizable in an actual system. For example, partition failures in the sense of correlated communication failures do not occur in this model. Partitions can be "simulated" by the independent failures of several processes, but are of vanishingly low probability. However, the protocols we develop using pbcast, such as our replicated data protocol, remain safe even when the system degrades from the model: for instance, messages will still be delivered at most once. In addition, protocols can be made self-healing. For instance, our replicated data protocol has guaranteed eventual convergence properties similar to normal gossip protocols: so if the system observes the model for long enough, the protocol will eventually recover from the "failure" and reconverge to a consistent state.

3 Pbcast Protocol

Pbcast is a broadcast protocol with properties that are useful for building voting-style protocols reminiscent of database quorum protocols. In addition to basic properties such as at-most-once delivery and integrity (only previously pbcast messages are delivered), pbcast guarantees totally ordered delivery and "probabilistic reliability" (described below). Although processes may not deliver the same sets of messages, all messages are delivered in a consistent total ordering, according to the time-stamp placed on them by their senders.

We begin with an unordered version of pbcast with static membership (see figure 1 for the protocol). Pbcast is adaptable to dynamic membership, but the protocol is much simpler with static membership. The protocol consists of a fixed number of rounds, in which each process participates in at most one round. A process initiates a pbcast by sending a message to a random set of other processes. When other processes receive a message for the first time, they gossip the message to some other randomly chosen members. Each process only gossips once: the first process does nothing after sending the initial messages and the other processes do nothing after sending their set of gossip messages. Processes choose the destinations for their gossip by tossing a weighted random coin for each other process to determine whether to send a gossip message to that process. Thus, the parameters of the protocol are:

- P: the set of processes in the system. $N = |P|$.
- R: the number of rounds of gossip to run.
- β: the probability that a process gossips to each other process (the weighting of the coin mentioned above).

The behavior of the gossip protocol mirrors a class of disease epidemics which nearly always infect either almost all of a population or almost none of it. The pbcast bimodal delivery distribution stems from the "epidemic" behavior of the gossip protocol. The normal behavior of the protocol is for the gossip to flood the network in a random but exponential fashion.

3.1 Adding total ordering.

In the protocol above, the pbcast messages are unordered. To achieve a total ordering of pbcasts, processes delay delivering a message until any earlier messages must have been already received. Since an earlier pbcast cannot be delivered any later than R rounds after it was sent, it is sufficient to delay delivery of pbcast messages until then (see appendix for the protocol). The addition of total ordering to the pbcast gossip protocol uses pbcast's fixed number of rounds. In conventional gossip protocols, this method of achieving a total ordering is not possible because the number of rounds before a message propagates in the network is not bounded. This method of achieving a total ordering is equivalent to that in [CASD85].

```
(* State kept per pbcast:  have I
 * received a message regarding
 * this pbcast yet? *)
let received\_already = false

(* Initiate a pbcast. *)
to pbcast(msg):
   deliver\_and\_gossip(msg,\nrounds)


(* Handle message receipt. *)
on receive Gossip(msg,round):
   deliver\_and\_gossip(msg,round)
```

```
(* Auxiliary function. *)
to deliver\_and\_gossip(msg,round):
   (* Do nothing if already received
    * it. *)
   if received\_already then return

   (* Mark the message as being seen
    * and deliver. *)
   received\_already := true
   deliver(msg)

   (* If last round, don't gossip. *)
   if round = 0 then return

   foreach p in \processes:
      do with probability \rate:
         sendto p Gossip(msg,round-1)
```

Figure 1: **Unordered Pbcast Protocol**. *Note: message receipt and pbcast are executed as atomic actions.*

4 Probabilistic Reliability

Pbcast provides reliability through its bimodal delivery distribution: when configured correctly, a pbcast is almost always delivered to "most" or to "few" processes, and almost never to "some" processes. A pbcast delivery distribution over the number of processes is depicted in figure 2a (on page 11). The figure plots the probability of different numbers of processes delivering a pbcast. For instance the probability that 26 out of the 50 processes deliver a pbcast is around 10^{-28}. This bimodal distribution property is presented here informally, but later we demonstrate how to calculate the actual probability distributions for a particular configuration of pbcast.

A bimodal distribution is useful for voting-style protocols where, as an example, updates must be made at a majority of the processes to be "valid." However, problems occur in such protocols when failures cause a large number of processes to carry out an update, but not a majority. Pbcast overcomes this difficulty through its bimodal delivery distribution by making votes almost always be weighted strongly for or against an update, and very rarely be evenly divided. By counting votes, it can almost always be determined whether an update was valid or not, even in the presence of some failed processes. (See the appendix for a presentation of a replicated data protocol built with pbcast. This is placed in the appendix due to limitations of space and in keeping with our emphasis in this paper on the core pbcast protocol. However, we also have a synchronization protocol that can be shown to be safe but only probabilisticly live, and a detailed treatment of both protocols is planned for a future publication.)

With pbcast, the "bad" cases are when "some" processes deliver the pbcast and these are the cases that pbcast makes unlikely to occur. We will call pbcasts that are delivered to "some" processes *failed* pbcasts, and pbcasts that are delivered to "few" processes *invalid* pbcasts. Thus, a pbcast which is delivered to only a few destinations is considered invalid but not failed.

5 Pbcast Analysis

The key characteristic of pbcast is its probabilistic bimodal delivery distribution, and this property is the one we are concerned with in this analysis. The analysis here is of the simpler, unordered protocol which has essentially the same behavior as the totally ordered protocol, other than the ordering property. Our analysis of pbcast will show how to calculate bounds on the probability of a pbcast failure. It would be preferable to present instead a closed form solution for this, but deriving closed form bounds on the behavior of non-trivial epidemics of the kind we are interested in is an open problem in epidemic theory. In the absence of closed form bounds, the approach of this analysis will be to derive a recurrence relation between successive rounds of the protocol, which will then be used to calculate an upper bound on the chance of a failed pbcast run. Plots of calculations based on this analysis are presented in the following section.

5.1 Notation and Probability Background

The following analysis uses some standard probability theory. We will introduce 3 types of random variables. Lower case variables, such as f, r, and s, are integral random variables; upper case variables, such as X, are binary random variables (they take values from $\{0,1\}$); and upper case bold variables, such as $\mathbf{X}$, are integral random variables corresponding to sums of binary variables of the same letter: $\mathbf{X} = \sum X_i$.

We use the standard notation, $P\{v = k\}$ to refer to the probability of the random variable v having the value k. For binary variables, $P\{X\} \equiv P\{X = 1\}$. With lower case integral random variables, in $P\{r\}$ the variable serves both to specify a random variable and as a binding occurrence for a variable of the same name ($g(x)$ is a single variable function): $P\{r\} = g(r) \equiv P\{r = r\} = g(r) \equiv P\{r = y\} = g(y)$ (after a change of variables).

The distributions of sums of independent, identically distributed binary variables are called binomial distributions. If $\forall_{0 \leq i < n} . P\{X_i\} = p$, then: $P\{\mathbf{X} = k\} = \binom{n}{k}(p)^k(1-p)^{n-k}$.

We use relations among random variables to derive bounds on the distributions of the weighted and unweighted sums of the variables. Let X_i, Y_i, and Z_i form finite sets of random variables, and $g(i)$ be a non-negative real valued function defined over the integers. If $\forall_i . P\{X_i\} \leq P\{Y_i\} \leq P\{Z_i\}$, then:

$$P\{\mathbf{Y} = k\} \leq P\{\mathbf{Z} \geq k\} - P\{\mathbf{X} \geq k+1\} \tag{1}$$

$$\sum_{0 \leq i < n} P\{\mathbf{X} = i\} g(i) \leq \sum_{0 \leq i < n} P\{\mathbf{Y} = i\} \max_{0 \leq j \leq i} g(j) \tag{2}$$

These theorems will be applied later in the analysis.

5.2 A recurrence relation

The first step is to derive a recurrence relation that bounds the probability of protocol state transitions between successive rounds. We describe the state of a round using three integral random variables: s_t is the number of processes that may gossip in round t (or in epidemic terminology the infectious processes), r_t is the number of processes in round t that have not received a gossip message yet (the susceptible processes), and f_t is the number of infectious processes in the current round which are faulty. Some relations between these variables:

- $s_0 = 1$, $r_0 = N - 1$, $f_0 = 0$
- $r_{t+1} + s_{t+1} = r_t$

- $\sum_{0 \le t \le R} s_t + r_R = N$

The recurrence relation we derive, $\mathcal{R}(s_t, r_t, f_t, s_{t+1})$, is a bound on the conditional probability, given the current state described by (s_t, r_t, f_t), that s_{t+1} of the r_t susceptible processes receive a gossip message from this round. Expressed as a conditional probability, this is: $P\{s_{t+1} | s_t, r_t, f_t\}$.

For each of the r_t processes, we introduce a binary random variable, X_i, corresponding to whether a particular susceptible process receives gossip this round. s_{t+1} is equal to the sum of these variables, $\sum X_i$ or equivalently $\mathbf{X}$. In order to get $\mathcal{R}(s_t, r_t, f_t, s_{t+1})$, we will derive bounds on the distribution of $\mathbf{X}$. Our derivation will be in four steps. First we consider $P\{X_i\}$ in the absence of faulty processes and with fixed message failures. Then we introduce, separately, generalized message failures and faulty processes, and finally we combine both failures. Then we derive bounds on $P\{\mathbf{X} = k\}$ for the most general case.

Fixed message failures. The analysis begins by assuming that there are no faulty processes, and that message delay failures occur with exactly ϵ probability, no more and *no less*. This second assumption limits the system from behaving with a more reliable message failure rate. In the absence of these sorts of failures, the behavior of the system is the same as a well known (in epidemic theory) epidemic model, called the chain-binomial epidemic. The literature on epidemics provides a simple method for calculating the behavior of these epidemics when there are an unlimited number of rounds and no notion of failures[Bai75]. We introduce constants $p = \beta(1-\epsilon)$ and $q = 1 - p$ (see above on page 3 for definitions of β and ϵ): p is the probability that both an infectious process gossips to a particular susceptible process and also the message does not experience a send omission failure (under the assumption of fixed message failures).

For each of the r_t susceptible processes and its corresponding variable, X_i, we consider the probability that at least one of the s_t infectious processes sends a gossip message which gets through. Expressed differently, this is the probability that not all infectious processes fail to send a message to a particular susceptible process:

$$P\{X_i\} = 1 - (1-p)^{s_t} = 1 - q^{s_t}$$

Generalized message failures. A potential risk in the analysis of pbcast is to assume, as may be done for many other protocols, that the worst case occurs when message loss is maximized. Pbcast's failure mode occurs when there is a partial delivery of a pbcast. A pessimistic analysis must consider the case where local increases in the message delivery probability decrease the reliability of the overall pbcast protocol. We extend the previous analysis to get bounds on $P\{X_i\}$, but where the message failure rate may be anywhere in the range of $[0..\epsilon]$.

Consider every process i that gossips, and every process j that i sends a gossip message to. With generalized message failures, there is a probability ϵ_{ij} that the message experiences a send omission failure, such that $0 \le \epsilon_{ij} \le \epsilon$. This gives bounds $[p_{lo}..p_{hi}]$ on p_{ij}, the probability that process i both gossips to process j and the message is delivered: $\beta(1-\epsilon) = p_{lo} \le \beta(1-\epsilon_{ij}) = p_{ij} \le p_{hi} = \beta$. (We also have $q_{lo} = 1 - p_{lo}$ and $q_{hi} = 1 - p_{hi}$.)

This in turn gives bounds on the probability of each of the r_t processes being gossiped to, expressed using the variables X_{hi} and X_{lo} which correspond to a fixed message failure rate model:

$$1 - q_{lo}^{s_t} = P\{X_{lo}\} \le P\{X_j\} \le P\{X_{hi}\} = 1 - q_{hi}^{s_t}$$

Process failures. Introducing process failures into the analysis is done in a similar fashion to that of generalized message failures. For simplicity in the following discussion, we again fix the probability of message failure to ϵ.

We assume that f_t of the s_t infectious processes that are gossiping in the current round are faulty. For the purposes of analyzing pbcast, there are 3 ways in which processes can fail. They can crash before, during, or after the gossip stage of the pbcast protocol. Regardless of which case applies, a process always sends a subset of the messages it would have sent had it not been faulty: a faulty process never introduces spurious messages. If all f_t processes crash before sending their gossip messages then the probability of one of the susceptible processes receiving gossip message, $P\{X_i\}$, will be as though there were exactly $s_t - f_t$ correct processes gossiping in the current round. If all f_t processes crash after gossiping then the probability will be as though all s_t processes gossiped and none of the f_t processes had failed. All other cases cause the random variables, X_i, to behave with some probability in between:

$$1 - q^{s_t - f_t} = P\{X_{lo}\} \leq P\{X_i\} \leq P\{X_{hi}\} = 1 - q^{s_t}$$

Combined failures. The bounds from the two previous sections are "combined" to arrive at:

$$1 - q_{lo}^{s_t - f_t} = P\{X_{lo}\} \leq P\{X_i\} \leq P\{X_{hi}\} = 1 - q_{hi}^{s_t}$$

Then we apply theorem 1 (on page 5) to get bounds on $P\{\sum X_j = k\}$, or $P\{\mathbf{X} = k\}$:

$$P\{\mathbf{X} = k\} \leq P\{\mathbf{X}_{hi} \geq k\} - P\{\mathbf{X}_{lo} \geq k + 1\}$$

Expanding terms, we get the full recurrence relation:

$$P\{s_{t+1} | s_t, r_t, f_t\} \leq \sum_{s_{t+1} \leq i \leq N} \binom{r_t}{i} (1 - q_{hi}^{s_t})^i (q_{hi}^{s_t})^{r_t - i} - \sum_{s_{t+1}+1 \leq i \leq N} \binom{r_t}{i} \left(1 - q_{lo}^{s_t - f_t}\right)^i \left(q_{lo}^{s_t - f_t}\right)^{r_t - i} \tag{3}$$

We define the right hand side of relation 3 to be $\mathcal{R}(s_t, r_t, f_t, s_{t+1})$, "an upper bound on the probability that with s_t gossiping processes of which f_t are faulty, and with r_t processes that have not yet received the gossip, that s_{t+1} processes will receive the gossip this round."

5.3 Computation of Pbcast

In this section we show how to calculate a bound on the probability of a pbcast, in a particular round and state, ending in a failed state on round R: $\mathcal{F}_t\left(s_t, r_t, \overline{f}_t\right)$ ($\overline{f}_t$ is the total number of faulty processes that have failed prior to time t, $\overline{f}_t = \sum_{0 \leq i < t} f_i$). Given $\mathcal{F}$, the reliability of pbcast can be found by examining the value of $\mathcal{F}$ for the initial state of the protocol, or $\mathcal{F}_0(1, N-1, 0)$.

Values of $\mathcal{F}$ are calculated in the context of a predicate that defines whether a run of the protocol failed or not, according to its final state. The predicate, $\mathcal{P}(S, F)$, is defined over the total number of infected processes (S) (possibly including some faulty processes) and the total number of faulty processes (F). This predicate can be defined differently, depending on the use of pbcast. We give below the predicate used for a replicated data protocol, which is the one used in all plots in this paper. A failed pbcast, as far as the replicated data protocol is concerned, is one in which failures make it impossible to determine whether or not a majority of the processes delivered the pbcast. This predicate pessimistically totals all of the processes that may have been infected, so that a failed pbcast is one in which both the number of infected processes is less than a majority (with faulty processes counting as being uninfected) and the number of infected processes is at least a majority (with faulty processes counting as being infected):

$$\mathcal{P}(S,F) = S - F < \frac{N+1}{2} \bigwedge S + F \geq \frac{N+1}{2}$$

The calculation works backwards from the last round. For each round, we sum over the possible number of failures in this round and the number of infectious processes in the next round. This is done using the calculations for the next round and the recurrence relation, $\mathcal{R}$, in order to get the two following equations. The first equation calculates bounds on the probabilities for round R; the second equation calculates bounds for the previous rounds (here we take $\mathcal{P}(S,F) = 1$ if true and 0 if false):

$$\begin{aligned} \mathcal{F}_R\left(s_R, r_R, \overline{f}_R\right) &\leq \sum_{0 \leq f_R \leq s_R + r_R} P\{f_t\} \mathcal{P}(N - r_R, \overline{f}_R + f_R) \\ \mathcal{F}_t\left(s_t, r_t, \overline{f}_t\right) &\leq \sum_{0 \leq f_t \leq s_t} \left(P\{f_t\} \sum_{0 \leq s_{t+1} \leq r_t} \mathcal{R}(s_t, r_t, f_t, s_{t+1}) \mathcal{F}_{t+1}\left(s_{t+1}, r_t - s_{t+1}, \overline{f}_t + f_t\right)\right) \end{aligned} \quad (4)$$

We do not know the exact distribution of $P\{f_t\}$ because individual processes can fail with probabilities anywhere in $[0..\tau]$. However, we can apply theorem 2 (on page 5) to get bounds on the two equations above. For example, the bound for equation 4 is:

$$\mathcal{F}_t\left(s_t, r_t, \overline{f}_t\right) \leq$$
$$\sum_{0 \leq f_t \leq s_t} \left(\binom{s_t}{f_t} (\tau)^{f_t} (1-\tau)^{s_t - f_t} \max_{0 \leq i \leq f_t} \sum_{0 \leq s_{t+1} \leq r_t} \mathcal{R}(s_t, r_t, i, s_{t+1}) \mathcal{F}_{t+1}\left(s_{t+1}, r_t - s_{t+1}, \overline{f}_t + i\right)\right)$$

Given the parameters of the system and a predicate defining failed final states of the protocol, we can now compute bounds on the probability of pbcast ending up in a failed state.

5.4 Digression: an Extension to Pbcast

When the process which initiates a pbcast is not faulty, it is possible to provide stronger guarantees for the pbcast delivery distribution. By having the process which starts a pbcast send more messages, an analysis can be given that shows that if the sender is not faulty the pbcast will almost always be delivered at "most" of the processes in the system. This is useful because an application can potentially take some actions knowing that its previous pbcast is almost certainly going to reach most of the processes in the system. The number of messages can be increased by having the process that initiates a pbcast use a higher value for β, for just the first round of the pbcast. This extension is not used in the following computations.

6 Evaluation and Scalability

Our evaluation of pbcast is framed in the context of its scalability. As the number of processes increases, pbcast scales according to several metrics. First, the reliability of pbcast grows with system size. Second, the cost per participant, measured by number of messages sent or received, remains at or near constant as the system grows. Having made these claims, it must be said that the version of pbcast presented and analyzed makes assumptions about a network that become less and less realizable for large systems. In practice, this issue could be addressed with a more hierarchically structured protocol, but our analysis has not been extended to such a protocol. In this section, we will address the scaling characteristics according to the metrics listed above, and then discuss informally how pbcast can be adapted for large systems.

6.1 Reliability.

Pbcast has the property that as the number of processes participating in a pbcast grows, the protocol becomes more reliable. In order to demonstrate this, we present a plot (figure 2b) of pbcast reliability as the number of processes are varied between 10 and 60, fixing fanout and failure rates. For instance, the plot shows that with 20 processes the reliability is around 10^{-13}. The plot almost fits a straight line with slope $= -0.45$, thus the reliability of pbcast increases almost ten-fold with every two processes added to the system.

6.2 Message cost and fanout.

Although not immediately clear from the protocol, the message cost of the pbcast protocol is roughly a constant multiple of the number of processes in the system. In the worst cast, all processes can gossip to all other processes, causing $O(n^2)$ messages per pbcast. β will be set to cause some expected *fanout* of messages, so that on average a process should gossip to about *fanout* other processes, where *fanout* is some constant, in practice at most 10 (unless otherwise stated, $fanout = 7$ in the plots presented in this paper). To get a particular fanout per process, $\beta = fanout/N$. If all processes gossip, the expected total number of messages sent in all rounds is then $N(N\beta) = N(N(fanout/N)) = N fanout$. Chernov-Hoeffding bounds can be used to show that the actual number of messages sent is very unlikely to deviate much. See figure 2c for a plot of reliability verses fanout when the number of processes and other parameters is held constant. For instance, the plot shows that with a fanout of 7.0, pbcast's reliability is around 10^{-13}. In general, the plot shows that the fanout can be increased to increase reliability, but eventually there are diminishing returns for the increased message cost.

On the other hand, fanout can be decreased as the system grows, keeping the reliability at fixed level. In figure 2d, reliability of at least "twelve nines" ($P\{failed\} \leq 10^{-12}$) is maintained, while the number of processes is increased. The graph shows that with 20 processes a fanout of 6.63 achieves "twelve nines" reliability, while with 50 processes a fanout of 4.32 is sufficient. As can be seen, larger numbers of processes can be configured with a smaller fanout (which decreases the amount of work per-process) and still maintain the same level of reliability.

6.3 An unscalable system model.

Although the protocol operating over the system model is scalable, the system model is not. The model assumes a flat network in which the cost of sending a message is the same between all pairs of processes. In reality, as a real system scales and the network loses the appearance of being flat, this assumption breaks down. There are two possible answers to this problem. The first is to consider pbcast suitable for scaling only to mid-sized systems (perhaps with as many as 100 processes). Certainly, at this size of system, pbcast provides levels of reliability that are adequate for most applications. The second possible solution may be to structure pbcast's message propagation hierarchically, so that a weaker system model can be used which scales to larger sizes. The structure of such a protocol, however, would likely complicate the analysis. Investigating the problem of scaling pbcast to be suitable for larger numbers of processes is an area of future work.

7 Conclusion

To summarize, pbcast provides a combination of totally ordered delivery and bimodal reliability useful for voting-style protocols. All broadcasts are delivered in a consistent total ordering across the system, but are not delivered reliably to individual processes. Pbcast usually behaves as a quorum-style abcast. However, in certain failure modes pbcast instead guarantees a bimodal delivery: almost always, either most or few processes deliver a pbcast. Pbcast works better with increasing system scale: as more processes are added to the system it reliability rapidly increases. Moreover, pbcast scales in number of messages and latency. Each broadcast almost always requires roughly $N fanout$ messages.

References

[Bai75] Norman Bailey. *The Mathematical Theory of Infectious Diseases.* Charles Griffin and Company, London, 2 edition, 1975.

[CASD85] Flaviu Cristian, Houtan Aghili, H. Ray Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. In *Proceedings of the Fifteenth International Symposium on Fault-Tolerant Computing*, pages 200–206, Ann Arbor, Michigan, June 1985. Institution of Electrical and Electronic Engineers. A revised version appears as IBM Technical Report RJ5244.

[DGH+87] 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. In *The Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12. ACM, August 1987.

[Dij74] EW Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the ACM*, 17:643–644, 1974.

[GT92] Richard A. Golding and Kim Taylor. Group Membership in the Epidemic Style. Technical Report UCSC-CRL-92-13, University of California, Santa Cruz, May 1992.

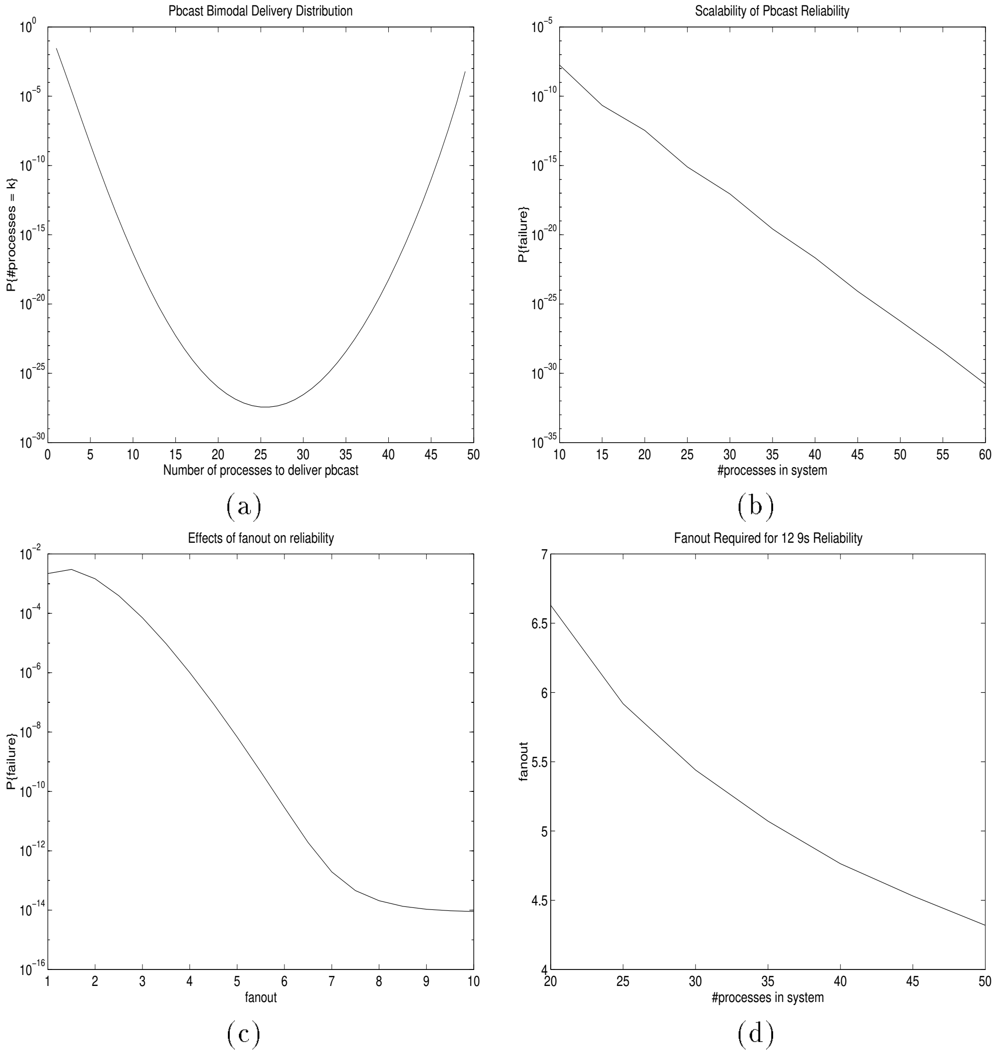

Figure 2: *Pbcast graphs: please note that several of these are semi-log graphs. (a) Computed pbcast bimodal delivery distribution, graphing number of processes (out of 50) that deliver a pbcast against the probability of that number of deliveries. (b) Scalability of pbcast reliability. Graphs the number of processes in the system against the probability of a failed pbcast. fanout is held constant (at 7), and R is increased as necessary. (c) Graph of reliability with different values of $fanout$. Pbcast reliability is graphed against $fanout$, with* 30 *processes. (d) Graph showing fanout being decreased as the number of processes grow. The required fanout to achieve 12 9's reliability is graphed against the number of processes in the system.*

A Applying pbcast's properties: a replicated data protocol.

Replicated data and synchronization protocols can be built using pbcast. We quickly describe here a replicated data protocol. Due to limitations of space and in keeping with the emphasis in this paper on the core pbcast protocol, we omit a full presentation. However, we do have a synchronization protocol that can be shown to be safe but only probabilisticly live, and a detailed treatment of both protocols is planned for a future publication.

The replicated data protocol maintains copies of a replicated variable at all processes. Updates to a variable are cumulative in that they into account the value of the variable prior to the update. Updates are made to the variable using pbcasts. When a process receives a pbcast'd update it buffers the update until the update "stabilizes" and is known to be valid or invalid. An update is valid exactly when it is delivered to a majority of the processes. Updates that are not valid are invalid. When a process knows that an update is valid or invalid, it considers the update to be stable. Prior to being stable, it considers the update to be unstable. Unstable updates are stabilized with a traditional gossip protocol which propagates unstable updates and the lists of processes to which they were and to which they were not delivered. This is called the gossip stabilization protocol. Assuming updates are delivered (or not delivered) at sufficiently large majorities relative to the number of faulty processes, the eventual convergence property guarantees that updates will eventually become stable at all non-faulty processes.

All processes in the system maintain a history of potential updates to a replicated variable, along with information about how many processes it was or was not delivered at. At any point, there are a set of potential updates with varying amounts of information regarding their stability. A process applies a valid update to a variable when it is known to be valid and there are no possible earlier, valid, unapplied updates. When a process finds that an update is valid (and thus delivered to at least a majority of the processes), it can also determine that there are no earlier, unknown, valid updates because such an update must have also been delivered to a majority.

Reads to the replicated variable can return the most recent known value of the replicated variable and the status of all pending updates. Another possibility for reads is to wait for all current pending updates to be stabilized and then eventually return the state of the variable at the logical time of the read. Both the reads just described return only local information and require no additional information. It is possible to speed up the stability protocol by causing extra rounds of gossip messages or by requesting the status of pending updates directly from other processes (or possibly all of them). Normally, the gossip stabilization protocol proceeds by "pushing" gossip to other processes. A read such as this can be visualized as "pulling" gossip to the process making the read[DGH+87].

The protocol is always safe: no two processes ever disagree about the validity or ordering of updates (although some may not have determined the validity of some updates yet). The protocol is probabilisticly live: if an update is delivered at approximately half the processes and some processes fail, it may not be possible to determine the update's validity, causing the protocol to become "frozen." However, pbcast's bimodal delivery property make such votes extremely unlikely to occur. When they do occur, the application must unfreeze the variable one way or another. One possibility is for the application to reset the value of the variable by introducing an update that does not depend on previous values of the variable. Thus any updates that may be preventing the system from making progress become "overwritten" and irrelevant, as soon as such an update become valid. Although such an overwriting update is made by only one process, that process may confer with other processes first before making the update.

As a final note, the replicated data can support *unsafe* reads on replicated variables. Such reads use the current knowledge about an unstable update to determine the conditional probability of it being valid or invalid. When it is known that a pbcast'd update has been delivered (or not delivered) at enough processes, the conditional probability of it being valid (or invalid) becomes very high. These guesses can usually be made with extremely high probabilities long before the updates all become stable, thus significantly decreasing the latency for the stabilization of updates but introducing the possibility of the reads being incorrect.

B Totally ordered Pbcast protocol

```
(* Local state: message buffer and
 * counter for generating unique
 * identifiers. *)
let buffer = {}
let id\_counter = 0


(* Initiate a pbcast. *)
to pbcast(msg):
   (* Create unique id for each message. *)
   let id = (my\_id, id\_counter)
   id\_counter := id\_counter + 1

   do\_gossip(time(),id,msg,\nrounds)


(* Handle message receipt. *)
on receive Gossip(timesent,id,msg,round):
   do\_gossip(timesent,id,msg,round)


(* Handle timeouts. *)
on timeout(time):
   (* Check for messages ready for
    * delivery.  Assumes buffer is
    * scanned in lexicographic
    * order of (sent,id). *)
   foreach (sent,id,msg) in buffer:
      if sent + \nrounds + 1 = time then
         buffer := buffer \\ (sent,id,msg)
         deliver(msg)


(* Auxiliary function. *)
to do\_gossip(timesent,id,msg,rnd):
   (* If have seen message already, do
    * nothing. *)
   if (timesent,id,msg) in buffer then
      return

   (* Buffer the message for later
    * delivery, and then gossip. *)
   buffer := buffer U (timesent,id,msg)
   set\_timer timesent + \nrounds + 1
```

```
(* If last round, do nothing more. *)
if rnd = 0 then return

foreach p in \processes
   with probability \rate
      send p Gossip(timesent,id,msg,rnd-1)
```