Efficient Update Diffusion in Byzantine Environments

Dahlia Malkhi
School of Computer Science and Engineering
The Hebrew University of Jerusalem, Israel
dalia@cs.huji.ac.il

Michael K. Reiter
Bell Labs, Lucent Technologies
reiter@research.bell-labs.com

Ohad Rodeh
School of Computer Science and Engineering
The Hebrew University of Jerusalem, Israel
orodeh@cs.huji.ac.il

Yaron Sella
School of Computer Science and Engineering
The Hebrew University of Jerusalem, Israel
ysella@cs.huji.ac.il

Abstract

We present a protocol for diffusion of updates among replicas in a distributed system where up to b replicas may suffer Byzantine failures. Our algorithm ensures that no correct replica accepts spurious updates introduced by faulty replicas, by requiring that a replica accepts an update only after receiving it from at least $b+1$ distinct replicas (or directly from the update source). Our algorithm diffuses updates more efficiently than previous such algorithms and, by exploiting additional information available in some practical settings, sometimes more efficiently than known lower bounds predict.

Keywords: FT Algorithms, Security, FT Communication.

1. Introduction

Update diffusion is a process by which an update that is initially known to a portion of a distributed system is eventually made known to the rest of the system. Diffusion is generally distinguished from similar problems like reliable broadcast in that it is a periodic (round-based) process that occurs asynchronously to the introduction of updates to the system. As such, it tends to be a background process that should incur minimal overhead.

Here we study diffusion in a setting in which the components of the system, here called *replicas*, may suffer Byzantine failures. The foremost requirement that we place on our protocol is that correct replicas accept no spurious updates introduced by faulty replicas, but rather accept only updates introduced from a trusted update source. Assuming that there is a known upper bound b on the number of faulty replicas, and that each legitimate update is introduced at $\alpha > b$ correct replicas, this can be achieved if each replica accepts only updates received from $b+1$ distinct replicas (or from the update source itself). To our knowledge, the only prior work that has investigated this scenario is [MMR99]. That work introduced two measures of a diffusion method: *delay* is the expected number of rounds from when an update is introduced and until it is accepted by all replicas, and *fan-in* is the expected maximum number of messages that need to be handled by any replica in any round. That work proved a lower bound of $\Omega(nb/\alpha)$ on the product of delay and fan-in for a system of n replicas, and proposed two algorithms for diffusion.

In this paper we describe a diffusion algorithm for this setting that scales better than the algorithms of [MMR99] and that, in certain practical settings, exploits additional information about the system to perform better than the $\Omega(nb/\alpha)$ lower bound would predict. Like one of the algorithms of [MMR99], here called "MMR Tree", our algorithm structures diffusion along a tree of logical nodes, each containing some number $\ell \geq 2b+1$ replicas. In such tree-based diffusion, delay of an update is strongly influenced by the rounds required before $b+1$ correct replicas of some node become active for the update. MMR Tree makes this happen quickly by the replicas in the root of the tree more often being the target for propagation than others. However, this results in considerably greater fan-in on these replicas, and the product of delay and fan-in suffers accordingly.

Here we explore two alternatives to cause sufficiently many correct replicas of some tree node to quickly become active for each update. In contrast to MMR Tree, however, our approaches yield fan-in very close to one. In the first alternative, we observe that in some systems, the possible sets of replicas to which updates are introduced are few and highly predictable. In this case, we simply align these sets with the tree nodes, so that the update introduction itself activates sufficiently many correct replicas in a node for that update. The delay and fan-in of this algorithm are inherited directly from our tree construction, which is very efficient (in particular, yielding a fan-in of one).

In the second approach, each replica intersperses its rounds of tree propagation with rounds in which it uses the *contents* of an update to select a node to which it will propagate for ℓ (interspersed) rounds. In this way, once every replica at which an update is introduced has used that update for selecting a node, sufficiently many correct replicas of that node will become active for that update within a fixed number of rounds. By utilizing a pseudorandom function for selecting a node as a function of the update, no one node is targeted significantly more than others, and so the fan-in remains very close to one. Furthermore, while updates await their diffusion to the selected node, they are piggybacked on other propagation messages to different nodes, and hence can simultaneously diffuse from multiple nodes in the tree. Thus, unless the arrival rate is prohibitively high, each update quickly becomes active in one or more nodes.

The rest of this paper is structured as follows. Section 1.1 relates our work to previous work. We present our system model and necessary definitions in Section 2. We describe our method of tree diffusion in Section 3. We then address the issue of activating sufficiently many replicas in a single node in Section 4. We conclude the paper with results of simulations of our algorithm in Section 5.

1.1. Related Work

Diffusion is a fundamental mechanism for driving replicated data to a consistent state in a highly decentralized system. Our work optimizes diffusion protocols in systems where arbitrary failures are a concern, and may form a basis of solutions for disseminating critical information in any survivable system. Specifically, application of our Byzantine diffusion methods can be found in the Fleet system [MR00], which provides a persistent object repository that survives the corruption of a threshold of the servers comprising it using Byzantine quorum replication techniques [MR98, MRW00]. A key property of Fleet in relation to the diffusion method introduced here is that each update is initially introduced in Fleet to a subset of servers exceeding the presumed resilience threshold b of the system. Other, similar applications might be found in any secure storage system which is highly decentralized, e.g., Intermemory [GY98], and various implementations of the Eternity service [And96, WRC00].

As mentioned above, the study of Byzantine diffusion was initiated in [MMR99]. Prior to that paper, previous work on update diffusion focused on systems that can suffer benign failures only. Notably, Demers et al. [DGH+87] performed a detailed study of epidemic algorithms for the benign setting, in which each update is initially known at a single replica and must be diffused to all replicas with minimal traffic overhead. One of the algorithms they studied, called *anti-entropy* and apparently initially proposed in [BLNS82], was adopted in Xerox's Clearinghouse project (see [DGH+87]) and the Ensemble system [BHO+99]. Similar ideas also underly IP-Multicast [Dee89] and MUSE (for USENET News propagation) [LOM94]. The main difference between diffusing updates in a system that exhibits benign failures only as opposed to a Byzantine environment is that in the latter case, an update received from b or fewer replicas cannot be trusted. Therefore, techniques by which more than b replicas corroborate each update must be employed.

Prior studies of update diffusion in distributed systems that can suffer Byzantine failures have focused on single-source broadcast protocols that provide reliable communication to replicas and replica agreement on the broadcast value (e.g., [LSP82, DS83, BT85, MR97]), sometimes with additional ordering guarantees on the delivery of updates from different sources (e.g., [Rei94, CASD95, MM95] and [KMM98, CL99]). The problem that we consider here is different from these works in the following ways. First, in these prior works, it is assumed that one replica begins with each update, and that this replica may be faulty—in which case the correct replicas can agree on an arbitrary update. In contrast, in our scenario we assume that at least a threshold $\alpha > 1$ of *correct* replicas begin with each update, and that only these updates (and no arbitrary ones) can be accepted by correct replicas. Second, these prior works focus on reliability, i.e., guaranteeing that all correct replicas (or all correct replicas in some agreed-upon subset of replicas) receive the update. Our protocols diffuse each update to all correct replicas only with some probability that is determined by the number of rounds for which the update is propagated before it is discarded. Our goal is to devise diffusion algorithms that are efficient in the number of rounds until the update is expected to be diffused globally and the load imposed on each replica as measured by the number of messages it receives in each round.

2. Preliminaries

Following the system model of [MMR99], our system consists of a universe S of n *replicas* to which updates are introduced over time. Up to some known threshold b of the replicas could be *faulty*; a faulty replica can deviate from its specification arbitrarily (Byzantine failures). Replicas that always satisfy their specifications are *correct*. We do not allow the use of digital signatures by replicas, and hence, our model is the full Byzantine model (versus the stronger "Byzantine with authentication" model, c.f., [CASD95]).

Replicas can communicate via a completely connected point-to-point network. Communication channels between correct replicas are reliable and authenticated, in the sense that a correct replica p_i receives a message on the communication channel from another correct replica p_j if and only if p_j sent that message to p_i.

Our work is concerned with the diffusion of *updates* among the replicas. Each update u is introduced to an *initial set* I_u of at least $\alpha \geq b+1$ correct replicas (i.e., $|I_u| \geq 2b+1$), and is then diffused to other replicas via message passing. The goal of a diffusion algorithm is to make all correct replicas *active* for u, where a replica p is active for u if $p \in I_u$ or if p has received u from $b+1$ distinct replicas. In order to prevent the diffusion of updates introduced spuriously by faulty replicas, a replica p does not send u to another replica until p is active for u.

Our diffusion protocols proceed in synchronous rounds. For simplicity, we assume that each update arrives at each replica in I_u simultaneously, i.e., in the same round at each replica in I_u. In each round, each replica selects one other replica to which it sends all updates for which it is active. A replica receives and processes all messages sent to it in a round, before the next round starts. We consider the following two measures of quality for diffusion protocols:

Delay: For each update, the delay is the worst-case expected number of rounds from the time the update is introduced to the system until all correct replicas accept the update. Formally, let η_u be the round number in which update u is introduced to the system, and let τ_p^u be the round in which a correct replica p accepts update u. The delay is $E[\max_{p,C}\{\tau_p^u\} - \eta_u]$, where the expectation is over the random choices of the algorithm and the maximization is over correct replicas p, all failure configurations C containing no more than b failures, and all behaviors of those faulty replicas. In particular, $\max_{p,C}\{\tau_p^u\}$ is reached when the faulty replicas send no updates, and so this is the behavior we assume from them when analyzing delay.

Fan-in: The fan-in measure, denoted by F^{in}, is the expected maximum number of messages that any correct replica receives in a single round from correct replicas under all possible failure scenarios. Formally, let ρ_p^i be the number of messages received in round i by replica p from correct replicas. Then the fan-in in round i is $E[\max_{p,C}\{\rho_p^i\}]$, where the maximum is taken with respect to all correct replicas p and all failure configurations C containing no more than b failures. *Amortized fan-in* is the expected maximum number of messages received over multiple rounds, normalized by the number of rounds. Formally, a k-amortized fan-in starting at round l is $E[\max_{p,C}\{\sum_{i=l}^{l+k} \rho_p^i/k\}]$. We emphasize that fan-in and amortized fan-in are measures only for messages from correct replicas.

The following bound presents an inherent tradeoff between delay and fan-in, for the case that the initial set I_u is arbitrarily designated (i.e., **any** choice of α replicas is equiprobable), independently from the diffusion algorithm A then:

Theorem 2.1 ([MMR99]) *Denote by D the algorithm's delay, and by F^{in} its D-amortized fan-in. Then $DF^{in} = \Omega(bn/\alpha)$, for $b \geq 2\log n$.*

The implication of Theorem 2.1 is that the delay can be decreased by increasing the fan-in. Unfortunately, in some cases this might be unacceptable. Theorem 2.1 introduces a new measure for the quality of a diffusion algorithm, namely DF^{in}, and raises the question: can the prediction of Theorem 2.1 regarding this metric be somehow circumvented? One contribution of the present work is an affirmative answer to this question. This is achieved by exploiting additional information often available in practical systems, as an integral part of the diffusion algorithm.

3. Tree propagation

In this section we introduce a tree-based propagation algorithm. While this algorithm does not guarantee finite delay, we extend it in Section 4 to provide finite delay. It thus serves as the basis for the propagation algorithms that constitute the main contribution of this paper.

In this algorithm, the universe S of replicas is partitioned into nonintersecting *nodes* $N_0, \ldots, N_{n/\ell}$, each containing $\ell \geq 2b+1$ replicas. (We assume $\ell | n$ for simplicity of presentation.) These nodes are arranged in a balanced d-ary tree. Let $\mathsf{parent}(i)$, $\mathsf{child}(i,1), \ldots, \mathsf{child}(i,d)$ denote the parent and d children of N_i, respectively. One or more of these can be undefined ($\perp$) for nodes missing a parent and/or children.

In addition, to each node N_i is associated a static *propagation schedule* σ_i, which is some (0-indexed) permutation of $\langle \mathsf{parent}(i), \mathsf{child}(i,1), \ldots, \mathsf{child}(i,d)\rangle$. Propagation schedules satisfy the constraint that for each $i, i' \in \{0, \ldots, n/\ell\}$ and $j \in \{0, 1, \ldots, d\}$, if $\sigma_i(j) \neq \perp$ and $\sigma_i(j) = \sigma_{i'}(j)$, then $i = i'$. That is, for no two distinct nodes $N_i, N_{i'}$ are $\sigma_i(j)$ and $\sigma_{i'}(j)$ the same (and defined). For example, to satisfy this constraint it suffices to choose schedules such that $\sigma_i(j) = N_{i'} \Leftrightarrow \sigma_{i'}(j) = N_i$. Intuitively, $\sigma_{i'}(j)$ determines the node to which the replicas in N_i propagate during rounds specified by j.

More precisely, propagation takes place in epochs of $2b+1$ consecutive rounds each. During epoch e, the replicas in node N_i send updates to the replicas of node $\sigma_i(e \bmod (d+1))$, so that each replica in $\sigma_i(e \bmod (d+1))$ is targeted by

$2b + 1$ replicas from N_i during the epoch. Furthermore, at every round, each replica receives updates from exactly one other replica. More specifically, suppose the replicas in node N_i are labeled $p_{i,0}, \ldots, p_{i,\ell-1}$. Then, these constraints can be satisfied if during round r, $0 \leq r < 2b + 1$, of epoch e, replica $p_{i,j}$ propagates to $p_{i',j'}$ where $N_{i'} = \sigma_i(e \bmod (d+1))$ and $j' = (j + r) \bmod \ell$. (This schedule could be trivially optimized to ensure that in each round, every replica sends to and receives from the same other replica).

Several properties of this algorithm are important for its use in Section 4. First, by construction, it maintains a fan-in of one: the same replica is never targeted by two different replicas in the same round. Second, suppose we define a node to be *active* for update u if all the correct replicas in that node are active for u. Then, we have the following result:

Proposition 3.1 *If a node is active for u by round r, then all correct replicas are active for u by round $r + 2(2b + 1)(d + 1)\log_d(n/\ell)$.*

Proof: Once a node N_i is active for u, all correct replicas in each of its neighboring nodes becomes active for u within $(d + 1)(2b + 1)$ rounds, since each replica in a neighboring node is the propagation target of $b+1$ correct, active replicas in N_i during those $(d+1)(2b+1)$ rounds. Since in the worst case, N_i is a leaf node, u must propagate all the way to the root of the tree and back down to the other leaves, i.e., twice the depth of the tree, or $2\log_d(n/\ell)$. As a result, the total number of rounds before u is active at all correct replicas is at most $(d + 1)(2b + 1) \cdot 2\log_d(n/\ell) = 2(2b + 1)(d + 1)\log_d(n/\ell)$. □

We note that despite Proposition 3.1, the delay of this algorithm is not finite: it is possible that no node ever becomes active for u. We thus extend this algorithm in Section 4 to ensure that some node eventually does become active for u.

4. Activating a tree node

Our main goal in this section is to extend the algorithm of Section 3 to ensure finite delay—specifically, forcing some node to become active for each update u—without increasing the fan-in of the algorithm significantly. There are any number of ways to accomplish this goal, such as by interspersing rounds of the algorithm in Section 3 with rounds of a known, finite-delay propagation algorithm (e.g., [MMR99]). However, this would provide little advantage over using these known, finite-delay propagation algorithms only.

Rather, the strategies we investigate here to ensure that some node becomes active for each update are derived from common practical scenarios. By exploiting additional information found in many practical settings, we obtain finite-delay diffusion algorithms that perform well in many common cases, and even better than the lower bounds of [MMR99] would predict. In one case, the additional information that we exploit are limitations on the collection of initial sets I_u to which updates u can be introduced. In the second case, we presume a low arrival rate of updates and develop an optimized propagation protocol for this case.

4.1. Limited initial sets

Consider a system in which updates are known to be introduced at one of only relatively few possible initial sets; specifically, I_u is drawn from a collection of sets $\{Q_1, \ldots, Q_m\}$ where $m \ll \binom{n}{\alpha}$. This additional information may permit an utterly trivial solution to the problem of ensuring that some node becomes active for each update: if the tree nodes can be arranged so that each Q_i *contains* some tree node, then the initial update to I_u will automatically make a node active for u. The delay of the resulting propagation algorithm is then at most $2(2b+1)(d+1)\log_d(n/\ell)$ as predicted in Proposition 3.1, and the fan-in remains equal to one.

More generally, if each Q_i intersects some tree node in $2b+1$ replicas (and thus, in at least $b+1$ correct ones), then the initial update to I_u will automatically activate $b+1$ correct replicas in some tree node. In this case, we employ an algorithm P_i for a node N_i containing $b + 1$ correct active replicas to activate the remaining replicas in N_i. The algorithm P_i can simply consist of ℓ rounds of propagation, in which the replicas of node N_i target each other in a round robin fashion. It suffices then for P_i to eventually execute at each node N_i for the tree diffusion protocol of Section 3 to terminate. To this end, we interleave the execution of the P_i algorithm (of ℓ rounds) with the execution of the rounds of the tree protocol. We call rounds of the former "self-rounds" and rounds of the latter "tree-rounds". During self-rounds, replicas of node N_i execute algorithm P_i, and during tree-rounds, replicas execute the tree protocol. If the fraction of self-rounds is R, and so the fraction of tree-rounds is $1 - R$, then the delay until u becomes active at all correct replicas is at most $\frac{1}{R}\ell + \frac{1}{1-R}2(2b + 1)(d + 1)\log_d(n/\ell)$.

One scenario in which initial sets are limited in this way is when updates are introduced at a *quorum* of replicas. An example of such a system is Fleet [MR00], a distributed object storage infrastructure that uses replication to mask Byzantine replica faults. Due to its anticipation of Byzantine replica faults, each update is introduced at a quorum of replicas as defined by a *masking quorum system* [MR98], i.e., any two quorums intersect in at least $2b + 1$ replicas. Moreover, propagation is employed in Fleet to diffuse up-

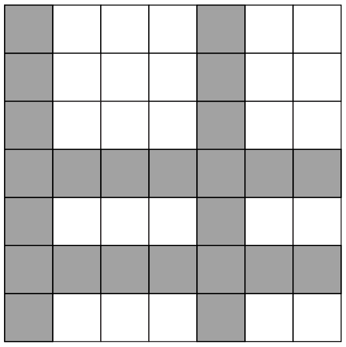

Figure 1. The M-grid construction [MRW00], $n = 7 \times 7, b = 3$, with one quorum shaded.

dates from the original quorum where they are introduced to the rest of the replicas [MMR99]. Several masking quorum systems are known that can be exploited in Fleet and that suffice for our new propagation method.

One example is the M-Grid construction of [MRW00]. In this construction, replicas are arranged in a logical $\sqrt{n} \times \sqrt{n}$ grid. A quorum consists of any choice of $\sqrt{b+1}$ rows and $\sqrt{b+1}$ columns (e.g., see Figure 1), which ensures the requisite $(2b+1)$-intersection between any two quorums provided that $b < \sqrt{n}/2$. In this case, choosing tree nodes so that each row of the grid contains at least one tree node suffices to yield the desired propagation algorithm. Since each quorum contains some tree node, the performance is simply that predicted by the tree protocol.

Another, more complex example, is the Recursive-Threshold (RT) construction of [MRW00]. In this construction, the replicas are arranged as the leafs of an m-ary selection tree, where each leaf contains m replicas. Quorums are constructed as recursive choices of k-out-of-m children of nodes, starting from the root up to the leaves, and selecting k-out-of-m replicas from each selected leaf (see Figure 2). Denoting the height of the selection-tree by h, this construction ensures that every pair of quorums intersects in $(2k-m)^h$ replicas, and hence this construction tolerates $b = ((2k-m)^h - 1)/2$ Byzantine failures. For the RT system, we choose our propagation tree nodes to be sub-trees of the RT selection-tree, of height ℓo, such that ℓo is the minimal height that satisfies $k^{\ell o} \geq 2b+1$. In this way, each RT quorum intersects some tree node in $2b+1$ replicas, and the requirements of our protocol are satisfied. The total delay predicted for this method is $O(\frac{1}{R}m^{\ell o} + \frac{1}{1-R}2(2b+1)(d+1)\log_d \frac{n}{m^{\ell o}})$. Since $m^{\ell o} \leq O(b^2)$, we have that the delay is $O(b(b + \log \frac{n}{b}))$.

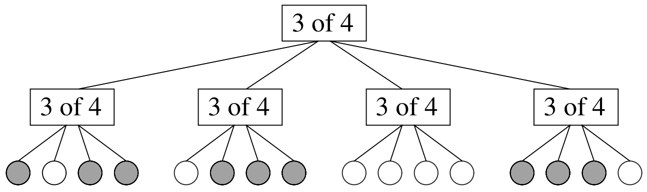

Figure 2. An $\mathrm{RT}(4,3)$ system of depth $h = 2$, with one quorum shaded.

4.2. Low update arrival rate

In this subsection we describe a second approach to ensuring that some node eventually becomes active for each update u. While this approach always ensures that some node becomes active for u, it offers an advantage in delay over other approaches primarily when the arrival rate of updates is low. Unlike the approach of Section 4.1, it works equally well regardless of the initial sets to which updates may be introduced, however, it does not improve the asymptotic behavior of the DF^{in} bound.

This approach employs a function $\mathsf{u2n}$ that maps each update u to some node N_i of the tree. Consider the protocol $\Pi_{p,u}$ for a replica $p \in I_u$ consisting of the following steps: p computes $N_i = \mathsf{u2n}(u)$, chooses an integer $j \in \{1, \ldots, \ell\}$ uniformly at random, and then performs ℓ rounds of propagation in which it propagates to replica $p_{i,(j+r) \bmod \ell}$ in round r, $r = 0, \ldots, \ell - 1$. The approach of this section requires each $p \in I_u$ to eventually execute $\Pi_{p,u}$ for each update u such that $p \in I_u$. As a result, node $\mathsf{u2n}(u)$ will eventually become active for u.

We integrate execution of $\Pi_{p,u}$ rounds with rounds of the tree protocol as follows; we refer to the former as "Π-rounds", and to the latter as "tree-rounds". Each p keeps a queue of updates introduced to it directly ("direct updates"). During tree-rounds, p executes the tree protocol, treating direct updates as any other update for which it is active. During Π-rounds, it executes rounds of protocol $\Pi_{p,u}$ for the update u at the head of its queue (if there is one). Once $\Pi_{p,u}$ is complete—i.e., after ℓ Π-rounds with u at the head of the queue—it dequeues u from the queue and henceforth treats u like any other update for which it is active.

The benefit of this algorithm under low update arrival rates is easy to illustrate in the extreme case of only one update u being introduced into the system in isolation. In this case, a tree node becomes active for u within ℓ Π-rounds, i.e., the ℓ rounds of $\Pi_{p,u}$ for each $p \in I_u$. Then, by Proposition 3.1, all correct replicas will be active for u in $2(2b+1)(d+1)\log_d(n/\ell)$ tree-rounds. If the fraction of Π-rounds is R, and so the fraction of tree-rounds is $1 - R$, then the rounds until u becomes active at all correct replicas is at most

$$\frac{1}{R}\ell + \frac{1}{1-R}(2(2b+1)(d+1)\log_d(n/\ell))$$

R can be tuned to minimize this delay for a given choice of ℓ. Unfortunately, however, the effects of this tuning can be lost as the load on the system grows, as we show in Section 5.

The fan-in of this approach is no longer identically one, since multiple replicas may target a single replica in the same round (specifically, in Π-rounds). However, an amortized fan-in very close to one can be achieved if u2n is a pseudorandom function, as will be shown in Section 5.

We recall the following lower bound known about the delay of *any* propagation method:

Theorem 4.1 ([MMR99]) *The delay of any diffusion algorithm A is $\Omega(b \log \frac{n}{\alpha})$.*

When $\alpha \approx b$, choosing $\ell = 2b+1$ for our method yields asymptotically optimal delay under low load.

5. Simulation results

In order to shed light on the performance of our protocols in practice, in this section we present the results of simulations of our protocols in varying system sizes and under various rates of update arrival. We consider a system in which update arrival is a Poisson process with a rate of λ updates per round. The parameters we vary in our exploration are the number n of replicas and the arrival rate λ. We measure two sets of experiments: In one, the initial sets to which updates are introduced are random choices of a quorum in the M-Grid quorum system. In the other, the initial sets are random selections of $b+1$ replicas. We fix the maximum assumed number b of replica failures in all of our simulations to either $b = 2$ or $b = 5$, and the node size $\ell = 2b+1$. It is important to note that our simulation did not model the actual failures of replicas. Rather, it simulates a system which could tolerate up to b failures, should such failures occur.

Our simulations focus on the MMR Tree algorithm as our point of comparison. Our simulation of MMR Tree employs the same node size as our method. As discussed in Section 1, MMR Tree quickly activates $b+1$ correct replicas in the root node for each update by making replicas in the root more likely to be selected as any replica's target of propagation in each round. Specifically, as the MMR Tree algorithm is described in [MMR99] and simulated here, each replica targets some replica in the root node in 25% of the rounds on average. To draw as direct a comparison as possible with our algorithms, when simulating our algorithm of Section 4.2, we designate 25% of rounds as Π-rounds, and the other 75% of rounds as tree-rounds. For the algorithm of Section 4.1, the initial set included all the replicas of a specific tree-node, hence there was no need to interleave self-rounds with tree-rounds. The MMR Tree algorithm also uses *update batching*, that is, whenever a replica needs to send an update to another replica, it scans its queue for other updates intended for the same replica, and sends them all in a single batch (i.e., message). Again, to keep the comparison on similar grounds, we implemented an identical optimization in our two new algorithms.

The results of our simulations are shown in Figures 3–8. Figures 3 and 4 illustrate the delay (D) and the delay times fan-in (DF^{in}) of the algorithm of Section 4.1 when $b = 2$ and updates are introduced to a quorum in the M-Grid quorum system of [MRW00] (and tree nodes are arranged so that each quorum includes at least one full tree node). Figure 6 illustrates D and DF^{in} for the algorithm in Section 4.2 when $\lambda = 5$, $b = 2$ and updates are introduced to $\alpha = b+1 = 3$ randomly selected replicas. Figures 6–8 illustrate the same measures as Figures 3–5 for the case $b = 5$. Each plotted point is the result of averaging the measure (D or DF^{in}) for the first 2000 updates to be accepted at all replicas.

The primary point we wish to convey with these graphs is that our protocols vastly outperform MMR Tree in DF^{in}, in all cases studied, as shown in the right side of each figure. Though our protocols in some cases have inferior delay, their improvement in DF^{in} result from the fact that our protocols hold F^{in} at or very close to one. In contrast, due to the increased load suffered by replicas in the root node in MMR Tree, F^{in} for that algorithm grows as a function of the system size; e.g., see Figures 3, 6, 6 and 8. Even for a constant system size of $n \approx 100$, the amortized fan-in is already significant and renders MMR Tree substantially inferior to our protocols in this measure (see Figures 4 and 7).

6. Conclusion

In this paper we have explored a tree-based propagation method for Byzantine environments, and utilized additional information available in certain practical settings to achieve propagation with an asymptotically lower product of delay and fan-in than the lower bound of [MMR99] would predict. In one case, we exploit additional information about the sets of replicas to which an update can be initially introduced. In the second, we exploit an assumed low arrival rate of updates. In both cases, we are able to achieve amortized F^{in} very close to 1, and delay that is asymptotically logarithmic in the system size n. We also presented simulations confirming that our algorithms offer a better product of delay and fan-in than previous algorithms.

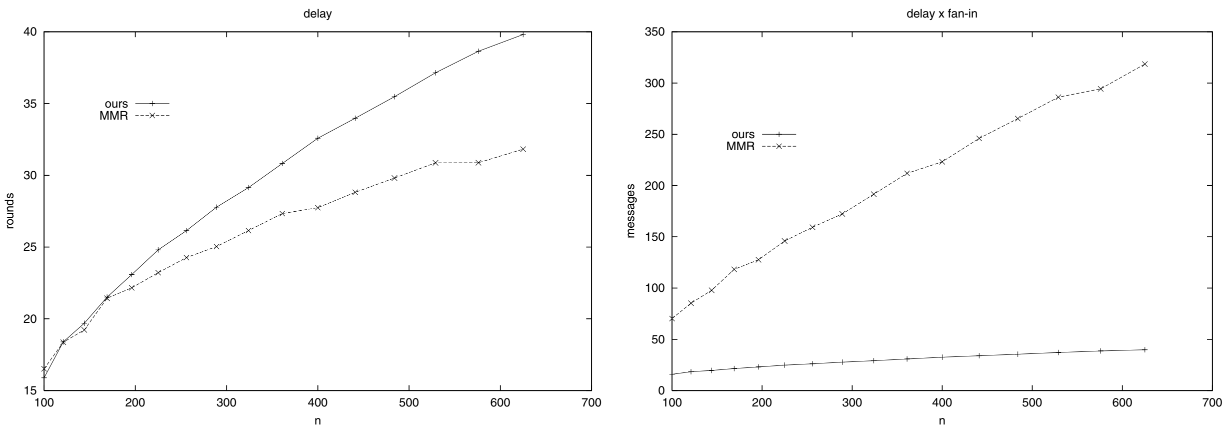

Figure 3. $\lambda = 5$, $b = 2$, **updates introduced to M-Grid quorums (Section 4.1)**

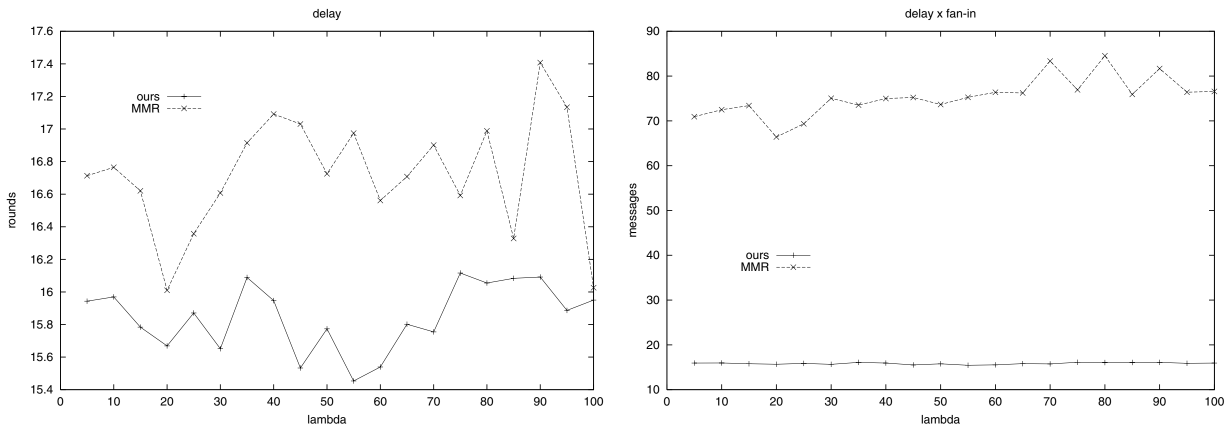

Figure 4. $n = 100$, $b = 2$, **updates introduced to M-Grid quorums (Section 4.1)**

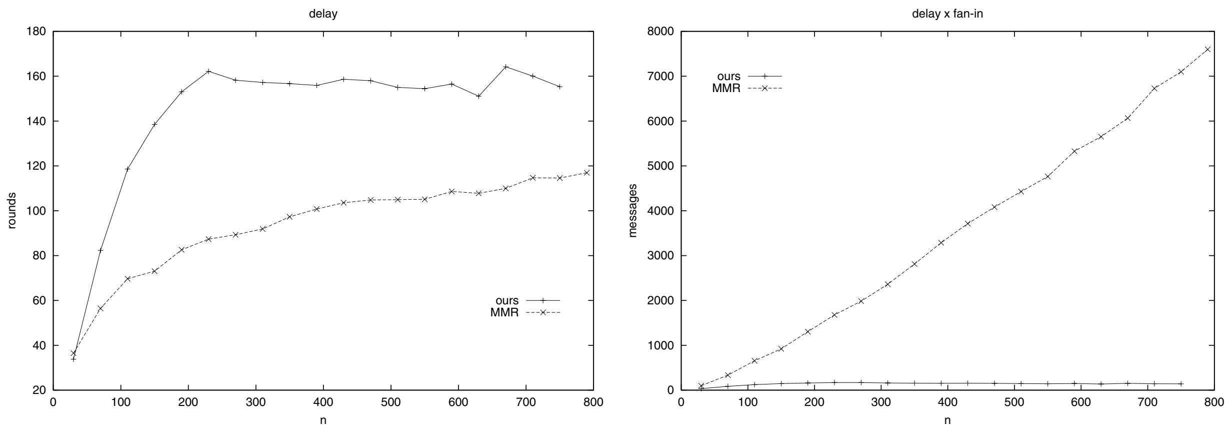

Figure 5. $\lambda = 5$, $b = 2$, **updates introduced to** $\alpha = 3$ **random replicas (Section 4.2)**

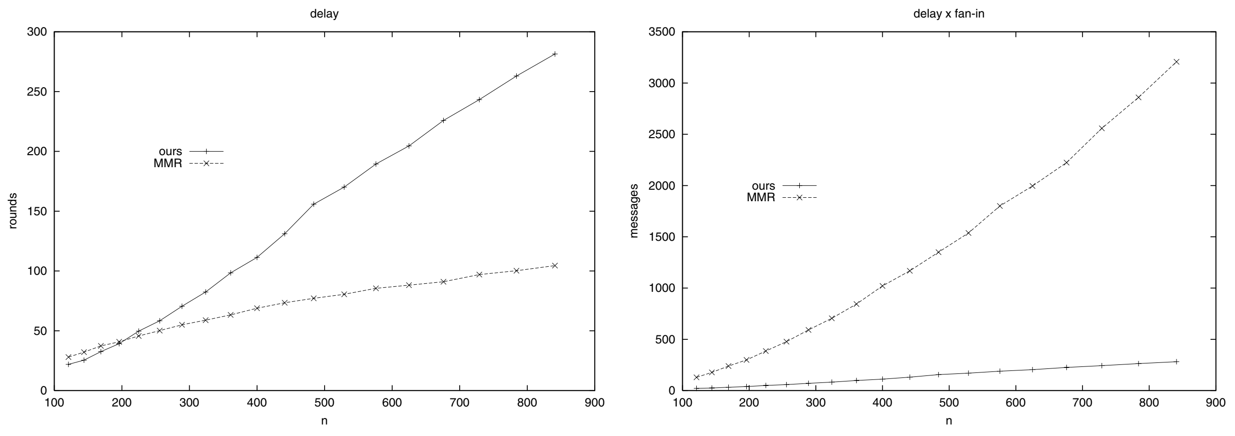

Figure 6. $\lambda = 5$, $b = 5$, **updates introduced to M-Grid quorums (Section 4.1)**

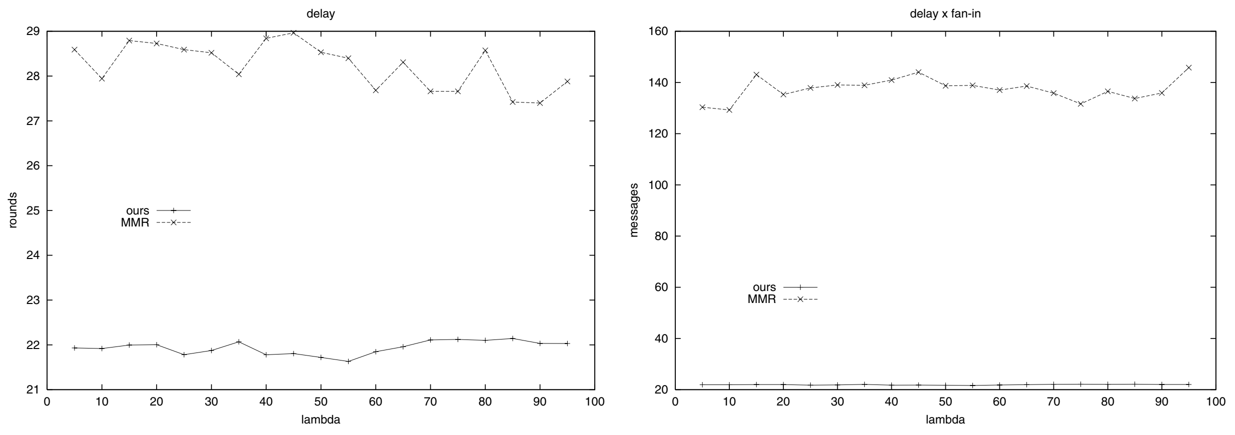

Figure 7. $n = 121$, $b = 5$, **updates introduced to M-Grid quorums (Section 4.1)**

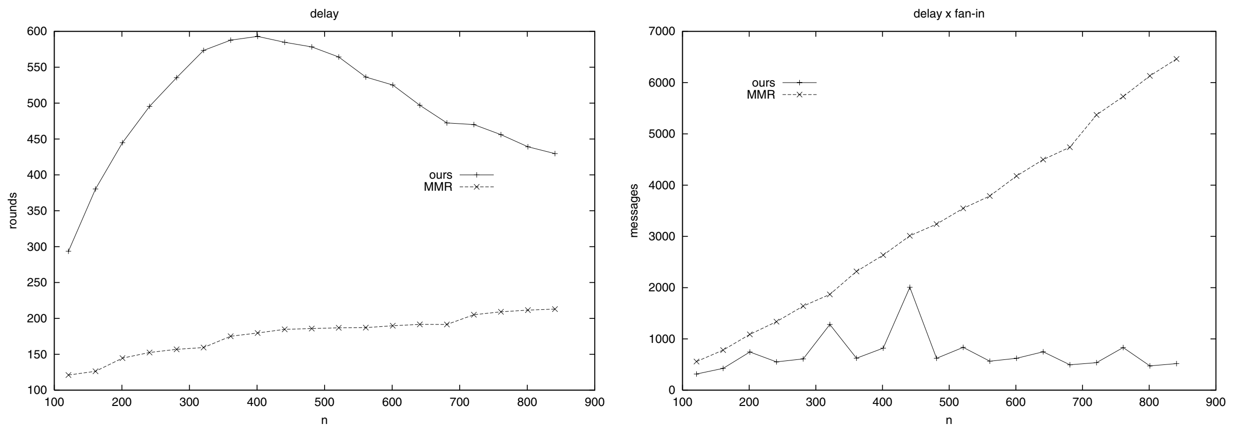

Figure 8. $\lambda = 5$, $b = 5$, **updates introduced to** $\alpha = 6$ **random replicas (Section 4.2)**

References

[And96] R. J. Anderson. The Eternity Service. In *Proceedings of Pragocrypt '96*, 1996.

[BHO+99] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budio and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems* 17(2):41–88, 1999.

[BLNS82] A. D. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder. Grapevine, An exercise in distributed computing. *Communications of the ACM* 25(4):260–274, 1982.

[BT85] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM* 32(4):824–840, October 1985.

[CL99] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, 1999.

[CASD95] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. *Information and Computation* 18(1), pages 158–179, 1995.

[Dee89] S. E. Deering. Host extensions for IP multicasting. SRI Network Information Center, RFC 1112, August 1989.

[DGH+87] 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 *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing*, pages 1–12, 1987.

[DS83] D. Dolev and R. Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal of Computing* 12(4):656–666, 1983.

[GY98] A. Goldberg and P. Yianilos. Towards an archival intermemory. In *Proceedings IEEE ADL*, pages 147–156, 1998.

[KMM98] K. P. Kihlstrom, L. E. Moser and P. M. Melliar-Smith. The SecureRing protocols for securing group communication. In *Proceedings of the 31st IEEE Annual Hawaii International Conference on System Sciences*, vol. 3, pages 317–326, January 1998.

[LOM94] K. Lidl, J. Osborne and J. Malcome. Drinking from the firehose: Multicast USENET news. In *Proceedings of the Usenix Winter Conference*, pages 33–45, January 1994.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems* 4(3):382–401, July 1982.

[MM95] L. E. Moser and P. M. Melliar-Smith. Total ordering algorithms for asynchronous Byzantine systems. In *Proceedings of the 9th International Workshop on Distributed Algorithms*, Springer-Verlag, September 1995.

[MMR99] D. Malkhi, Y. Mansour, and M. K. Reiter. On diffusing updates in a Byzantine environment. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, pages 134–143, October 1999.

[MR97] D. Malkhi and M. Reiter. A high-throughput secure reliable multicast protocol. *Journal of Computer Security* 5:113–127, 1997.

[MR98] D. Malkhi and M. Reiter. Byzantine quorum systems. *Distributed Computing* 11(4):203–213, 1998.

[MR00] D. Malkhi and M. K. Reiter. An architecture for survivable coordination in large-scale systems. *IEEE Transactions on Knowledge and Data Engineering* 12(2):187–202, March/April 2000.

[MRW00] D. Malkhi, M. Reiter, and A. Wool. The load and availability of Byzantine quorum systems. *SIAM Journal of Computing* 29(6):1889–1906, 2000.

[Rei94] M. K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80, November 1994.

[WRC00] M. Waldman, A. D. Rubin, and L. F. Cranor. Publius, A robust, tamper-evident and censorship-resistant web publishing system. In *Proceedings of the 9th USENIX Security Symposium*, 2000.