Scalable Byzantine Reliable Broadcast

Rachid Guerraoui
EPFL, Lausanne, Switzerland
rachid.guerraoui@epfl.ch

Petr Kuznetsov
LTCI, Télécom Paris, IP Paris, Paris, France
petr.kuznetsov@telecom-paristech.fr

Matteo Monti
EPFL, Lausanne, Switzerland
matteo.monti@epfl.ch

Matej Pavlovic
EPFL, Lausanne, Switzerland
matej.pavlovic@epfl.ch

Dragos-Adrian Seredinschi
EPFL, Lausanne, Switzerland
dragos-adrian.seredinschi@epfl.ch

Abstract

Byzantine reliable broadcast is a powerful primitive that allows a set of processes to agree on a message from a designated sender, even if some processes (including the sender) are Byzantine. Existing broadcast protocols for this setting scale poorly, as they typically build on *quorum systems* with strong intersection guarantees, which results in linear per-process communication and computation complexity.

We generalize the Byzantine reliable broadcast abstraction to the probabilistic setting, allowing each of its properties to be violated with a fixed, arbitrarily small probability. We leverage these relaxed guarantees in a protocol where we replace quorums with stochastic *samples*. Compared to quorums, samples are significantly smaller in size, leading to a more scalable design. We obtain the first Byzantine reliable broadcast protocol with logarithmic per-process communication and computation complexity.

We conduct a complete and thorough analysis of our protocol, deriving bounds on the probability of each of its properties being compromised. During our analysis, we introduce a novel general technique that we call adversary decorators. Adversary decorators allow us to make claims about the optimal strategy of the Byzantine adversary without imposing any additional assumptions. We also introduce Threshold Contagion, a model of message propagation through a system with Byzantine processes. To the best of our knowledge, this is the first formal analysis of a probabilistic broadcast protocol in the Byzantine fault model. We show numerically that practically negligible failure probabilities can be achieved with realistic security parameters.

2012 ACM Subject Classification Mathematics of computing → Probabilistic algorithms; Mathematics of computing → Stochastic processes; Theory of computation → Distributed algorithms; Theory of computation → Distributed computing models

Keywords and phrases Byzantine reliable broadcast, probabilistic distributed algorithms, scalable distributed systems, stochastic processes

Digital Object Identifier 10.4230/LIPIcs.DISC.2019.22

Related Version An extended version of this article – which includes a thorough analysis of our protocols – appears on arXiv at `https://arxiv.org/abs/1908.01738v1` [34].

Funding This work has been supported in part by the European ERC Grant 339539 - AOC.

1 Introduction

Broadcast is a popular abstraction in the distributed systems toolbox, allowing a process to transmit messages to a set of processes. The literature defines many flavors of broadcast, with different safety and liveness guarantees [14, 25, 35, 42, 48]. In this paper we focus on Byzantine reliable broadcast, as defined by Bracha [12]. This abstraction is a central building block in practical Byzantine fault-tolerant (BFT) systems [15, 19, 33]. We tackle the problem of its scalability, namely reducing the complexity of Byzantine reliable broadcast, and seeking good performance despite a large number of participating processes.

In Byzantine reliable broadcast, a designated sender broadcasts a single message. Intuitively, the broadcast abstraction ensures that no two correct processes deliver different messages (*consistency*), either all correct processes deliver a message or none does (*totality*), and that, if the sender is correct, all correct processes eventually deliver the broadcast message (*validity*). This must hold despite a certain fraction of Byzantine processes, potentially including the sender. We denote by N the number of processes in the system, and f the fraction of processes that are Byzantine. Existing algorithms for Byzantine reliable broadcast scale poorly as they typically have $O(N)$ per-process communication complexity [13, 42, 45, 53]. The root cause for poor scalability of these algorithms is their use of quorums [43, 56], i.e., sets of processes that are large enough to always intersect in at least one correct process. The size of a quorum grows linearly with the size of the system [14].

To overcome the scalability limitation of quorum-based broadcast, Malkhi *et al.* [46] generalize quorums to the probabilistic setting. In this setting, two random quorums intersect with a fixed, arbitrarily high probability, allowing the size of each quorum to be reduced to $O(\sqrt{N})$. We are not aware of any Byzantine reliable broadcast algorithm building on probabilistic quorums; nevertheless, such an algorithm could have a per-process communication complexity reduced from $O(N)$ to $O(\sqrt{N})$. The active$_t$ protocol [42] uses a form of samples for an optimistic path, but relies on synchrony and has a linear worst-case complexity (that is arguably very likely to occur with merely a moderate number of faulty processes).

1.1 Samples

In this paper, we present a probabilistic gossip-based Byzantine reliable broadcast algorithm having $O(\log N)$ per-process communication and computation complexity, at the expense of $O(\log N / \log \log N)$ latency. Essentially, we propose *samples* as a replacement for quorums. Like a probabilistic quorum, a sample is a randomly selected set of processes. Unlike quorums, samples do not need to intersect. Samples can be significantly smaller than quorums, as each sample must be large enough only to be *representative* of the system with high probability.

A process can use its sample to gather information about the global state of the system. An old Italian saying provides an intuitive understanding of this shift of paradigm: *"To know if the sea is salty, one needs not drink all of it!"* Intuitively, we leverage the law of large numbers, trading performance for a fixed, arbitrarily small probability of non-representativeness. [1]

Throughout this paper, we extensively use samples to estimate the number of processes satisfying a set of *yes-or-no* predicates, e.g., the number of processes that are ready to deliver a message m. Consider the case where a correct process π queries K randomly selected

[1] To get an intuition of the difference between quorums and samples, consider the emulation of a shared memory in message passing [3]. One writes in a quorum and reads from a quorum to fetch the last value written. Our algorithms are rather in the vein of "write all, read any." Here we would "write" using a gossip primitive and "sample" the system to seek the last written value.

processes (a sample) for a predicate P. Assume that a fraction p of correct processes from the whole system satisfies predicate P. Let x be the fraction of positive responses (out of K) that π collects. By the Chernoff bound, the probability of $|x - p| \geq f + \epsilon$ is smaller or equal to $\exp(-\lambda(\epsilon)K)$, where λ quickly increases with ϵ. For sufficient K, the probability of x differing from p by more than $f + \epsilon$ can be made exponentially small.

Our algorithms use a *sampling oracle* that returns the identity of a process from the system picked with uniform probability. In a permissioned system (i.e., one where the set of participating processes is known) sampling reduces to picking with uniform probability an element from the set of processes. In a permissionless system subject to Byzantine failures and slow churn, a (nearly) uniform sampling mechanism is achievable using gossip [10].

1.2 Scalable Byzantine Reliable Broadcast

Our probabilistic algorithm, Contagion, allows each property of Byzantine reliable broadcast to be violated with an arbitrarily small probability ϵ. We show that ϵ scales sub-quadratically with N, and decays exponentially in the size of the samples. As a result, for a fixed value of ϵ, the per-node communication complexity of Contagion is logarithmic.

We build Contagion incrementally, relying on two sub-protocols, as we describe next.

First, **Murmur** is a probabilistic broadcast algorithm that uses simple message dissemination to establish *validity* and *totality*. In this algorithm, each correct process relays the sender's message to a randomly picked *gossip sample* of other processes. For the sample size $\Omega(\log N)$, the resulting gossip network is a connected graph with $O(\log N / \log \log N)$ diameter, with high probability [21, 17]. In case of a Byzantine sender, however, Murmur does not guarantee consistency.

Second, **Sieve** is a probabilistic consistent broadcast algorithm (built upon Murmur) that guarantees *consistency*, i.e., no two correct processes deliver different messages. To do so, each correct process uses a randomly selected *echo sample*. Intuitively, if enough processes from any echo sample confirm a message m, then with high probability no correct processes in the system delivers a different message m'. Sieve, however, does not ensure totality. If a Byzantine sender broadcasts multiple conflicting messages, a correct process might be unable to gather sufficient confirmations for either of them from its echo sample, and consequently would not deliver any message, even if some correct process delivers a message.

Finally, **Contagion** is a probabilistic Byzantine reliable broadcast algorithm that guarantees validity, consistency, and totality. The sender uses Sieve to disseminate a consistent message to a subset of the correct processes. In order to achieve totality, Contagion mimics the spreading of a contagious disease in a population. A process samples the system and if it observes enough other "infected" processes in its sample, it becomes infected itself. If a critical fraction of processes is initially infected by having received a message from the underlying Sieve layer, the message spreads to all correct processes with high probability. If a process observes enough other infected processes, it delivers. As in the original deterministic implementation by Bracha [12], the crucial point here is that "enough" for becoming infected is less than "enough" for delivering. This way, with high probability, either all correct processes deliver a message or none does – Contagion satisfies totality. The other two important properties (validity and consistency) are inherited from the underlying (Contagion and Sieve) layers.

1.3 Probability Analysis and Applications

A major technical contribution of this work is a complete, formal analysis of the properties of our three algorithms. To the best of our knowledge, this is the first such analysis applied to a probabilistic broadcast algorithm in the Byzantine fault model, and this turned out to be challenging. Intuitively, providing a bound on the probability of a property being violated reduces to studying a joint distribution between the inherent randomness of the system and the behavior of the Byzantine adversary. Since the behavior of the adversary is arbitrary, the marginal distribution of the Byzantine's behavior is unknown.

We develop two novel strategies to bound the probability of a property being violated, which we use in the analysis of Sieve and Contagion respectively.

(1) When evaluating the consistency of Sieve, we show that a bound holds for every possibly optimal adversarial strategy. Essentially, we identify a subset of adversarial strategies that we prove to include the optimal one, i.e., the one that has the highest probability of compromising the consistency of Sieve. We then prove that every possibly optimal adversarial strategy has a probability of compromising the consistency of Sieve smaller than some ϵ.

(2) When evaluating the totality of Contagion, we show that the adversarial strategy does not affect the outcome of the execution. Here, we show that any adversarial strategy reduces to a well-defined sequence of choices. We then prove that, due to the limited knowledge of the Byzantine adversary, every choice is equivalent to a random one.

Our analysis shows that, for a practical choice of parameters, the probability of violating the properties of our algorithm can be brought down to 10^{-16} for systems with thousands of processes.

In the rest of this paper, we state our system model and assumptions (Section 2), and then present our Murmur, Sieve, and Contagion algorithms (Sections 3 to 5). Our algorithm descriptions are high-level, but throughout this paper we will often refer the interested reader to the corresponding appendices containing all details (including pseudocode and formal proofs); to respect conference proceedings space limits, we place these appendices in the extended version of this article [34]. We discuss the security and complexity of our algorithms in Section 6, and cover related work in Section 7.

2 Model and Assumptions

We assume an asynchronous message-passing system where the set Π of $N = |\Pi|$ processes partaking in an algorithm is fixed. Any two processes can communicate via a reliable authenticated point-to-point link.

We assume that each correct process has access to a local, unbiased, independent source of randomness. We assume that every correct process has direct access to an oracle Ω that, provided with an integer $n \leq N$, yields the identities of n distinct processes, chosen uniformly at random from Π. Implementing Ω is beyond the scope of this paper, but it is straightforward in practice. In a system where the set of participating processes is known, sampling reduces to picking with uniform probability an element from the set of processes. In a system without a global membership view that may even be subject to slow churn, a (nearly) uniform sampling mechanism is available in literature due to Bortnikov *et al.* [10].

At most a fraction f of the processes are Byzantine, i.e., subject to arbitrary failures [40]. Byzantine processes may collude and coordinate their actions. Unless stated otherwise, we denote by $\Pi_C \subseteq \Pi$ the set of correct processes and by $C = |\Pi_C| = (1 - f)N$ the number of

correct processes. We assume a static Byzantine adversary controlling the faulty processes, i.e., the set of processes controlled by the adversary is fixed at the beginning and does not change throughout the execution of the protocols.

We make standard cryptographic assumptions regarding the power of the adversary, namely that it cannot subvert cryptographic primitives, e.g., forge a signature. We also assume that Byzantine processes are not aware of (1) the output of the local source of randomness of any correct process; and (2) which correct processes are communicating with each other. The latter assumption is important to prevent the adversary from poisoning the view of the system of a targeted correct process without having to bias the local randomness source of any correct process. Even against ISP-grade adversaries, we can implement this assumption in practice by means such as onion routing [18] or private messaging [54].

3 Probabilistic Broadcast with Murmur

In this section, we introduce the *probabilistic broadcast* abstraction and its implementation, Murmur. Briefly, probabilistic broadcast ensures validity and totality. We use this abstraction in Sieve (Section 4) to initially distribute the message from a sender to all correct processes.

The probabilistic broadcast interface assumes a specific sender process σ. An instance pb of probabilistic broadcast exports two events. First, process σ can request through $\langle pb.\text{Broadcast} \mid m\rangle$ to broadcast a message m. Second, the indication event $\langle pb.\text{Deliver} \mid m\rangle$ is an upcall for delivering message m broadcast by σ. For any $\epsilon \in [0, 1]$, we say that probabilistic broadcast is ϵ-secure if:

1. **No duplication:** No correct process delivers more than one message.
2. **Integrity:** If a correct process delivers a message m, and σ is correct, then m was previously broadcast by σ.
3. **ϵ-Validity:** If σ is correct, and σ broadcasts a message m, then σ eventually delivers m with probability at least $(1 - \epsilon)$.
4. **ϵ-Totality:** If a correct process delivers a message, then every correct process eventually delivers a message with probability at least $(1 - \epsilon)$.

3.1 Gossip-based Algorithm

Murmur (presented in detail in [34, Appendix A, Algorithm 1]) distributes a single message across the system by means of gossip: upon reception, a correct process relays the message to a set of randomly selected neighbors. The algorithm depends on one parameter: *expected gossip sample size* G.

Upon initialization, every correct process uses the sampling oracle Ω to select (on average) G other processes to gossip with. Gossip links are reciprocated, making the gossip graph undirected.

To broadcast a message m, the designated sender σ signs m and sends it to all its neighbors. Upon receiving a correctly signed message m from σ for the first time, each correct process delivers m and forwards m to every process in its neighborhood.

3.2 Analysis Using Erdös-Rényi Graphs

The detailed analysis, provided in [34, Appendix A, Sections A.3 and A.4], formally proves the correctness of Murmur by deriving a bound on ϵ as a function of the algorithm and system parameters. Here we give a very high-level sketch of our probabilistic analysis of Murmur.

3.2.1 No duplication, integrity and ϵ-validity

Here is the intuition behind the properties satisfied by Murmur [34, Appendix A.3]:

1. No duplication: A correct process maintains a *delivered* variable that it checks and updates when delivering a message, preventing it from delivering more than one message.
2. Integrity: Before broadcasting a message, the sender signs that message with its private key. Before delivering a message m, a correct process verifies m's signature. This prevents any correct process from delivering a message that was not previously broadcast by the sender.
3. ϵ-Validity: Upon broadcasting a message, the sender also immediately delivers it. Since this happens *deterministically*, Murmur satisfies 0-validity, independently from the parameter G.

3.2.2 ϵ-Totality

Murmur satisfies ϵ-totality with ϵ upper-bounded by a function that decays exponentially with G, and increases polynomially with f [34, Appendix A.4]. We prove that the network of connections established among the correct processes is an undirected Erdős–Rényi graph [21]. Totality is satisfied if such graph is connected.

Erdős–Rényi graphs are well known in literature [1] to display a connectivity phase transition: when the expected number of connections each node has exceeds the logarithm of the number of nodes, the probability of the graph being connected steeply increases from 0 to 1 (in the limit of infinitely large systems, this increase becomes a step function). We use this result to compute the probability of the sub-graph of correct processes being connected and, consequently, of Murmur satisfying totality ([34, Theorem 4]).

4 Probabilistic Consistent Broadcast with Sieve

In this section, we first introduce the probabilistic consistent broadcast abstraction, which allows (a subset of) the correct processes to agree on a single message from a (potentially Byzantine) designated sender. We then discuss Sieve, an implementation of this abstraction. We use probabilistic consistent broadcast in the implementation of Contagion (see Section 5) as a way to consistently disseminate messages. Sieve itself builds on top of probabilistic broadcast (see Section 3).

Probabilistic consistent broadcast does not guarantee totality, but it does guarantee consistency: despite a Byzantine sender, no two correct processes deliver different messages. If the sender is Byzantine, however, it may happen with a non-negligible probability that only a proper subset of the correct processes deliver the message.

For any $\epsilon \in [0, 1]$, we say that probabilistic consistent broadcast is ϵ-secure if it satisfies the properties of **No duplication** and **Integrity** as defined above, and:

- **ϵ-Total validity:** If σ is correct, and σ broadcasts a message m, every correct process eventually delivers m with probability at least $(1 - \epsilon)$.
- **ϵ-Consistency:** With probability at least $(1 - \epsilon)$, no two correct processes deliver different messages.

4.1 Sample-Based Algorithm

Sieve (presented in detail in [34, Appendix B, Algorithm 3]) uses `Echo` messages to consistently distribute a single message to (a subset of) the correct processes: before delivering a message, a correct process samples the system to estimate how many other processes received the same message. The algorithm depends on two parameters: the *echo sample size* E and the *delivery threshold* $\hat{E}$.

Upon initialization, every correct process uses the sampling oracle Ω to select an *echo sample* $\mathcal{E}$ of size E, and sends an `EchoSubscribe` message to every process in $\mathcal{E}$. Upon broadcasting, the sender uses the underlying probabilistic broadcast (e.g., Murmur) to initially distribute a message to every correct process. This step does not ensure consistency, so processes may see conflicting messages if the sender σ is Byzantine. Upon receiving a message m from probabilistic broadcast, a correct process π sends an $(\texttt{Echo}, m)$ message to every process that sent an `EchoSubscribe` message to π. (Note that, due to the no duplication property of probabilistic broadcast, this can happen only once per process.) Upon collecting $\hat{E}$ $(\texttt{Echo}, m)$ messages from its echo sample $\mathcal{E}$, π delivers m. Notably, if π delivers m, then with high probability every other correct process either also delivers m, or does not deliver anything at all, but never delivers $m' \neq m$.

4.2 Analysis Using Adversary Decorators

Here we present a high-level outline of the analysis of Sieve; for a complete formal treatment, see [34, Appendix B], where we prove the correctness of Sieve by deriving a bound on ϵ.

4.2.1 No duplication and integrity

Sieve deterministically satisfies these properties the same way as Murmur does [34, Appendix B.3].

4.2.2 ϵ-Total Validity

Since we assume a correct sender σ (by the premise of total validity), a bound on the probability ϵ of violating total validity can easily be derived from the probability of the underlying probabilistic broadcast failing and from the probability of some process' random echo sample having more than $E - \hat{E}$ Byzantine processes [34, Appendix B.4].

4.2.3 ϵ-Consistency

While the intuition why Sieve satisfies consistency is rather simple, proving it formally is the most technically involved part of this paper. We now provide the intuition and present the techniques we use to prove it, while deferring the full body of the formal proof to [34, Appendices B.5-B.10].

In order for Sieve to violate consistency, two correct processes must deliver two different messages (which can only happen if the sender σ is malicious). This, in turn, means that two correct processes π and π' must observe two different messages m and m' sufficiently represented in their respective echo samples. I.e., π receives $(\texttt{Echo}, m)$ at least $\hat{E}$ times and π' receives $(\texttt{Echo}, m')$ at least $\hat{E}$ times.

Note that a correct process only sends $(\texttt{Echo}, m)$ for a single message m received from the underlying probabilistic broadcast layer. The intuition of Sieve is the same as in quorum-based algorithms. With quorums, if enough correct processes issue $(\texttt{Echo}, m)$ to make at

least one correct process deliver m, the remaining processes (regardless of the behavior of the Byzantine ones) are not sufficient to make any other correct process deliver m'. For Sieve, this holds with high probability as long as $\hat{E}$ is sufficiently high and the fraction f of Byzantine processes is limited.

To prove these intuitions, we first describe Simplified Sieve [34, Appendix B.6], a strawman variant of Sieve that is easier to analyze. We prove that Simplified Sieve guarantees consistency with strictly lower probability than Sieve does [34, Appendix B.8, Lemma 12]. Thus, an upper bound on the probability of Simplified Sieve failing is also an upper bound on the probability of Sieve failing.

Next, we analyze Simplified Sieve using a novel technique that involves modeling the adversary as an algorithm that interacts with the system through a well-defined interface [34, Appendix B.7]. We start from the set of all possible adversarial algorithms and gradually reduce this set, while proving that the reduced set still includes an *optimal* adversary [34, Appendix B.9]. (An adversary is optimal if it maximizes the probability ϵ of violating consistency.) Intuitively, we prove that certain actions of the adversary always lead to strictly lowering ϵ, and thus need not be considered. For example, an adversary can only decrease its chance of compromising consistency when omitting `Echo` messages.

To this end, we introduce the concept of *decorators*. A decorator is an algorithm that lies between an adversary and a system. It emulates a system and exposes the corresponding interface to the decorated adversary. At the same time, the decorator also exposes the interface of an adversary to interact with a system. The purpose of a decorator is to alter the interaction between the adversary and the system. For any decorated adversary, we prove that the decorator does not decrease the probability ϵ of the adversary compromising the system. Thus, a decorator effectively transforms an adversary into a stronger one. Each decorator maps a set of adversaries into one of its proper subsets that is easier to analyze [34, Appendix D].

Through a series of decorators, we obtain a tractable set of adversaries that provably contains an optimal one. Then we derive the bound on ϵ under these adversaries [34, Theorem 9].

5 Probabilistic Byzantine Reliable Broadcast with Contagion

Our main algorithm, Contagion, implements the probabilistic Byzantine reliable broadcast abstraction. This abstraction is strictly stronger than probabilistic consistent broadcast, as it additionally guarantees ϵ-totality. Despite a Byzantine sender, either none or every correct process delivers the broadcast message.

For any $\epsilon \in [0, 1]$, we say that probabilistic Byzantine reliable broadcast is ϵ-secure if it satisfies the properties of **No duplication**, **Integrity**, **ϵ-Validity**, **ϵ-Consistency** and **ϵ-Totality**, as already defined in previous sections.

5.1 Feedback-Based Algorithm

Our algorithm implementing probabilistic Byzantine reliable broadcast is called Contagion and we present it in detail in [34, Appendix C, Algorithm 7]. It uses a feedback mechanism to securely distribute a single message to every correct process. The main challenge of Contagion is to ensure totality; we prove that the other properties are easily inherited from the underlying layer with high probability.

The basic idea of Contagion roughly corresponds to the last stage of Bracha's broadcast algorithm [12]. During the execution of Contagion for message m, processes first become *ready* for m. A correct process π can become ready for m in two ways:

1. π receives m from the underlying consistent broadcast layer.
2. π observes a certain fraction of other processes being ready for m.

A correct process delivers m only after it observes enough other processes being ready for m.

Unlike Bracha, we use samples (as opposed to quorums) to assess whether enough nodes are ready for m (and consequently our results are all probabilistic in nature). Upon initialization, every correct process selects a ready sample $\mathcal{R}$ of size R and a delivery sample $\mathcal{D}$ of size D. Our algorithm depends on four parameters: the *ready and delivery sample sizes* R and D, and the *ready and delivery thresholds* $\hat{R}$ and $\hat{D}$.

The delivery sample $\mathcal{D}$ is the sample used to assess whether m can be delivered. A correct process π delivers m if at least $\hat{D}$ out of the D processes in π's delivery sample are ready for m.

The purpose of the ready sample $\mathcal{R}$ is to create a feedback loop, a crucial part of the Contagion algorithm. When a correct process π observes at least $\hat{R}$ out of the R other processes in π's ready sample to be ready for m, π itself becomes ready for m. A direct consequence of such a feedback loop is the existence of a critical fraction of processes that, when ready for m, cause all the other correct processes become ready for m with high probability.

We require that $\hat{R}/R < \hat{D}/D$, i.e., the fraction of ready processes π needs to observe in order to become ready itself is smaller than the fraction of ready processed required for π to deliver m. Totality is then implied by the following intuitive argument. If a correct process π delivers m, it must have observed a fraction of at least $\hat{D}/D$ other processes being ready for m. As this fraction is higher than the critical fraction required for all correct processes to become ready for m, all correct processes will eventually become ready for m. Consequently, all correct processes will eventually deliver m. On the other hand, if too few processes are initially ready for m, such that the critical fraction is not reached, with high probability no correct process will observe the (even higher) fraction $\hat{D}/D$ of ready processes in its sample. Consequently, no correct process delivers m.

To broadcast a message m, the sender σ initially uses probabilistic consistent broadcast (Section 4) to disseminate m consistently to (a subset of) the correct processes. All correct processes that receive m through probabilistic consistent broadcast become ready for m. If their number is sufficiently high, according to the mechanism described above, all correct processes deliver m with high probability. If only a few correct processes deliver receive m from probabilistic consistent broadcast, with high probability no correct process delivers m.

5.2 Threshold Contagion Game

Before presenting the analysis of Contagion, we overview the *Threshold Contagion* game, an important tool in our analysis. In this game, we simulate the spreading of a contagious disease (without a cure) among members of a population, the same way the "readiness" for a message spreads among correct processes that execute our Contagion algorithm.

Threshold Contagion is played on the nodes of a directed multigraph, where each node represents a member of a population (whose state is either *infected* or *healthy*), and each edge represents a *can-infect* relation. An edge (a, b) means that a can infect b. We also call a the *predecessor* of b. In our Contagion algorithm, this corresponds to a being in the ready sample of b. Analogously to Contagion, a node becomes infected when enough of its predecessors are infected.

Threshold Contagion is played by one player in one or more *rounds*. At the beginning of each round, the player infects a subset of the healthy nodes. In the rest of the round, the infection (analogous to the readiness for a message) propagates as follows. A healthy

node that reaches a certain threshold ($\hat{R}$) of infected predecessors becomes infected as well (potentially contributing to the infection of more nodes). The round finishes when no healthy node has $\hat{R}$ or more infected predecessors, or when all nodes are infected.

In the analogy with our Contagion algorithm, infection by a player at the start of each round corresponds to a process receiving a message from the underlying probabilistic consistent broadcast layer. Infection through other nodes is analogous to observing $\hat{R}$ ready processes in the ready sample.

We analyze the Threshold Contagion game, and compute the probability distribution underlying the number of nodes that are infected at the end of a each round, depending on the number of healthy nodes infected by the player. Applying this analysis to the Contagion algorithm (the adversary being the player), we obtain the probability distribution of the number of processes ready for a message, which, in turn, allows us to compute a bound on the probability of violating the properties of Contagion. We provide all details on the Threshold Contagion game itself in [34, Appendix E].

5.3 Analysis Using Threshold Contagion

Here we present an outline of the analysis of Contagion; for a full formal treatment, see [34, Appendix C].

5.3.1 No duplication and integrity

Contagion deterministically satisfies these properties the same way as our previous algorithms do [34, Appendix C.3].

5.3.2 ϵ-Validity

Assuming a correct sender σ (by the premise of validity), we derive a bound on the probability ϵ of violating validity from the probability of the underlying probabilistic consistent broadcast failing and from the probability of σ's random delivery sample containig more than $D - \hat{D}$ Byzantine processes [34, Appendix C.4].

5.3.3 ϵ-Consistency

When computing the upper bound on the probability ϵ of compromising consistency [34, Appendix C.9], we assume that if the consistency of the underlying probabilistic consistent broadcast is compromised, then the consistency of probabilistic Byzantine reliable broadcast is compromised as well. The rest of the analysis assumes that probabilistic Byzantine reliable broadcast is consistent.

In such case, every correct process receives at most one message m^* from the underlying probabilistic consistent broadcast. Simply by acting correctly, Byzantine processes can cause any correct process to eventually deliver m^*. Consistency is compromised if the adversary can also cause at least one correct process to deliver a message $m \neq m^*$, given that no correct process becomes ready for m by receiving it through the underlying probabilistic consistent broadcast.

We start by noting that, since a correct process π can be ready for an arbitrary number of messages, the set of processes that are eventually ready for m is not affected by which processes are eventually ready for a message m^*. If enough processes in π's delivery sample are eventually ready both for m and m^*, then π can deliver either m or m^*. In this case, the adversary (who controls the network scheduling, see Section 2) decides which message π delivers.

The probability of m being delivered by any correct process is maximized when every Byzantine process behaves as if it was ready for m [34, Appendix C.9, Lemma 28]. Note that a Byzantine process being ready for m behaves identically to a correct process that receives m through probabilistic consistent broadcast. We model the adversarial system using a single-round game of Threshold Contagion where both correct and Byzantine processes are represented as nodes in the multigraph and all nodes representing Byzantine processes are initially infected [34, Appendix C.7, Lemma 26].

Given the distribution of the number of correct processes that are ready for m at the end Threshold Contagion, we compute the probability that at least one correct process will deliver $m \neq m^*$. This probability, combined with the probability that the consistency of probabilistic consistent broadcast is violated, yields the probability ϵ of violating the consistency of Contagion.

5.3.4 ϵ-Totality

Again, to compute an upper bound on the probability of our algorithm compromising totality, we assume that compromising the consistency of probabilistic consistent broadcast also compromises the totality of probabilistic Byzantine reliable broadcast. Assuming that probabilistic consistent broadcast satisfies consistency, at most one message m^* is received by any correct process through the underlying probabilistic consistent broadcast. We loosen the bound on the probability of compromising totality (and simplify analysis) by considering totality to be compromised if any message $m \neq m^*$ is delivered by any correct process. This allows us to focus on message m^*. We further loosen the bound by assuming that the Byzantine adversary can arbitrarily cause any correct process to become ready for m^*. Whenever this happens, zero or more additional correct processes will also become ready for m^* as a result of the feedback loop described in Section 5.1. To compromise totality, there must exists at least one correct process that delivers m^* and at least one correct process does not.

We prove [34, Appendix C.10.3, Lemma 31] that the optimal adversarial strategy to compromise totality is to repeat the following. (1) Make a correct node ready for m^*. (2) Wait until the "readiness" propagates to zero or more correct nodes. (3) Have specific Byzantine processes behave as correct processes ready for m^*, if this leads to some (but not all) correct processes delivering m^*. Totality is satisfied if, after every step of the adversary, either the feedback loop makes all correct processes deliver m^* (relying only on correct processes' ready samples), or no correct process delivers m^* (even with the "support" of Byzantine processes) [34, Theorem 14]. Otherwise, totality is violated.

We study this behavior with a multi-round game of Threshold Contagion, where only correct processes are represented as nodes in the multigraph and, at the beginning of each round, the player (i.e., the adversary) infects one uninfected node. From the probability distribution of the number of infected nodes after each round, we derive the probability of compromising totality by message m^*. This probability equals to the probability that there is at least one round after which the number of infected nodes allows some but not all the processes to deliver m^*.

6 Security and Complexity Evaluation

In Sections 3 to 5, we introduced three algorithms, Murmur, Sieve and Contagion, and outlined their analysis (deferring the formal details to the appendices).

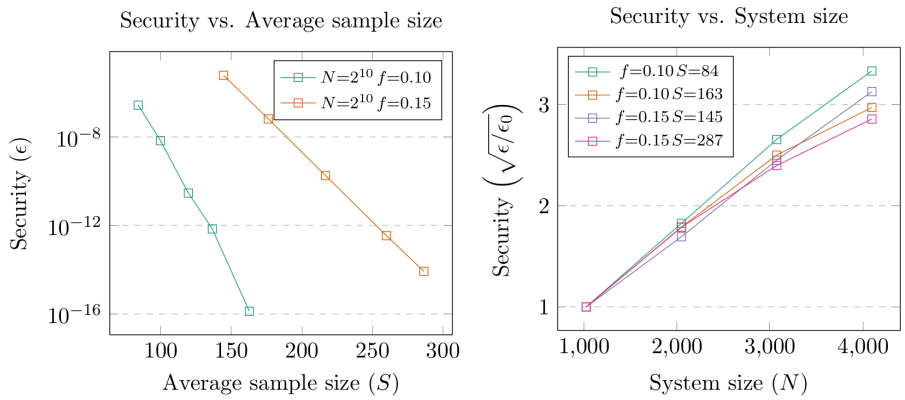

Figure 1 Left – ϵ-security of Contagion, as a function of the average sample size $S = \langle G, E, R, D \rangle$. We use a system size of 1024 processes and fractions of tolerated Byzantine processes $f = 0.1$ and $f = 0.15$. **Right** – Square root of the normalized ϵ-security of Contagion, as a function of the system size N, for various fractions of Byzantine processes (f) and average sample sizes (S). We normalize the values in each series by the first element of that series. All lines appearing to grow sub-linearly with a square-rooted y-axis demonstrates that the normalized ϵ security grows sub-quadratically.

The modular design of our algorithm allows us to study its components independently. We employ numerical techniques to maximize the ϵ-security of Contagion, under the constraint that the sum of all the sample sizes of a process is constant ($G + E + R + D = const$). Since a process communicates with all the processes in its samples, this corresponds to a fixed communication complexity.

For a given system size N and fraction of Byzantine processes f, we relate this per-process communication complexity to the ϵ-security of Contagion. As Figure 1 (left) shows, the probability ϵ of compromising the security of Contagion decays exponentially in the average sample size S.

We also study how the ϵ-security of Contagion changes as a function of the system size N, for a fixed set of parameters (G, E, R, D). Figure 1 (right) shows that the ϵ-security is bounded by a quadratic function in N. Thus, for a fixed security ϵ, the average sample size (and consequently, the communication complexity of our algorithm) grows logarithmically with the system size N.

Given that a process π only exchanges a constant number of messages with each member of π's samples, and the sample size is logarithmic in system size, each node needs to exchange $O(\log N)$ messages. Thus, for N nodes in the system, the overall message complexity is $O(N \log N)$. The latency in terms of message delays between broadcasting and delivery of a message is $O(\log N / \log \log N)$. Specifically, the latency converges to $O(\log N / \log \log N)$ message delays for gossip-based dissemination with Murmur (we prove this in [34, Appendix A.4, Theorem 5]), and 2 message delays in total for `Echo` (Sieve) and `Ready` (Contagion) messages.

7 Related Work

At its base, our broadcast algorithm relies on gossip. There is a great body of literature studying various aspects of gossip, proposing flavors of gossip protocols for different environments and analyzing their complexities [2, 6, 8, 7, 4, 20, 23, 30, 52, 28, 26, 27, 55, 57, 29, 36]. However, to the best of our knowledge, we propose the first highly scalable gossip-based reliable broadcast protocol resilient to Byzantine faults with a thorough probabilistic analysis.

The communication pattern in the implementation of both our Sieve and Contagion algorithms can be traced back to the Asynchronous Byzantine Agreement (ABA) primitive of Bracha and Toueg [13] and the subsequent line of work [12, 15, 42, 50]. Indeed, our echo-based mechanism in Sieve resembles algorithms from classic quorum-based systems for Byzantine consistent broadcast [53, 49]. The ready-based mechanism in Contagion is inspired by a two-phase protocol appearing in several practical (quorum-based) systems [15, 19, 44]. Compared to classic work on this topic, the key feature of Contagion and Sieve is that they replace the building block of quorum systems with stochastic samples, thus enabling better scalability for the price of abandoning deterministic guarantees.

There is significant prior work on using epidemic algorithms to implement scalable *reliable* broadcast [9, 22, 37, 41]. Under benign failures or constant churn, these algorithms ensure, with high probability, that every broadcast message reaches all or none, and that all messages from correct senders are delivered. Our goal is to additionally provide *consistency* for broadcast messages, and tolerate *Byzantine* environments [13, 45, 53]. To the best of our knowledge, we are the first to apply the epidemic sample-based methodology in this context. Our main algorithm Contagion scales well to dynamic systems of thousands of nodes, some of which may be Byzantine. This makes it a suitable choice for *permissionless* settings that are gaining popularity with the advent of blockchains [47].

Distributed clustering techniques seek to group the processes of a system into clusters, sometimes called shards or quorums, of size $O(logN)$ [5, 31, 32, 38, 39, 51]. This line of work has various goals (e.g., leader election, "almost everywhere" agreement, building an overlay network) and they also aim for scalable solutions. The overarching principle in clustering techniques is similar to our use of samples: build each cluster in a provably random manner so that the adversary cannot dominate any single cluster. Samples in our solution are private and individual on a per-process basis, in contrast to clusters which are typically public and global for the whole system.

The idea of *communication locality* appears in the context of secure multi-party computation (MPC) protocols [11, 16, 24]. This property captures the intuition that, in order to obtain scalable distributed protocols and permit a large number of participants, it is desirable to limit the number of participants each process must communicate with. All of our three algorithms have this communication locality property, since each process coordinates only with logarithmically-sized samples. In contrast to secure MPC protocols, our algorithms have different goals, system model, or assumptions (e.g., we do not assume a client-server model [24], nor do we seek to address privacy issues). Our algorithms can be used as building blocks towards helping tackle scalability in MPC protocols, and we consider this an interesting avenue for future work.

References

1 Daron Acemoglu and Asu Ozdaglar. 6.207/14.15: Networks - Lecture 4: Erdős–Rényi Graphs and Phase Transitions, 2009. URL: `https://economics.mit.edu/files/4622`.

2 Dan Alistarh, Seth Gilbert, Rachid Guerraoui, and Morteza Zadimoghaddam. How Efficient Can Gossip Be? (on the Cost of Resilient Information Exchange). In *Proceedings of the 37th International Colloquium Conference on Automata, Languages and Programming: Part II*, ICALP'10, pages 115–126, Berlin, Heidelberg, 2010. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=1880999.1881012`.

3 Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *JACM*, 42(1), 1995.

4 Chen Avin, Michael Borokhovich, Keren Censor-Hillel, and Zvi Lotker. Order Optimal Information Spreading Using Algebraic Gossip. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '11, pages 363–372, New York, NY, USA, 2011. ACM. `doi:10.1145/1993806.1993883`.

5 Baruch Awerbuch and Christian Scheideler. Towards a scalable and robust DHT. *Theory of Computing Systems*, 45(2):234–260, 2009.

6 Petra Berenbrink, Robert Elsaesser, and Tom Friedetzky. Efficient Randomised Broadcasting in Random Regular Networks with Applications in Peer-to-peer Systems. In *Proceedings of the Twenty-seventh ACM Symposium on Principles of Distributed Computing*, PODC '08, pages 155–164, New York, NY, USA, 2008. ACM. doi:10.1145/1400751.1400773.

7 Petra Berenbrink, Robert Elsässer, and Thomas Sauerwald. Communication Complexity of Quasirandom Rumor Spreading. In *Proceedings of the 18th Annual European Conference on Algorithms: Part I*, ESA'10, pages 134–145, Berlin, Heidelberg, 2010. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=1888935.1888952.

8 Petra Berenbrink, Robert Elsässer, and Thomas Sauerwald. Randomised Broadcasting: Memory vs. Randomness. *Theoretical Computer Science*, 520:306–319, April 2010. doi:10.1007/978-3-642-12200-2_28.

9 Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal Multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, May 1999. doi:10.1145/312203.312207.

10 Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: Byzantine resilient random membership sampling. *Computer Networks*, 53(13):2340–2359, 2009. Gossiping in Distributed Systems. doi:10.1016/j.comnet.2009.03.008.

11 Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication Locality in Secure Multi-party Computation. In *Theory of Cryptography*, 2013.

12 Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

13 Gabriel Bracha and Sam Toueg. Asynchronous Consensus and Broadcast Protocols. *JACM*, 32(4), 1985.

14 Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to Reliable and Secure Distributed Programming.* Springer Publishing Company, Incorporated, 2nd edition, 2011.

15 Christian Cachin and Jonathan A. Poritz. Secure Intrusion-tolerant Replication on the Internet. In *DSN*, 2002.

16 Nishanth Chandran, Wutichai Chongchitmate, Juan A. Garay, Shafi Goldwasser, Rafail Ostrovsky, and Vassilis Zikas. The Hidden Graph Model: Communication Locality and Optimal Resiliency with Adaptive Faults. In *ITCS '15*, 2015.

17 Fan Chung and Linyuan Lu. The Diameter of Sparse Random Graphs. *Advances in Applied Mathematics*, 26:257–279, 2001.

18 Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-generation Onion Router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association. URL: http://dl.acm.org/citation.cfm?id=1251375.1251396.

19 Sisi Duan, Michael K. Reiter, and Haibin Zhang. BEAT: Asynchronous BFT Made Practical. In *CCS*, 2018.

20 Robert Elsässer and Dominik Kaaser. On the Influence of Graph Density on Randomized Gossiping. *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 521–531, 2015.

21 Paul Erdös and Alfréd Rényi. On Random Graphs. *Publicationes Mathematicae*, 6:290–297, 1959.

22 P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight Probabilistic Broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, November 2003. doi:10.1145/945506.945507.

23 Yaacov Fernandess, Antonio Fernández, and Maxime Monod. A Generic Theoretical Framework for Modeling Gossip-based Algorithms. *SIGOPS Oper. Syst. Rev.*, 41(5):19–27, October 2007. doi:10.1145/1317379.1317384.

24 Juan Garay, Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. The price of low communication in secure multi-party computation. In *Annual International Cryptology Conference*, pages 420–446. Springer, 2017.

25 Juan A Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively Secure Broadcast, Revisited. In *PODC*, pages 179–186, 2011.

26 Chryssis Georgiou, Seth Gilbert, Rachid Guerraoui, and Dariusz R. Kowalski. On the Complexity of Asynchronous Gossip. In *Proceedings of the Twenty-seventh ACM Symposium on Principles of Distributed Computing*, PODC '08, pages 135–144, New York, NY, USA, 2008. ACM. doi:10.1145/1400751.1400771.

27 Chryssis Georgiou, Seth Gilbert, Rachid Guerraoui, and Dariusz R. Kowalski. Asynchronous Gossip. *J. ACM*, 60(2):11:1–11:42, May 2013. doi:10.1145/2450142.2450147.

28 Chryssis Georgiou, Seth Gilbert, and Dariusz R. Kowalski. Meeting the deadline: on the complexity of fault-tolerant continuous gossip. *Distributed Computing*, 24(5):223–244, December 2011. doi:10.1007/s00446-011-0144-6.

29 Mohsen Ghaffari and Merav Parter. A Polylogarithmic Gossip Algorithm for Plurality Consensus. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 117–126, New York, NY, USA, 2016. ACM. doi:10.1145/2933057.2933097.

30 George Giakkoupis, Yasamin Nazari, and Philipp Woelfel. How Asynchrony Affects Rumor Spreading Time. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 185–194, New York, NY, USA, 2016. ACM. doi:10.1145/2933057.2933117.

31 Rachid Guerraoui, Florian Huc, and Anne-Marie Kermarrec. Highly dynamic distributed computing with byzantine failures. In *PODC*, 2013. URL: http://dl.acm.org/citation.cfm?doid=2484239.2484263.

32 Rachid Guerraoui, Anne-Marie Kermarrec, Matej Pavlovic, and Dragos-Adrian Seredinschi. Atum: Scalable Group Communication Using Volatile Groups. In *Proceedings of the 17th International Middleware Conference*, Middleware '16, pages 19:1–19:14, New York, NY, USA, 2016. ACM. doi:10.1145/2988336.2988356.

33 Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos Seredinschi. The Consensus Number of a Cryptocurrency. In *PODC*, 2019.

34 Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. Scalable Byzantine Reliable Broadcast (Extended Version). *arXiv preprint arXiv:1908.01738, Version 1*, 2019. arXiv:1908.01738v1.

35 Vassos Hadzilacos and Sam Toueg. Fault-tolerant broadcasts and related problems. In Sape J. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 1993.

36 Bernhard Haeupler, Gopal Pandurangan, David Peleg, Rajmohan Rajaraman, and Zhifeng Sun. Discovery Through Gossip. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12, pages 140–149, New York, NY, USA, 2012. ACM. doi:10.1145/2312005.2312031.

37 Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-Man: Gossip-based Fast Overlay Topology Construction. *Comput. Netw.*, 53(13):2321–2339, August 2009. doi:10.1016/j.comnet.2009.03.013.

38 Valerie King, Steven Lonargan, Jared Saia, and Amitabh Trehan. Load Balanced Scalable Byzantine Agreement through Quorum Building, with Full Information. In *International Conference on Distributed Computing and Networking*, pages 203–214. Springer, 2011.

39 Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable Leader Election. In *SODA*, 2006.

40 Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *TOPLAS*, 4(3), 1982.

41 Meng-Jang Lin, Keith Marzullo, and Stefano Masini. Gossip Versus Deterministically Constrained Flooding on Small Networks. In *Proceedings of the 14th International Conference on Distributed Computing*, DISC '00, pages 253–267, London, UK, UK, 2000. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=645957.675970.

42 Dahlia Malkhi, Michael Merritt, and Ohad Rodeh. Secure Reliable Multicast Protocols in a WAN. In *ICDCS*, 1997.

43 Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 569–578. ACM, 1997.

44 Dahlia Malkhi and Michael K. Reiter. A High-Throughput Secure Reliable Multicast Protocol. In *CSFW*, 1996.

45 Dahlia Malkhi and Michael K. Reiter. A High-Throughput Secure Reliable Multicast Protocol. *Journal of Computer Security*, 5(2):113–128, 1997. doi:10.3233/JCS-1997-5203.

46 Dahlia Malkhi, Michael K Reiter, Avishai Wool, and Rebecca N Wright. Probabilistic Quorum Systems. *Inf. Comput.*, 170(2):184–206, November 2001. doi:10.1006/inco.2001.3054.

47 Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

48 Fernando Pedone and André Schiper. Handling message semantics with generic broadcast protocols. *Distributed Computing*, 15(2):97–107, 2002.

49 Michael K. Reiter. Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart. In *CCS*, 1994.

50 Michael K. Reiter and Kenneth P. Birman. How to securely replicate services. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3), 1994.

51 Christian Scheideler. How to Spread Adversarial Nodes? Rotate! In *STOC*, pages 704–713. ACM, 2005.

52 Suman Sourav, Peter Robinson, and Seth Gilbert. Slow Links, Fast Links, and the Cost of Gossip. *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 786–796, 2018.

53 Sam Toueg. Randomized Byzantine Agreements. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, pages 163–178, New York, NY, USA, 1984. ACM. doi:10.1145/800222.806744.

54 Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 137–152, New York, NY, USA, 2015. ACM. doi:10.1145/2815400.2815417.

55 Spyros Voulgaris, Márk Jelasity, and Maarten van Steen. A Robust and Scalable Peer-to-peer Gossiping Protocol. In *Proceedings of the Second International Conference on Agents and Peer-to-Peer Computing*, AP2PC'03, pages 47–58, Berlin, Heidelberg, 2004. Springer-Verlag. doi:10.1007/978-3-540-25840-7_6.

56 Marko Vukolic. The Origin of Quorum Systems. *Bulletin of the EATCS*, 101:125–147, 2010. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/183.

57 B. Zhang, K. Han, B. Ravindran, and E. D. Jensen. RTQG: Real-Time Quorum-based Gossip Protocol for Unreliable Networks. In *2008 Third International Conference on Availability, Reliability and Security*, pages 564–571, March 2008. doi:10.1109/ARES.2008.139.