Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems

Torsten Hoefler
Department of Computer Science
ETH Zurich, Zurich, Switzerland
htor@inf.ethz.ch

Amnon Barak and Amnon Shiloh
Department of Computer Science
The Hebrew University of Jerusalem
Jerusalem, 91904, Israel

Zvi Drezner
College of Business and Economics
California State University
Fullerton, CA 92834, USA

***Abstract*—Large-scale parallel programming environments and algorithms require efficient group-communication on computing systems with failing nodes. Existing reliable broadcast algorithms either cannot guarantee that all nodes are reached or are very expensive in terms of the number of messages and latency. This paper proposes Corrected-Gossip, a method that combines Monte Carlo style gossiping with a deterministic correction phase, to construct a Las Vegas style reliable broadcast that guarantees reaching all the nodes at low cost. We analyze the performance of this method both analytically and by simulations and show how it reduces the latency and network load compared to existing algorithms. Our method improves the latency by 20% and the network load by 53% compared to the fastest known algorithm on 4,096 nodes. We believe that the principle of corrected-gossip opens an avenue for many other reliable group communication operations.**

***Keywords*-gossip algorithms; reliable broadcast**

I. Introduction

Maintaining group communication in parallel and distributed computing environments in the presence of failures is very important for parallel computations and system management. Such environments range from loosely connected ad-hoc sensor networks, such as the Internet of Things (IoT), through datacenter cluster computers, to tightly-coupled highly-reliable parallel supercomputers. In all these cases, endpoints can fail at any time. Specifically, the probability that any node in a running application fails, grows with the number of nodes, thus large-scale parallel applications must continue to work despite node-failures. These applications usually build on group communication to coordinate their computations and manage their resources. For example, distributed operating systems [5], schedulers [7], [23], databases [11], and parallel programming and runtime environments [20] require fast global group communication.

Different use-cases have different reliability requirements. While more specialized systems such as schedulers or operating systems can often accept a degree of inconsistent information, others, such as parallel programming frameworks, need to offer consistent semantics to the programmer [20]. If, for example, an update to a scheduling system is lost, then the quality of the scheduling decisions may be reduced, but if a message that is part of a broadcast operation in the Message Passing Interface (MPI) programming system is lost, then the program might deadlock.

This paper focuses on efficient broadcasting as an important primitive on its own [4], but hopes that other group communications can be derived from its results. Reliable broadcasts that tolerate r failures can be implemented using r-connected communication graphs where each node can be reached through r vertex-disjoint paths from the root. However, this adds significant communication overheads, since the total work is $\mathcal{O}(rn)$ on n nodes - assuming a simple network model with unit-time latency and message injection and reception times. Alon, Barak, and Manber propose a scheme for $r = n - 1$ with a best failure-free latency of $\mathcal{O}(\log n)$ and a worst-case latency of $\mathcal{O}(n)$ [1] but total work of $\mathcal{O}(n^2)$, which makes it prohibitively expensive in practice. The major benefit of r-connected graphs is that they do not require any failure-detector, but this comes at a cost: if more than r components fail, then reliability may be compromised. In performance-critical scalable systems, such schemes are only practical for small values of r due to the high work overhead. The most typical r-connected graphs are binomial graphs [10] with $r = \lceil \log_2 n \rceil$.

If a failure-detector is available, then self-healing structures can be used to route around failing nodes [3], [8], [18]. Yet, these structures require a latency of at least $2 \log_2 n$ and a total work of at least $2n$ messages to guarantee that all alive (active) nodes are reached. However, failure-detectors cause additional overheads, e.g.; control messages, can cause high overheads in the case of failures. Also, in practice, timeouts are complex to configure for reliably distinguishing slow nodes or congested networks from failed nodes. The runtime of these self-healing algorithms varies between $\mathcal{O}(\log n)$ and $\mathcal{O}(n)$, depending on the number of failures.

We improve upon these previous results by *combining Monte Carlo style gossiping with a parallel deterministic recovery protocol* in order to provide a set of reliable Las Vegas style broadcast algorithms at various consistency levels. Our algorithms that we call corrected-gossip require no failure detector and outperform existing approaches in terms of latency and number of messages (total work).

II. Failures, Performance and Consistency

In this paper we consider only crash failures where nodes run correctly unless they fail, in which case they do not make further relevant actions. We model the system as a set of nodes P where the number of nodes $|P| = N$. We assume that P does not change during the execution of a program (it can be considered as a static name space of addresses from $0 ... N - 1$). At any point in time, the set P consists of two disjoint subsets: A - all active nodes, and F - all inactive (failed) nodes. We denote $|A| = n$, thus $|F| = N - n$. It is assumed that nodes might fail (transit from A to F) at any time, but nodes may only become active (transit from F to A) in a controlled manner that prevents them from re-joining any former activities.

We assume that communications are reliable; that there are no disrupting network failures (so even if an individual link fails, the network as a whole can recover); that no messages are lost; and any node can communicate with any other node in latency L. Furthermore, sending or receiving a message delays a node by O. Our communication model is equivalent to the well-known LogP model [9] for $o = O$ and $g \ll o$, $|P| = n$. We assume that messages can be sent and received at the same time. For simplicity we assume that L is divisible by O.

A broadcast in a system with n active nodes delivers a message M from a single node to clients on all other $n - 1$ nodes [4]. This broadcast can support various properties such as ordering and fault tolerance [4]. Broadcasts are *posted* by a root node which invokes `bcast(`M`)` to broadcast the message M. The broadcast call returns immediately with a status that can be checked for completion later. The reliability of broadcasts can be described with the following four standard guarantees for non-faulty nodes: (I) *integrity* (all received messages have been sent), (II) *no duplicates* (each sent message is received by only one node), (III) *nonfaulty liveness* (messages broadcast by a live node are received by all live nodes), and (IV) *faulty liveness* (a message sent by a failed node is either received by all live nodes or by none of them) (cf. [4], [17]). We provide no guarantees for messages received by faulty nodes. We call a broadcast algorithm *consistent* if it guarantees properties III and IV.

We generalize nonfaulty liveness and faulty liveness in the following way: (III) a message sent by an active node is received by at least $f(n)$ nodes, and (IV) a message sent by a failed node is either received by at least $f(n)$ live nodes or by none of them. The function $0 \leq f(n) \leq n$ determines the consistency level. We call a broadcast weakly consistent if $Pr(f(n) > n - \log n) = 1 - \mathcal{O}(1/n^{\beta})$ for $\beta \geq 1$ and strongly consistent if $f(n) = n$. For many management tasks, weakly consistent states are sufficient.

Dynamic load balancing, for example, works sufficiently well with outdated information delivered by a weakly consistent broadcast. Such schemes often use highly-resilient and simple gossip algorithms for communications. However, weakly consistent broadcasts can complicate reasoning about systems because nodes that did not receive a given broadcast message may be in a different state (e.g., stage of the algorithm) than nodes that received the message. This is often unacceptable, especially in parallel programs written in the common Bulk Synchronous Parallel [22] programming mode. Thus, depending on the use-case, consistent broadcast algorithms may be required. In this paper we show how to transform an inconsistent probabilistic gossip protocol into a consistent deterministic protocol with minimal overheads.

III. Corrected-Gossip

Gossip algorithms have been widely successful in various contexts that did not require strong consistency. Yet, they become rapidly inefficient once about 50% of the nodes were reached, because messages are more likely to be sent to nodes that were already reached. We design three different protocols based on the idea of combining randomized and deterministic algorithms for improving the broadcast latency. These three algorithms allow us to choose various tradeoffs between consistency, simplicity, and performance. The three algorithms are: (1) *opportunistic*, which applies the correction without checking for completion; (2) *checked*, which runs the correction until all nodes received the message, provided that no nodes fail during the correction; and (3) *fail-proof*, which applies the correction and guarantees that all active nodes receive the message, provided that no more than f nodes fail during the operation. *Our algorithms do not require multicast, failure detectors, timeouts, acknowledgments, or reconfiguration procedures.*

All algorithms start with a single item of information (color) at the root node and run T gossip rounds to spread the color randomly. Drezner and Barak [12] show that for $T \geq 1.639 \cdot \log_2(N)$, each node is reached with high probability for unsynchronized gossip. Yet simulations show that this scheme is not strongly consistent even in the failure-free case. For example, for N=1,000 and T=17, the gossip colors all the nodes only 95.1% of the time.

To improve consistency, we employ various correction protocols to distribute the data deterministically. The simplest such protocol arranges all N nodes in a ring where each colored node i sends correction messages to its neighbors, starting with $(i + 1) \mod N$, $(i + 2) \mod N$.

A. The Gossip Phase

All our algorithms begin with a gossip phase, starting with a single colored node (the root). Each colored node sends a message to some other random node every O units of time. Messages carry a virtual time-counter, so that all the (colored) nodes stop sending new messages at time T. The gossip phase continues for another L+O time units (between T and $T + L + O$), allowing messages that are already on their way to be received before starting the correction phase. Pseudo-code for the gossip phase is shown in lines 5–6 of Algorithm 1 in Appendix A.

Consider a system with N nodes of which $n \leq N$ are active. For brevity, we call the root node and nodes that have received the message during the gossip phase *g-nodes* and nodes that first received the message during the correction phase, *c-nodes*.

Let $c(t)$ be the expected number of g-nodes at time t. Note that at time $t = 0$, $c(0) = 1$, while at $t < 0$, $c(t) = 0$.

Lemma 1: The expected g-node count at time $t + O$ is

$$c(t+O) = c(t)+(n-c(t)) \left[1 - \left(1 - \frac{1}{N-1}\right)^{c(t-L-O)}\right]. \quad (1)$$

Proof: Consider the creation of g-nodes as a coloring problem where all nodes but the root start uncolored and nodes reached by a message are colored. Since only nodes that are colored at $t-L-2O$ can color a node at time t, and since nodes do not select themselves, the probability that an uncolored node did not receive a message at time $t + O$ is $\left(1 - \frac{1}{N-1}\right)^{c(t-L-O)}$. As there are $n-c(t)$ active non-colored nodes just before $t + O$, the lemma follows. ■

Note that $\lim_{t\to\infty} c(t) = n$, so for any desired δ ($0 < \delta < 1$), by selecting t such that $c(t) \geq n-\delta$, the gossip phase can itself be used to color all the nodes with probability $1 - \delta$.

Figure 1 shows the expected number of colored nodes for $L = O = 1$, $n = N = 1,024$. It is easy to see that the rate of coloring new nodes (i.e., the first derivative of the $c(t)$ function) increases up to around 512 nodes ($t \approx 18$), then it decreases towards zero. This is intuitive since when about 50% of the nodes are reached, most random messages will reach already-colored nodes. The lowest progress is reached when one needs to wait for the last few nodes to be colored.

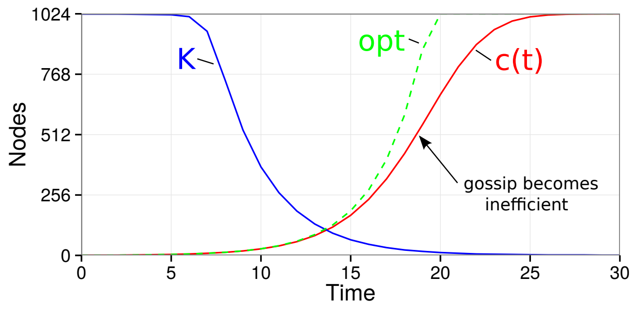


Figure 1: Expected number of g-nodes $L = O = 1$, $N = n = 1,024$, the optimal tree ("opt") would require $t = 20$

This observation motivates the use of a deterministic correction once the random gossip becomes ineffective.

1) Outcome of the Gossip Phase: Once the gossip phase ends, we view all the nodes as ordered on a ring (node-id modulo N for addressing). As we shall see, it is useful at this stage to find and bound the length of the longest uncolored chain of consecutive nodes as this determines the needed time to reach all the nodes in the correction phase.

Figure 1 also shows the 99%-probable longest uncolored chain, K, as a function of time. We will later use K to compute the optimal number of gossip rounds for our algorithms. While the results in Figure 1 require a time-consuming simulation, below we present a rough approximation function that can be evaluated quickly with several simplifying assumptions.

Let T denote the time when the gossip phase stops sending new messages. When the correction phase starts, the average number of colored nodes is $c(T + L + O)$, so the probability that an arbitrary node is colored is $c(T + L + O)/N$.

For any $K \geq 0$, consider a specific node i out of all N nodes. The probability that node i is colored, nodes $i + 1, \ldots, i + K$ are uncolored and node $i + K + 1$ is colored is:

$$p(K) = \frac{c(T + L + O)^2(N - c(T + L + O))^K}{N^{K+2}}.$$

Since there are N possible sequences of K consecutive non-colored nodes, assuming independence, the probability that such a sequence exists is:

$$\alpha_K = 1 - (1 - p(K))^N.$$

The probability p_K that the maximal sequence of non-colored nodes is K, is equal to the probability that a sequence of K exists and longer sequences do not exist, which is:

$$p_K = \alpha_K \prod_{i=1}^{N-K-1} \left(1 - \alpha_{K+i}\right). \quad (2)$$

B. Opportunistic Corrected-Gossip

In our first algorithm, Opportunistic Corrected-Gossip (OCG), the gossip phase is followed by a correction phase, where the g-nodes send information in both directions alternately along a virtual ring for a fixed time, C. Note that the c-nodes do not send any messages. Note also that already-colored nodes ignore all further messages. Algorithm 1 in Appendix A shows the detailed procedure as performed by each node.

Figure 2 shows an example with 10 nodes, $L = O = 1$, $T = 2$, and $C = 6$ for a broadcast originating at node 1. Dark nodes are g-nodes and light nodes are c-nodes or unreached nodes. Dashed lines illustrate gossip messages and their start and end-times. Solid lines represent correction steps and their start and end times. This is an artificially-poor example in order to demonstrate a case where OCG fails to reach all the nodes (in this case node 8).

Claim 1: OCG can be easily extended to provide (I), integrity; and (II), no duplication.

Proof: In a system that issues more than one broadcast, nodes might receive messages from earlier broadcasts as well as multiple messages of the same broadcast. Both can be prevented by counting the number of broadcasts that were started by each root node: the initiating root node can increment this counter before calling `bcast()` and each message can carry this counter. Each node can keep a received-bcast counter, $c[i]$, per root-node i, then discard all messages with root-node i and a counter smaller or equal than $c[i]$. When new nodes join, they should run a special protocol to reset their $c[i]$ for all active nodes. ■

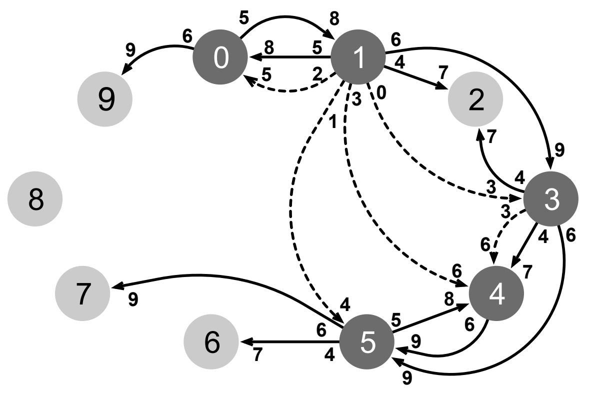


Figure 2: Opportunistic Corrected-Gossip example for $N = 10$, $L = O = 1$, $T = 2$, $C = 6$; g-nodes are darker

Claim 2: By selecting large enough values of T and C, we can reduce the probability that the correction phase fails to reach all the nodes below any desired δ.

Proof: If the maximal chain of consecutive uncolored nodes between any two g-nodes is K, then the correction phase of OCG will reach (and color if active) all the nodes in $KO + L + O$. Let $\overline{K}$ denote the smallest value of K for which $\sum_{i=K+1}^{N-1} p_i < \delta$ (see Eq. (2) for definition of p_i). Note that $\overline{K}$ is a function of N, n, T, L, and, δ. Once selecting a value for T, we can compute $\overline{K}$, then set $C = \overline{K}O + L + O$ (since $\overline{K}$ is only an estimate, in practice we recommend adding another O to C). ■

Corollary 1: By selecting T and C accordingly, OCG can be adjusted to be weakly consistent or strongly consistent with high probability guaranteeing properties (III) and (IV).

Selecting the number of gossip steps: We select T to minimize the overall latency $T + L + O + \overline{K}O + L + O$:

$$T_{opt}^{OCG} = \underset{T}{\operatorname{argmin}}(T + 2L + (2 + \overline{K})O) \ . \tag{3}$$

To approximate T using Equation (3), first we pick a δ which is the probability to not reach 100% of the nodes, then compute $\overline{K}$ and T_{opt}^{OCG}. Note that more than one value of T can produce the minimum. In that case, we recommend to select the T that can withstand the largest reduction in δ.

To demonstrate the accuracy of the above approximation, we simulated 10^6 runs of OCG with $L = O = 1$ on $N = n = 1,024$ nodes for various values of T and measured the total time to reach all the nodes. We repeated this experiment 10 times and in Figure 3 we plot (in dots) the maximum of all observed total-times for each T. The figure also shows (in solid line) the corresponding analytical results, based on T_{opt}^{OCG} (Equation (3)).

Assuming that a system is expected to run the OCG algorithm m times and that we wish the overall chance of failure to be under ψ, we can achieve this by selecting a δ such that $1 - (1 - \delta)^m \leq \psi$, or $\delta \leq 1 - (1 - \psi)^{\frac{1}{m}}$.

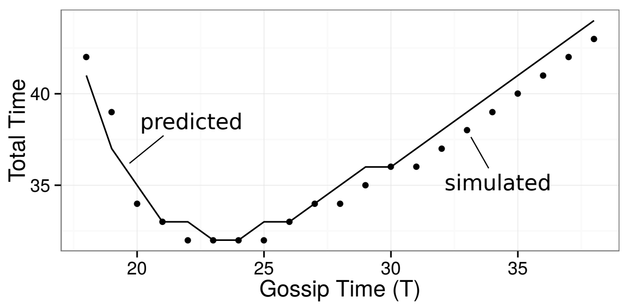


Figure 3: Opportunistic Corrected-Gossip evaluation

Figure 3 shows that the approximation can predict the optimal gossip time, $T = 24$, rather reliably. Here we choose $\delta = 1-(1-0.5)^{\frac{1}{10^6}} = 6.93 \cdot 10^{-7}$ to guarantee with about 50% probability that all 10^6 runs of the algorithm are successful - and observed a maximal difference of only one time-step in all the experiments.

Discussion: As our equations are only approximations, we recommend adding a small margin to the selected T. In all our experiments, we added one O to T.

The design space for the correction phase is large. OCG is meant to minimize the latency in networks with a high global bandwidth, assuming that $O \leq L$. In contrast, when $O > L$, one could utilize c-nodes as additional message sources in the algorithm. In the extreme case, a g-node could send a message which is forwarded by a chain of c-nodes until another g-node is reached. This strategy, which is beyond the scope of this paper, could reduce the number of messages and thus the total work.

C. Checked Corrected-Gossip

The Checked Corrected-Gossip (CCG) algorithm ensures that all active nodes are reached, provided that no nodes fail while the algorithm runs. Each node completes the correction phase once it "knows" that it is no longer needed for coloring all the active nodes. After the gossip phase We differentiate between c- and g-nodes: c-nodes do not send any messages in this algorithm. Each g-node i begins by sending a "forward" message to its successor $i + 1$, then O units later it sends a "backward" message to its predecessor $i - 1$, then to $i + 2$, $i - 2$, $i + 3$, $i - 3$, etc. (all modulo N).

Each message is marked whether it is a forward or a backward message. All g-nodes monitor all received correction messages and record the distance of the sender of the first forward and first backward messages as $m_{\triangleright}$ and $m_{\triangleleft}$ respectively. Note that because the gossip phase continues for another $L + O$ time-units in order allow all gossip-messages to arrive, we know that these first messages are also sent by the closest neighbouring g-nodes from each side. Once a g-node sent a forward message to distance $m_{\triangleleft}$, it stops sending further forward messages. Similarly, once a g-node sent a backward message to distance $m_{\triangleright}$, it stops sending further backward messages. Once a g-node no longer needs to send neither forward nor backward messages, it exits the algorithm. Algorithm 2 in Appendix A shows the detailed procedure as performed by each node.

The CCG protocol is guaranteed to reach all active nodes, assuming that no node fails during the operation. However, in some unfortunate sequences of events that include node-failures during the correction-phase, some nodes might become stuck and fail to complete the algorithm. Despite this improbable inconsistency, this protocol is practical and comparable to a two-phase commit where a leader-failure can result in a similar situation [15].

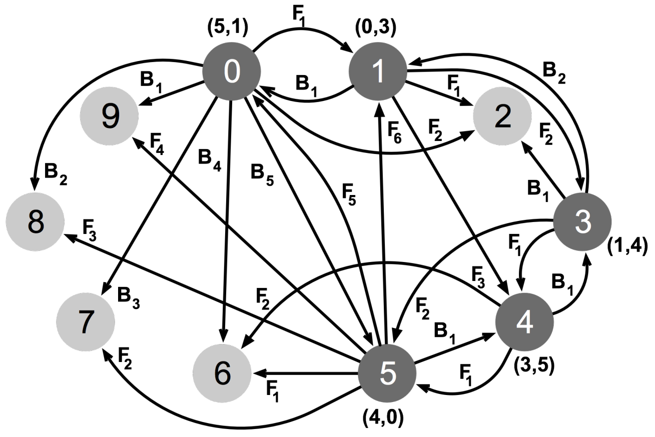


Figure 4: Checked Corrected-Gossip example

Figure 4 shows an example with 10 nodes, $L = O = 1$ and $T = 4$. We omit the gossip part, which is already shown in Figure 2; dark nodes are g-nodes. We only show forward (F) and backward (B) messages of the correction phase. Their subscript indicates the sending order from their respective source g-node. The tuple (x, y) next to each g-node indicates $(m_{\triangleleft}, m_{\triangleright})$. Observe that all nodes are reached after 2 correction steps $(+L/O + 1)$ but the algorithm exits after 5 steps $(+L/O + 1)$, when all g-nodes received a message from their closest g-nodes in both directions.

Claim 3: CCG provides integrity (I), no duplication (II), and is strongly consistent (provides (III) nonfaulty liveness and (IV) faulty liveness) with $f(a) = a$ if no node fails during the correction phase.

Proof: (I) and (II) are similar to Claim 1 and (IV) does not apply due to the assumption that no node fails while the algorithm runs.

We prove (III) by contradiction. The proof is intuitive and does not require the formal details of Algorithm 2. We assume that node i is not reached after the algorithm ends while at least one other node delivered the message. Obviously, node i cannot be a g-node.

Let a and b be the closest g-nodes in i's forward and backward directions on the virtual ring (note that if the gossip phase failed to reach any node, then $a = b = root$, but we do not require a and b to be distinct). Both, a and b enter the while loop on line 10 of Algorithm 2.

Besides node i, only uncolored nodes can be in between a and b (otherwise, a and b would not be the closest g-nodes). Node a will send backward messages to $a-1$, $a-2$, ... until it learned about b and sent a message to b. Similarly, node b will send forward messages to $b+1$, $b+2$, ... until it learned about a and sent a message to a. Because i is in between a and b, it must have received two correction messages. ■

Selecting the number of gossip steps: Similarly to OCG, we select T to minimize the latency. However, since the algorithm only ends after each node learned about its surrounding g-nodes, the time to execute the correction is about twice as long, $T + L + O + 2\overline{K}O + L + O$:

$$T_{opt}^{CCG} = \underset{T}{\operatorname{argmin}}(T + 2L + (2 + 2\overline{K})O) \ . \tag{4}$$

Note that more than one value of T can produce the minimum, so we recommend to select among them the T that can withstand the largest reduction in δ.

To demonstrate the accuracy of this approximation, we simulated 10^6 runs of CCG on $N = n = 1,024$ nodes for various values of T and measured the total time to reach all the nodes and complete the algorithm. We repeated this experiment 10 times and in Figure 5 we plot (in dots) the maximum of all observed total-times for each T. The figure also shows (in solid line) the corresponding analytical results, based on T_{opt}^{CCG} (Equation (4)). As before, we used $\delta = 1-(1-0.5)^{\frac{1}{10^6}} = 6.93 \cdot 10^{-7}$ in our approximations. The figure shows that the approximation can predict the optimal gossip time $T = 25$ (for $L = O = 1$) rather reliably and we have observed a maximal difference of only two time-steps in all the experiments.

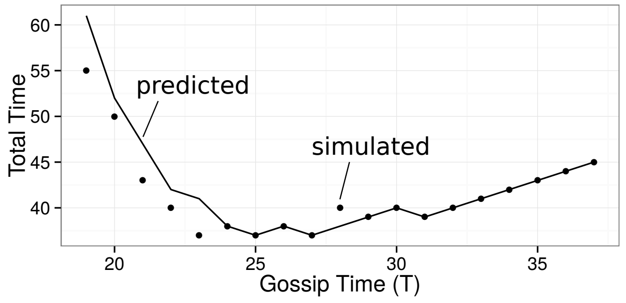


Figure 5: Checked Corrected-Gossip evaluation

Discussion: Similar to OCG, we could use different algorithm variants for the correction step. For example, if a failure detector between neighboring nodes exists, then the algorithm could forward a single message along the ring, skipping failed nodes.

Since some nodes may not be reached if a node fails during the correction step, CCG is not strongly consistent.

The next section introduces failure-proof corrected-gossip (FCG) derived from CCG.

D. Failure-proof Corrected-Gossip

The Failure-proof Corrected-Gossip (FCG) algorithm is designed to overcome up to f online failures (while the algorithm is running). We show that at the conclusion of the algorithm, either all remaining active nodes are colored or none. Each node completes the correction phase of FCG once it "knows" that it is no longer needed either for the coloring of all active nodes or for allowing other nodes to complete.

First we make an observation of a necessary but insufficient condition for an f-resilient broadcast. While it is not strictly needed, it may help to understand the algorithm:

Observation 1: In a reliable broadcast with guaranteed f-resilience, no colored node exits the operation before ensuring that either itself or at least one of $f+1$ other nodes will distribute the message to all remaining uncolored nodes.

Proof: Assume a colored node x exits before it knows about $f+1$ other nodes that will deliver the message. Let $F \subset N$ denote the set of nodes that fail during the operation ($|F| \leq f$). Let C denote the set of all nodes that are colored right after x exits and all of x's messages have been delivered but before any new node will be colored. Since x exited, $x \notin F$ (failure after exiting is not relevant since x cannot influence the algorithm's state anymore). Now, if $C \subset F$ then no other node in $N \backslash F$ will receive the message. ■

The FCG algorithm is similar to CCG but runs until each node knows that it has done its part, including to ensure that at least $f+1$ other g-nodes exist, so that at least one of those will not fail and will be able to continue propagating the color to fulfill Observation 1. In practice, each node ensures that at least $f+1$ other g-nodes exist around it in each direction, which is even stronger than what Observation 1 requires. FCG performs one more round than CCG (the finalization round), where once a g-node knows about f other g-nodes in any direction, it informs the nearby nodes in the opposite direction about them in order to help them to exit.

In the very unlikely case that less than $f+1$ g-nodes remain active, convergence might not be achieved with g-nodes alone. In this case, c-nodes wait for a timeout, then enter SOS mode where they send messages to all the nodes, which then themselves enter SOS mode. The probability of the SOS case can be reduced arbitrarily by increasing the gossip time, T (similar to OCG as shown in Claim 2), so this pathological case is not relevant in practice and was never observed in our experiments. Yet, it must be included to ensure FCG's correctness in the extremely unlikely worst-case.

Algorithm 3 in Appendix A shows the detailed procedure as performed by each node. Specifically, the gossip phase (lines 6-7) is similar to OCG and CCG. The first receive (line 4) can also receive correction messages. Lines 8-10 handle the SOS case and all c-nodes enter lines 11-14 where they wait for either the final message or trigger the SOS after a timeout. All g-nodes send messages forward ($\triangleright$) and backward ($\triangleleft$) (line 27) as in CCG, yet now they include the array $k_\triangleright$ or $k_\triangleleft$ in their backward and forward messages, respectively. These arrays accumulate up to the $f+1$ closest g-nodes that are known to be alive in the respective direction: they are updated with the source of forward or backward messages as well as with the transitive information received from the source g-nodes (lines 21-22). The functions $\measuredangle_\triangleright(x)$ and $\measuredangle_\triangleleft(x)$ sort the k nodes in x by distance from the calling node on the ring in either forward or backward direction, respectively. Each node that receives an SOS message also enters SOS mode (line 23).

Once a g-node learns about f other g-nodes from either the forward or the backward direction, it enters the finalization round with respect to that direction and sends final messages backwards or forwards accordingly, starting again from its immediate neighbors (line 24). Once a g-node passes the (f+1)-furthest g-node in either direction, it stops sending in that direction (line 25). Once a node stopped sending in both directions, it terminates. If a node sends to itself without finding f+1 g-nodes in either direction, it invokes the SOS protocol (line 26).

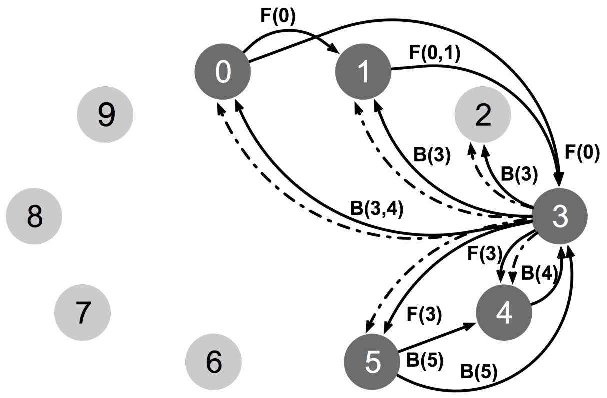


Figure 6: Failure-proof Corrected-Gossip example

Figure 6 shows an example with 10 nodes, $L = O = 1$ and $f = 1$. Dark nodes are g-nodes and for clarity, we only focus on correction messages related to node 3. Shown in parentheses are the direct and indirect nodes included in each message. In this example, node 3 terminates after receiving F(0,1), B(4) and B(5). Before terminating, it sends final messages (dashed).

The next two lemmas are required for the proof of correctness.

Lemma 2: If any g-node, k, delivers without SOS then messages will be sent to all nodes.

Proof: Since k is a g-node and not in SOS, it can only exit once it stops sending in both the forward and backward directions (line 18). Wlog, consider the forward direction. Before stopping to send forward, a g-node must detect $f+1$ other g-nodes (line 24) in the forward direction (or enter SOS mode in line 28). Furthermore, it will have sent final messages to at least all those $f+1$ g-nodes before exiting.

More formally, we say that a node covers another set of nodes if it will send final messages to these nodes. Each g-node that does not enter SOS mode, covers at least $f+1$ g-nodes of the logical ring of size N. We now show by induction that the whole ring must be covered if a single g-node exits without SOS mode. For the base-case, assume that node A_1 terminates, then it has covered all nodes up to its $f+1$st g-node B_1. Since at most f g-nodes can fail, there will be at least one remaining g-node R_1 among the covered nodes. In the induction step, we set $A_{i+1} = R_i$ (the remaining g-node from the previous step) which is guaranteed to cover $f+1$ g-nodes up to B_{i+1}. It is easy to see that each induction step reduces the set of uncovered nodes by at least one. ■

Lemma 3: If any node receives a message, it will deliver.

Proof: There are three types of messages: (1) forward or backward messages, (2) SOS messages, (3) gossip messages. By definition, a c-node received (1) or (2), but not (3). If a c-node accumulates through (1) information about at least $f+1$ different g-nodes, then it will deliver (line 13), otherwise it will eventually trigger SOS (line 14). A g-node received (3) by definition and during the correction phase it will either discover $f+1$ other g-nodes in both directions in the loop at line 18 or trigger SOS (line 28). Either way, it will deliver. ■

Corollary 2: All nodes that receive a message eventually terminate.

Claim 4: FCG provides integrity (I), no duplication (II), and is strongly consistent (provides (III) nonfaulty liveness and (IV) faulty liveness) with $f(a) = a$ if no more than f nodes fail during the correction step.

Proof: (I) and (II) are similar to Claim 1. We proceed to prove (III) which assumes that the root r does not fail during the operation.

To prove (III) by contradiction, we assume a node $i \neq r$ does not deliver while a node $j \neq i$ delivers the message. Thus, i is not a g-node because g-nodes always deliver (Lemma 3). Further, no message could have been sent to i, or else it would have delivered (Lemma 3). This implies that no node returned from SOS because it would have sent to all nodes before delivering. This contradicts Lemma 2. ■

Corollary 3: FCG can withstand any number of failures happening before the algorithm or during the gossip phase.

This can be shown using the techniques above and the full proof is omitted for brevity.

When is the SOS protocol needed: To determine if the SOS protocol is needed during normal operations, let us look at the case where the SOS protocol is disabled as if it would not exist[1]. Let us define an "r-node" as a g-node that does not fail while the FCG algorithm is running.

Lemma 4: If the SOS protocol is disabled, if any r-node receives information about f other g-nodes in one direction ($\rhd$ or $\lhd$), then all r-nodes will receive information about $f+1$ g-nodes in that direction.

Proof: Wlog, consider the $\rhd$ direction. As soon as an r-node, A, receives information in the $\rhd$ direction about f or more other g-nodes, it will start sending final $\rhd$ messages and will not stop until it either reaches itself or receives information about at least $f+1$ g-nodes in the $\lhd$ direction. If A reaches itself, then the claim is evident. Otherwise A will send final $\rhd$ messages to all those $f+1$ g-nodes that it learned about through $\lhd$ messages (including all the nodes in between): since we assume no more than f online failures, at least one of those $f+1$ nodes, B_1 is an r-node. Using induction it is then similar to show that once all the r-nodes in the range between A and B_i are covered, either A receives a final $\rhd$ message from B_i or some new r-node, B_{i+1}, receives it, where the distance between B_{i+1} to A is shorter than the distance between B_i to A. Within a finite number of steps, that distance shrinks to 0, so before that happens all r-nodes are covered all the way to A. ■

Claim 5: If the number of g-nodes is at least f^2+f+1, FCG will complete even if the SOS protocol is disabled.

Proof: While ignoring any c-nodes and uncolored nodes in between, let us look only at sequences of consecutive g-nodes. If any such sequence of $f+1$ r-nodes exists, then its furthest end in the $\rhd$ direction will be informed in the $\rhd$ direction about the other f r-nodes; and its furthest end in the $\lhd$ direction will be informed in the $\lhd$ direction about the other f r-nodes. The above lemma will therefore be fulfilled in both directions, so every r-node will be informed about $f+1$ nodes in each direction and complete the algorithm. It can easily be seen that no placement of f non-r-nodes within a closed chain of f^2+f+1 (or more) g-nodes can prevent the formation of an $f+1$-long consecutive sequence of r-nodes (the worst case is to place non-r-nodes in positions 0, f, $2f$, $3f$, …, $(f-1)f$). ■

Similarly to OCG and CCG, we can approximate the number of needed gossip rounds analytically. The rather technical derivation of an upper bound is available in Appendix B and is not critical for the presentation of the algorithms.

IV. Performance Analysis

The performance of fault-tolerant broadcast algorithms is often characterized by two metrics: (1) the total number of messages sent (work) to complete the algorithm and (2) the latency needed to complete the operation. Both metrics depend on T in all the variants of corrected-gossip, as well as on C for OCG. The worst-case latency for all algorithms

[1]This analysis requires that a node may not trigger the SOS mode in line 28 of Algorithm 3 when sending to itself

requires at least $N - 1$ steps in the very unlikely case that the gossip reached no nodes (or $T = 0$). Thus, we limit our analysis to the more interesting and common case that only needs a small number of additional correction steps once T is well chosen.

A. Simulation Setup

Since an accurate analytical method for assessing the latency in the LogP model is infeasible, we use Monte Carlo style simulations to predict it, then use the analytical approximations in Section III to determine the optimal T. For GOS (see below) and OCG, we select $\delta = 6.9315 \cdot 10^{-7}$ to have at least 50% chance of consistency in all 10^6 trials. For CCG and FCG, we aim to minimize the latency.

Our simulation creates failures by marking nodes as inactive either at the beginning or during the execution. We simulate various numbers of originally-inactive nodes as well as up to f nodes that fail during the simulation.

We use realistic LogP parameters that we gathered from Piz Daint, a Cray XC30 supercomputer with 5,272 nodes connected by the Aries interconnect. All nodes start at time zero and we record the latency until the last active node receives the broadcast and completes the algorithm. We also record the total work (the number of messages sent) as well as the consistency (the percentage of nodes reached).

B. Other FT Broadcast Algorithms

Existing algorithms can be categorized into three classes: probabilistic, restarting, and redundant. Probabilistic algorithms use randomness to reach nearly all nodes with high probability. They are the only algorithms that provide only weak consistency. Restarting algorithms restart if they detect a change in the virtual topology due to a failure and redundant algorithms send messages redundantly $f+1$ times to overcome up to f failures. We describe three specific existing algorithms to represent each class.

1) Gossip: The simplest fault-tolerant broadcast is a probabilistic gossip algorithm (GOS) where each node, once it received a color, sends it to random peers until a fixed global time is reached [16]. As described before, the efficiency of this approach rapidly declines after 50% of the nodes are colored. Nevertheless, it can be a very effective loosely-consistent fault-tolerant broadcast algorithm.

2) Buntinas' FT Broadcast: Buntinas' fault tolerant consistent broadcast (BFB) was proposed as part of a consensus algorithm in MPI's agreement protocols [8]. This algorithm supports an arbitrary number of failures by using a dynamic tree where each message contains the information about the children in the next levels. Once all children have been reached, the delivery is acknowledged back to the root. The protocol relies on failure detectors to find out if any node failed before or during the operation. If a failed node is detected and the message cannot be delivered, a NACK message is immediately propagated to the root which in turn re-starts the whole operation with a modified tree.

The BFB algorithm restarts after each new node failure and has a minimal latency (in case of no online failures) of $T^{BFB} = 2(2O + L)\log_2 N$.

3) Binomial Graphs: Binary (dissemination) graphs (BIGs) are a special case of perfectly connected graphs [10] that can tolerate up to $\log_2 P - 1$ failures using static routing. In these networks, each node p is connected to a neighbor set $N = \{p + 2^x \mod P | 0 \leq x \leq \lfloor \log_2 P \rfloor\}$ of $\log_2 P$ other nodes. Each node sends the first received message to all its neighbors and the communication finishes after $\log_2 P$ rounds. Each node can start forwarding messages as soon as it receives them. Thus in the optimal case, the completion time is equal to a single binomial tree where the last node receives $\log_2 P$ more messages. Assuming no faults, the communication finishes after $T^{BIG} = (2O + L)\log_2 P + O \log_2 P$.

This is similar to multiple random spanning trees as proposed by Birman et al. [6] but is more resilient because it guarantees $\log_2 P$ vertex-disjoint paths.

C. Expected Failures

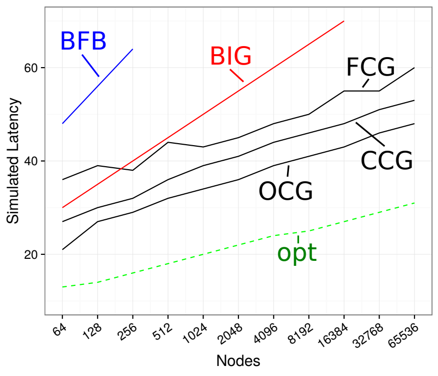
For our study, we assume a mean-time between failures of 18,304 hours as reported for compute servers of the TSUBAME 2.0 supercomputer at the Tokyo Institute of Technology [14]. Assuming that many scientific computing jobs run for 12 hours between checkpoints, we can compute the expected number of failures occuring during an application's run. For example, with 4,096 nodes, we expect ≈ 2.69 node failures during the application run.

D. Case study with N=4,096 nodes

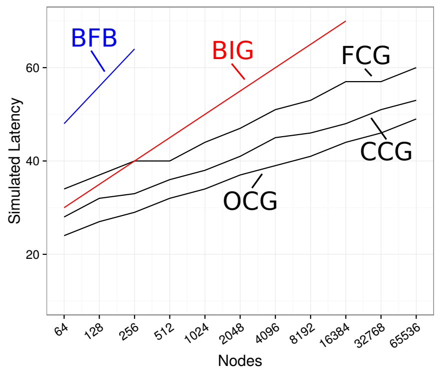
Table 7 shows a comparison of all the presented algorithms on a medium-sized system of 4,096 nodes. The results for GOS, OCG, CCG, and FCG are simulated and the results for BIG and BFB are modeled analytically as described above. We ran 10^6 simulations for each combination of parameters and report the expected latency, total work (number of messages) and inconsistency (expected share of nodes not reached). All reported values are mean values and all non-parametric confidence intervals were within 2% of the reported median.

For BFB, we assume that $\lceil 20\% \rceil$ of the failures happen during the operation. We analyze the algorithms for no failures ($\hat{f} = 0$) and the expected failure scenario ($\hat{f} = 3$). We run FCG with f equal to $\hat{f}$. The probability $\hat{p}$ that CCG is inconsistent, i.e., because nodes fail during the execution of the algorithm, is at most $\hat{p} = \frac{4{,}096 \cdot 55\mu s}{18{,}304 h \cdot 60^2 s/h \cdot 10^6 \mu s/s} = 3.4 \cdot 10^{-9}$ and thus, we never observed an inconsistent execution. We always choose $f = 1$ for FCG because the probability that two nodes fail during the execution of the algorithm is $\hat{p}^2 = 7.2 \cdot 10^{-19}$. BIG is consistent up to $\hat{f} = 9$ because it can accept up to $\log_2(4,096) - 1 = 11$ failures. As opposed to the corrected-gossip variants, BIG's static

algorithm	$\hat{f}$	T	lat	work	incon.
GOS [12]	0	50	53	95,418	2e-5%
GOS [12]	3	50	53	95,331	8e-6%
OCG	0	32	42	38,400	1e-4%
OCG	3	32	42	38,355	3e-4%
CCG	0	36	44	19,057	0%
CCG	3	34	46	16,952	0%
FCG	0	37	48	23,153	0%
FCG	3	37	51	23,101	0%
BIG [2]	0	-	60	49,152	0%
BIG [2]	3	-	60	49,152	0%
BFB [8]	0	-	96	4,096	0%
BFB [8]	3	-	144	8,192	0%



(a) Scaling for failure-free execution



(b) Scaling for an execution with failures

Figure 7: Latency [μs], work [# msgs] and consistency of reliable broadcast algorithms for $N = 4,096$ (in the table), $L = 2\mu s$, $O = 1\mu s$ for the expected failures in a 12-hour period.

overlay topology also suffers from nodes that failed before the algorithm started. Yet, the probability that 11 randomly chosen nodes disconnect the 4,096 node-graph is essentially zero ($< 10^{-40}$).

The results show that the corrected-gossip variants are superior across the board. OCG for example, delivers 99.999% consistency with 60% less messages (work) and 20% lower latency than GOS, while FCG uses over 50% less work and 15% lower latency to guarantee the same reliability as BIG. Further, while BFB requires the least amount of messages, FCG's latency is nearly 3 times less.

E. Scalability

We now investigate how well the different algorithms scale with the number of nodes in the system. For this, we ran simulations with increasing numbers of nodes and we show the result for a failure-free execution in Figure 7a. As before, we ran 10^6 simulations and all non-parametric confidence intervals were within 2% of the plotted median value. The lowest (fastest) dashed line marked "opt" indicates the theoretically-optimal (non-fault-tolerant) broadcast which provides a lower bound to the problem. We plot the median values for OCG, CCG, and FCG and omitted GOS for clarity because it performed significantly worse in all configurations. The red and blue lines plot the best-case times for BIG and BFB, respectively. For a weakly consistent configuration, OCG wins regardless of the number of nodes. A strongly consistent configuration using FCG outperforms BIG in terms of latency with 512 or more nodes and scales significantly better. We omitted work plots due to space restrictions but FCG consistently completes with less than 50% of the work of BIG.

Figure 7b shows the results for a scaling run with 1.5625% (N/64) node failures. Here we omit "opt" because it would not be consistent. In fact, it would not even be weakly consistent because a failure close to the top of the tree can leave nearly half the nodes unreached. All executions are strongly consistent, except OCG which reached at least 99.999% consistency in all cases.

BIG is consistent for $\hat{f} < \log_2 N$ but the expected number of failures for a 12 hour job grows as $\bar{f}(N) = \frac{12h \cdot N}{18304h}$. Thus, for $N > 22,001$, BIG may not be consistent on the TSUBAME 2.0 system (with low probability).

As in the failure-free case, OCG outperforms GOS across the board. FCG scales better and outperforms BIG when the number of nodes exceed 256. As for the work, it is very similar to the failure-free case. Our results demonstrate that corrected-gossip is better in terms of latency, number of messages and scalability.

V. Related Work

Fault-tolerant runtime design is an increasingly important topic. Birman et al. propose bimodal multicast, where an unreliable multicast is followed by gossip rounds to correct the lost messages. However, the correction step is not deterministic and transforms the "all or nothing" guarantee into "almost all or almost none" [6]. Similarly, Felber and Pedone propose a probabilistic reliable broadcast [13]. Hoefler, Siebert, and Rehm [19] show how to correct an unreliable multicast for full consistency without node failures.

Generally, deterministic constructions only work up to a fixed number of failures f. For example, binomial graphs [2] only tolerate up to $\log_2 P$ worst-case failures without healing. Self-healing variants [3] tolerate arbitrary failures assuming fast failure detectors which may not always be practical. Sun et al. [21] propose a reliable multicast protocol where k "loggers" guarantee deterministic delivery of all messages after a probabilistic gossip-based algorithm. In contrast, our algorithm combines gossip with deterministic correction to guarantee consistency despite failing nodes.

VI. Conclusions

We designed a series of practical algorithms for reliable broadcasting in unreliable environments. We utilize a combination of probabilistic gossip and deterministic correction to tune and implement various consistency levels. Our results show that the combination of randomness and determinism enables more scalable fault-tolerant protocols with up to 30% lower latency and 60% less messages than existing algorithms.

We described the model-driven tuning of all parameters of the algorithms (number of gossip and correction steps), which makes them readily usable for fault-tolerant communications, for example in MPI libraries or other parallel systems where latency and work is critical.

We believe that our idea to combine probabilistic communication with deterministic correction, should also be suitable for other communication operations such as MPI's collective communications, providing reliable group communication in unreliable environments.

VII. Acknowledgments

We thank Angelika Steger and Gustavo Alonso for many inspiring discussions and Darius Buntinas for sharing his implementation of BFB. We also thank the anonymous reviewers for their constructive comments. This research was supported by the DFG, German Priority Programme 1648 “Software for Exascale Computing" (SPPEXA), project FFMK.

References

[1] N. Alon, A. Barak, and U. Manber. On disseminating information reliably without broadcasting. In *7th Intl. Conf. on Distr. Comp. Sys. (ICDCS)*, pages 74–81, Berlin, 1987.

[2] T. Angskun, G. Bosilca, and J. Dongarra. Binomial graph: A scalable and fault-tolerant logical network topology. In I. Stojmenovic, R. Thulasiram, L. Yang, W. Jia, M. Guo, and R. de Mello, editors, *Parallel and Distributed Processing and Applications*, volume 4742 of *Lecture Notes in Computer Science*, pages 471–482. Springer, 2007.

[3] T. Angskun, G. Fagg, G. Bosilca, J. Pjesivac-Grbovic, and J. Dongarra. Self-healing network for scalable fault-tolerant runtime environments. *Future Generation Computer Systems*, 26(3):479 – 485, 2010.

[4] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition).* John Wiley Interscience, 2004.

[5] A. Barak and O. La'adan. The mosix multicomputer operating system for high performance cluster computing. *Future Gener. Comput. Syst.*, 13(4-5):361–372, 1998.

[6] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, 1999.

[7] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *Proc. 11th USENIX Conf. on Operating Systems Design and Implementation*, OSDI'14, pages 285–300, 2014.

[8] D. Buntinas. Scalable distributed consensus to support mpi fault tolerance. In *Proc. IEEE 26th Int'l Symp. Parallel & Distributed Processing (IPDPS)*, pages 1240–1249, 2012.

[9] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Logp: Towards a realistic model of parallel computation. In *Proc. 4th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, PPOPP '93, pages 1–12, 1993.

[10] A. H. Dekker and B. D. Colbert. Network robustness and graph topology. In *Proc. 27th Conf. on Computer Science (ACSC '04), Volume 26*, pages 359–368, 2004.

[11] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, PODC '87, pages 1–12, 1987.

[12] Z. Drezner and A. Barak. An asynchronous algorithm for scattering information between the active nodes of a multicomputer system. *J. Par. Distrib. Comp.*, 3(3):344–351, 1986.

[13] P. Felber and F. Pedone. Probabilistic atomic broadcast. In *Proc. 21st IEEE Symp. on Reliable Distributed Systems*, pages 170–179, 2002.

[14] Global Scientific Information and Computing Center. Failure History of TSUBAME2.0 and TSUBAME2.5, 2014.

[15] J. Gray and L. Lamport. Consensus on transaction commit. *ACM Trans. Database Syst.*, 31(1):133–160, 2006.

[16] V. Hadzilacos and S. Toueg. Distributed systems (2nd ed.). chapter Fault-tolerant Broadcasts and Related Problems, pages 97–145. 1993.

[17] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical report, Cornell Univ., Ithaca, NY, 1994.

[18] T. Herault, A. Bouteiller, G. Bosilca, M. Gamell, K. Teranishi, M. Parashar, and J. Dongarra. Practical scalable consensus for pseudo-synchronous distributed systems. In *Proc. Int'l Conf. for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 31:1–31:12, 2015.

[19] T. Hoefler, C. Siebert, and W. Rehm. A practically constant-time MPI Broadcast Algorithm for large-scale InfiniBand Clusters with Multicast. In *Proc. 21st IEEE Int'l Par. & Distr. Proc. Symp. (CAC'07 Workshop)*, page 232, 2007.

[20] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard Version 3.0, Sep. 2012.

[21] Q. Sun and D. C. Sturman. A gossip-based reliable multicast for large-scale high-throughput applications. In *Proc. 2000 Int'l Conf. on Dependable Systems and Networks (DSN '00)*, page 347, 2000.

[22] L. G. Valiant. A bridging model for parallel computation. *Comm. ACM*, 33(8):103–111, 1990.

[23] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

Appendix A.
Pseudo-code for all algorithms

The following listing shows the OCG algorithm. The function `rand(0..N-1\i)` selects a random node other than the caller.

Algorithm 1: Opportunistic Corrected-Gossip

```
1  $time = 0$; $i$ = my_node_id; $off=1$;
2  if $i$ != $root$ then
3      wait to receive ($time$, data);
4      $time$ += $L/O+1$;
   // gossip until time T and then wait L/O+1
5  while $time$++ $< T+L/O+1$ do
6      if $time < T$ then send ($time$, data) to rand($\{0..N-1\}\backslash i$);
   // correction - nodes colored here do not send
7  while $time < C$ do
8      if $time$++ $< C$ then send ($C$, data) to ($i + off$)%$N$;
9      if $time$++ $< C$ then send ($C$, data) to ($i - off$)%$N$;
10     $off$++;
```

The following listing shows the CCG algorithm.

Algorithm 2: Checked Corrected-Gossip

```
   // assume ▷ and ◁ are special time values > T
1  $time = 0$; $i$ = my_node_id; $off = 1$;
2  if $i$ != $root$ then
3      wait to receive ($time$, data);
       // only g-nodes go to gossip and correction
4      if $time = \triangleright \vee time = \triangleleft$ then exit;
5      $time$ += $L/O+1$
   // gossip until time T and then wait L/O+1
6  while $time$++ $< T+L/O+1$ do
7      if $time < T$ then send ($time$, data) to rand($\{0..N-1\}\backslash i$);
   // distance of first forward and backward msgs
8  $m_{\triangleright} = m_{\triangleleft} = \infty$;
   // send further forward or backward messages
9  $s_{\triangleright} = s_{\triangleleft} = true$;
10 while $(s_{\triangleright} \vee s_{\triangleleft})$ do
       // do for forward as well as for backward
11     for $dir \in \{\triangleright, \triangleleft\}$ do
12         while $src$ = $check\ for\ receive\ (id,\ data)$ do
13             if $t \in \{\triangleright, \triangleleft\} \wedge m_t > \delta_{\bar t}(src)$ then $m_t = \delta_{\bar t}(src)$;
14         if $off > m_{dir}$ then $s_{dir} = false$;
           // ▷ evaluates to 1 and ◁ to -1
15         if $s_{dir}$ then send ($dir$, data) to ($i + dir{\cdot}off$)%$N$;
           // (analysis assumes we wait O in else)
       // we looped around one full circle
16     if $off \geq N$ then exit;
17     $off$++;
```

For conciseness, we assume in both the CCG and FCG Algorithms that ▷ and ◁ are special time values bigger than T and that they evaluate to 1 and -1 in multiplications, respectively. In the FCG Algorithm which follows, we also assume that *false* is a special invalid node id which does not exist in the system and that any valid node id evaluates to *true* in conditions; and that the functions $\delta_{\triangleright}(x)$ and $\delta_{\triangleleft}(x)$ denote the distance between the calling node and x on the ring either forward or backward, respectively.

Algorithm 3: Failure-proof Corrected-Gossip

```
   // assume ▷, ◁, ★ are special time values > T
1  $time = 0$; $i$ = my_node_id;
   // arrays of size $f+1$ of known g-nodes
2  $k_{\triangleleft} = k_{\triangleright}$ = k = [$false$,..,$false$];
3  if $i$ != $root$ then
4      wait to receive ($time$, [], data);
5      if $time$ != ▷ $\wedge$ $time$ != ◁ $\wedge$ $time$ != ★ then $time$ +=
         $L/O+1$;
   // gossip until time T and then wait L/O+1
6  while $time$++ $< T+L/O+1$ do
7      if $time < T$ then send ($time$, [], data) to
         rand($\{0..N-1\}\backslash i$);
   // SOS handling (both g- and c-nodes)
8  if $time$ = ★ then
9      for $i \in \{0..N-1\}\backslash i$ do  send (★, [], data) to $i$ ;
10     exit;
   // c-nodes await completion (SOS after timeout)
11 if $time = \triangleright \vee time = \triangleleft$ then
12     while $src$ = $check\ for\ receive\ (id,\ k,\ data)$ do
13         if $|k_{\triangleright} = src \cup k \cup k_{\triangleright}| \geq f+1$ then exit;
14         if $time$ = ★ $or\ timeout\ expired$ then goto line 9;
   // g-nodes enter correction step
15 $off_{\triangleright} = 1$; $off_{\triangleleft} = 1$; $done = 0$;
   // sending ▷ or ◁ messages
16 $s_{\triangleright} = true$; $s_{\triangleleft} = true$;
   // sending final-round ▷ or ◁ messages
17 $f_{\triangleright} = false$; $f_{\triangleleft} = false$;
18 while $(s_{\triangleright} \vee s_{\triangleleft})$ do
19     for $dir \in \{\triangleright, \triangleleft\}$ do
20         while $src$ = $check\ for\ receive\ (t,\ k,\ data)$ do
21             if $t \in \{\triangleright, \triangleleft\} \wedge\ !f_t$ then
22                 $k_{\bar t} = ⊿_t(src \cup k \cup k_{\bar t})[0..f]$
23             if $t$ = ★ then goto line 9;
           // f+1 g-nodes found, start final phase
24         if $|k_{dir}| \geq f \wedge\ !f_{dir}$ then $f_{dir}$ = true; off$_{dir}$=1;
             $s_{dir}$=true;
           // stop if passed f-th g-node in final
25         if $off_{dir} > \delta_{dir}(⊿_{dir}(k_{dir})[f])$ then $s_{dir}$ = false;
26         if $s_{dir} \wedge off_{dir} \leq N$ then
               // ▷ evaluates to 1 and ◁ to -1
27             send ($dir$, $k_{dir}$, data) to ($i + dir{\cdot}off_{dir}$)%$N$
                 $off_{dir}$++;
       // full loop without f+1 g-nodes: SOS
28     if $off_{\triangleright} > N \vee off_{\triangleleft} > N$ then goto line 9;
```

Appendix B.
Selecting the number of gossip steps of FCG

As in OCG and CCG, we select T to minimize the latency. Each g-node in FCG must fulfill the following tasks:

1) Learn about $f+1$ g-nodes in $\rhd$ direction.
2) Learn about $f+1$ g-nodes in $\lhd$ direction.
3) Send final messages up to the $f+1^{st}$ $\rhd$ g-node.
4) Send final messages up to the $f+1^{st}$ $\lhd$ g-node.

We now provide the intuition for a worst-case analysis of the runtime of FCG.

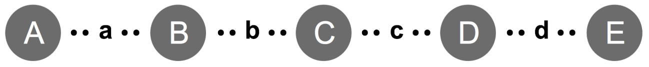


Figure 8: Description for FCG analysis

Figure 8 shows how each g-node can be seen as the center node C between four other g-nodes A, B, D, and E (for $f = 1$). The values a-d are integers that represent the distances between neighboring g-nodes, which is the number of c-nodes between them plus one. Next, we derive an upper bound on the total number of nodes in the A-E chain.

Bounding the length of a chain of G g-nodes: Let $V \geq 2$ be an integer. For any $G \geq V$, consider a specific node i out of all N nodes. The probability that i starts a sequence of G consecutive nodes containing exactly V g-nodes is:

$$q(G, V) = \frac{c(T + L + O)^V (N - c(T + L + O))^{G-V} (G - 2)!}{N^G (V - 2)!(G - V)!}.$$

Since there are N possible sequences of V consecutive g-nodes, assuming independence, the probability that such a sequence exists is: $\beta_G = 1-(1 - q(G, V))^N$. The probability q_G that the length of the maximal sequence of V consecutive g-nodes is G, is equal to the probability that such a sequence of length G exists and longer sequences do not exist: $q_G = \beta_G(1 - \beta_{G+1})(1 - \beta_{G+2}) \dots$. Let $\overline{G}_V$ be the smallest value of G for which $\sum_{i=G+1}^{N} q_i < \delta$. Note that $\overline{G}_V$ is a function of N, n, T, L, δ, and V.

Bounding the runtime of FCG: We assume Wlog., that all nodes start the correction phase at time zero; that the alternating order of message-sending starts to the left (i.e., $B \leftarrow C$) then to the right (i.e., $C \rightarrow D$); and that $f = 1$ and therefore we will be looking at $V = 5$. To fulfill task 1, C must receive notice from A. The worst-case for this is when A is reset to send final messages by learning about $f+1 = 2$ other g-nodes right before it would send to C. The time for C to learn about A is thus bounded by $T_1 \leq (4(a+b)-3)O + L + 2O \leq 4(\overline{G}_V - 3)O - 3O + L + 2O = 4\overline{G}_V O + L - 13O$. The argument for E in the forward direction is similar but starts one O earlier: $T_2 \leq 4\overline{G}_V O + L - 14O$. To achieve task 3, D will reach C after at most $(4c - 4)O + (L + 2O)$ and C will send final messages towards A at least once every two cycles, which will take at most $2(a + b)O$. The total time $T_3 = [(4c - 4)O + L + 2O] + [2(a + b)O] = 2(a + b + c)O + 2cO - 4O + L + 2O \leq 2(\overline{G}_V - 2)O + 2(\overline{G}_V - 4)O + L - 2O = 4\overline{G}_V O + L - 14O$. Task 4 is similar to task 3 except that B starts backward messages one cycle later, thus, the worst case is $T_4 = 4\overline{G}_V O + L - 13O$.

The worst-case time is thus $T_{1-4} = 4\overline{G}_V O + L - 13O$ and the approximated best T to complete FCG for $f = 1$ is

$$T_{opt}^{FCG} = \underset{T}{\operatorname{argmin}}(T + 4\overline{G}_V O + L - 13O) \ . \qquad (5)$$

While several values of T can produce the minimum, we recommend to select the T that can withstand the largest reduction in δ.

Equation (5) is obviously a theoretical upper-bound. In practice, it is very rare for FCG to take as long. To analyze the accuracy of this upper bound, we simulated 10^6 runs of FCG with $L = O = 1$ on $N = n = 1,024$ nodes for various values of T and measured the number of steps to complete the correction phase. We ran this experiment 10 times and in Figure 9 we plot (in dots) the maximum of all observed latencies for each T. The figure also shows (in solid line) the upper bound for the total time, resulting from Equation (5). As before, we used $\delta = 6.93 \cdot 10^{-7}$ for our approximations. While the upper-bound approximation is less accurate than in CCG and OCG, it can be computed quickly and can be used to find a good T.

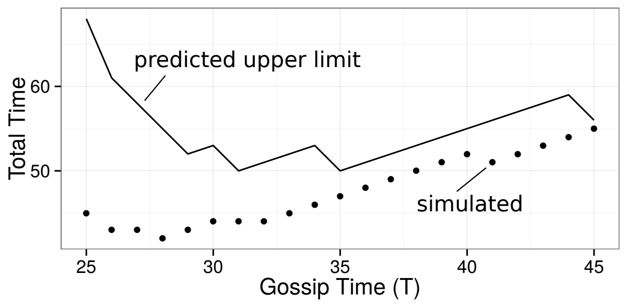


Figure 9: Failure-proof Corrected-Gossip evaluation