Towards Self-Stabilizing Byzantine Fault Tolerance Based on Randomization: Overview and Challenges

1 Introduction

For over a decade the rising distributed ledger technology (with blockchain being the most prominent of its flavors) has breathed new interest into Byzantine fault tolerance (BFT) research [1, 4, 34, 60, 86, 115, 117]. Especially the blockchain "rush" of the past recent years has driven researchers to invest effort into more time, communication, energy-efficient distributed ledger solutions. Expectations are still high for the future of this technology and the research field is mature but still far from being saturated. Given that BFT is in the heart of distributed ledger protocols, the attested rejuvenation of BFT research is no stumper. Whilst several Byzantine consensus approaches like proof of work, or proof of stake have appeared as novel in the field, past experience in the Byzantine-tolerant state machine replication paradigm was also employed in ongoing research.

Solutions following the more "traditional" BFT solutions have leveraged on both deterministic and randomized solutions. The former depend on timing assumptions [3, 20, 36, 72, 117] and failure detection, and the latter on randomization, both with the aim to bypass the landmark FLP impossibility result, which shows that agreement in asynchrony is impossible in the presence of even a single failure [63]. Appending the blocks of a blockchain in a commonly accepted order is equivalent to the task of BFT state machine replication [1, 115], and the blockchain protocols employed are based on solving consensus or atomic broadcast.

In a parallel line of work, some solutions seek to endow BFT with the fault tolerance property of self-stabilization. This property enables the automatic recovery of a system from any "illegal" (unanticipated) system state back to a state where it achieves its designed functionality [54]. Existing self-stabilizing BFT protocols are deterministic, based either on timing assumptions [21] or on failure detection [55]. We look into existing work on asynchronous BFT solutions employing randomization, as well as asynchronous randomized Byzantine consensus and atomic broadcast as these can be leveraged towards state machine replication. We also highlight potential challenges that could arise while constructing a self-stabilizing BFT solution based on randomization.

2 State Machine Replication

Fault-tolerance is a vital system property [62, 70, 82, 104]. This is not surprising since system failures, whatever their source might be (power failures, hardware failures, communication interruptions, DoS and other cyber-attacks), have already had severe repercussions on industries, services, and governments. The commodity of fault-tolerance has thus seen major day-to-day gains [23, 24, 92] over the past decades. A system designer who ignores today's state-of-the-art in fault tolerance is paving the way to utter failure. A standard fault tolerance technique is redundancy.

Building systems that cater for redundancy, namely keeping multiple copies of a distributed object, renders systems more robust by masking replica (or processor) failures and provides availability and performance in the case of queries [46, 62, 111]. Nevertheless, redundancy is not a free desert. Using it gives rise to an important problem; that of consistency: How can the system retain consistency of state in every single replica when the distributed object is being changed possibly concurrently? The user of a distributed system must be able to receive a consistent, up-to-date view of the distributed object it is querying or processing.

Replication [39] is a well-known and widely studied paradigm in distributed computing for creating multiple copies of a (possibly) dynamic object. The aim is to ensure that redundancy is leveraged towards masking failures and providing availability, but not at the expense of the reliability of data, while also attempting to preserve performance during replica updates [104]. There are several approaches, and, depending on the task, one may choose to relax the need for strong consistency, for example, to achieve better performance and availability [22].

Several methods have been suggested over the years to retain the consistency of replicated objects. State machine replication (SMR) [73, 106] is one of them. It emulates the state transitions of one replica—possibly of a leader also known as primary or coordinator—in every other replica in the system through the execution of the same commands, with the same input, on the same state variables, and in the same order. There is a very long list of research papers on the topic from many years back. In the last few years, though, a flavor of SMR tolerating malicious entities (also known as Byzantine Fault Tolerance - BFT) became a hot topic. The reason? Blockchain. We will study the blockchain connection later in our survey.

3 Byzantine Fault Tolerance

The most severe type of failure in distributed computation is the arbitrary or "Byzantine" one [76], in which some processors may act arbitrarily by not following the defined algorithm. This model abstracts the "intentions" of the faulty components of the system. It includes both malicious agent actions, and non-intentional faults like occasional communication failures causing packet corruption of outgoing data, or a flickery power supply [111]. State machine replication in the presence of arbitrary faults is a long-studied problem [36, 45] and solutions of it come under the name of Byzantine Fault Tolerance (BFT). The research area is defined by two main impossibility results. The FLP [63] impossibility suggesting that asynchronous agreement is impossible if a single failure is allowed, and Lamport et al.'s result [76] that agreement is achievable (only in synchrony) if the processors that can act arbitrarily constitute less than a third of the processors' set.

Since then, several ways have been proposed to bypass the impossibilities for the general case [45]. Popular ways to do this, are to introduce some synchronism to the model and assume a known delay to the system's communication. Another way is to employ a failure detection mechanism [37], while some works move on to modify the problem and ask for weaker guarantees. Finally, a different approach is to relax the requirement for a deterministic solution and use randomization [9], a direction that we will further elaborate on in the sequel.

A seminal paper on BFT replication is Practical Byzantine Fault Tolerance (PBFT) by Castro and Liskov [36]. Its main assumption to achieve Byzantine-tolerant replication is that the number of Byzantine processors, do not exceed one third of the complete processor cohort. Further, it requires a consistent initial state and an unbounded local memory and access to cryptographic mechanisms to ensure that messages are not forged. Do note that if any of these assumptions fail and lead to corruption of the local state, then the system is likely to malfunction and fail to provide its safety guarantees. A series of similar works either build on PBFT like the one by Cachin et al. [29], Zyzzyva [72], the Spinning algorithm with a rotating primary [114], BFT-SMART [20], and ABSTRACT [11].

Two directions that can lead to BFT are to solve either Byzantine agreement/consensus or Byzantine atomic broadcast. The first solves the problem of a set of correct processors agreeing (in the presence of malicious entities) on a single value. This can be extended to agreement on several values and further on a common order of delivery of several messages. The second approach is a message delivery protocol guaranteeing that the same set of messages is either delivered to any correct processor in the system in the same order or it is not delivered at all. The two approaches were shown to be equivalent [37, 87]. Because the FLP result was shown to hold for the Byzantine agreement, then due to the equivalence the impossibility also holds for Byzantine atomic broadcast. Examples of papers tackling Byzantine consensus is Byzantine Paxos [75] and Fast Byzantine Consensus [84]. A survey on the topic can be found in [45]. Byzantine atomic broadcast is a well-researched topic [49, 102]. It is commonplace to build atomic broadcast on top of a multi-value consensus protocol [43].

Distributed ledgers, a form of which is the blockchain, require some way of agreeing on a common history of transactions (these being the blocks added to the blockchain) [1, 67, 98, 105, 115] [15, 16, 17, 18, 19]. An efficient, highly fault-tolerant SMR service can achieve this since it can leverage high availability and a high volume of transactions. A blockchain system with low throughput and slow transaction processing is doomed to be annihilated by high competition.

3.1 BFT and blockchain

Currently, the Bitcoin cryptocurrency caters for (a very limited number of) 7 transactions per second. At the same time the annual computing power to mine Bitcoin corresponds to a whopping 33.5 metric tons of carbon dioxide equivalent [91]. Mining is generating proof of work which is the mean by which the bitcoin blockchain achieves Byzantine agreement on the order of blocks. The low number of transactions per second and the high energy consumption in a society that strives to become greener have been a strong motive to research other directions. Newer cryptocurrencies and other blockchain (and in general distributed ledger) technologies, are seeking better, more economic, eco-friendly and scalable ways to build an immutable transaction log [79, 86, 115]. BFT solutions are promising in providing this service.

BFT consensus is an integral part of these advances [1, 42] as explained before. BFT is employed by the Hyperledger Fabric [30, 115]. Specifically, it uses the BFT-SMART [20] implementation. HoneyBadgerBFT [86] guarantees tolerance of up to a third of malicious processors and a throughput of tens of thousands of transactions per second in an asynchronous setting by building a novel atomic broadcast algorithm. Hybrid approaches of BFT and of simpler crash-tolerant algorithms have emerged to improve performance. Examples of this approach are XFT [79] and Tendermint [27]. The topic is hot and moving fast, driven both by academic and industrial research [38, 39]. To our knowledge, most BFT-based protocols assume some kind of assumptions on the synchrony of the system, beyond the standard liveness assumption that messages are eventually delivered. Only those protocols that employ randomization do not need additional synchrony assumptions. We review several works that require timing assumptions and look more carefully at solutions employing randomization in the next subsection.

Hotstuff [117] is a BFT protocol for the partial synchrony model. It is proposed to be the first such algorithm that is also linear in communication complexity and can achieve consensus within the actual network delay (and not the maximum). The protocol proceeds in views (a monotonically increasing counter) and each view has a leader. The state comprises of a tree whose nodes store the client's requests and other data related to the algorithm. Branches of the tree become committed when the leader collects votes from a quorum of $n - f$ processors approving the three phases of committing. A threshold signature scheme generates the signed votes of the processors to generate an *authenticator* of the commit decision. The authors then proceed to an interesting simplification of the first protocol to provide Chained HotStuff. In

this version of the algorithm it is noted that the initial protocol had three phases that where similar. They, thus, present a single generic view to replace these, where the second phase of a client request is actually relayed to the processor of the next view.

SBFT [66] is a BFT service optimized for the executes of Ethereum EVM smart contracts of hundreds of of replicas. It employs a message collector processor and threshold signatures to reduce the communication cost. Asynchrony is assumed, but synchrony (distinguished as two different synchrony modes) is required for liveness. The protocol has three phases (following the PBFT paradigm) and uses a primary to proceed replication. Each view has different primary who collects requests from clients and bundles them and sends them to the other replicas. Two rounds of signing requests (with threshold signatures) and sending them to a collector replica are performed before the request is executed and the client is informed. View changes take place upon timer expire or if the primary is shown as malicious by publicly verifiable proof. The evaluation shows improvements over PBFT, and that optimizations induced on SBFT do indeed have effect. Finally, experiments on real-life Ethereum smart contract executions are suggested to gain a tenfold speed-up (in transactions per second).

Tendermint [27] is another round-based Byzantine consensus algorithm for the partially synchronous model. It uses a gossip protocol for inter-process communication, and digital signatures for messages, such us a processor can know who was the original sender of a message forwarded by any other processor. SMR takes place by sequential instances of "validity predicate-based" consensus to decide upon each block. This consensus protocol follows the usual agreement and termination properties that any two correct properties decide on the same value and that all correct processes eventually decide. A third property, validity, holds when a given predicate $valid()$ is true. The protocol, rather than considering n to be the number of processors, takes n to be the total voting power of processes in the system. In this sense, it is optimal as it can tolerate the combined voting power of malicious processors to be up to $n = 3f + 1$. The consensus rounds proceed, with each requiring three phases before committing a block, and blocking is prevented by using timeouts. The termination mechanism is considered as novel, and takes advantage of the gossip-based communication.

The Algorand cryptocurrency [40] uses the Byzantine agreement protocol BA*. BA* does not count a vote per processor, but requires 2/3 of some weighted user total to append a block. It is a consensus by committee protocol, where the committee is chosen randomly with some bias introduced according to the weights of the users. Committee selection is performed privately and non-interactively by local computations using cryptographic public keys and information from the blockchain itself. This approach is suggested to mitigate attacks on users before they initiate the BA* protocol. Algorand requires a series of assumptions, such as that more than 2/3 of the money in the currency is held by honest users, and for liveness it requires *strong synchrony*, namely, that messages are delivered within a known time bound. It also requires "weak synchrony" to maintain safety. In such periods of weak synchrony the system proceeds by sending tentative blocks, which are committed in periods of strong synchrony.

Placing guarantees on timing assumptions can seem reasonable if a system performs well "most times" in terms of communication. Nevertheless, approaches that can immediately progress once messages are delivered, rather than delaying even after some synchrony is established, have gained particular appeal. Creating a system that is free from synchrony assumptions appears more graceful than struggling with communication uncertainty in decentralized systems. Given the FLP impossibility, one can trade timing assumptions with non-determinism. (We note that while Algorand uses randomness to choose a group of verifier processors, its correctness depends on timing assumptions. The random component serves as a measure against the common cryptocurrency issue of concentration of voting power by a handful of users.)

3.2 Randomized BFT

As was expected the research community that tackled the BFT problem was interested in randomized solutions as well. Randomized algorithms were popularized in computer science for two generally accepted attributes: simplicity and removal of timing assumptions [101], [88, p. xiv], [97, p. ix]. For some problems, the only known polynomial time solutions are probabilistic.

3.2.1 Preliminaries

Given the FLP impossibility mentioned before, the results by Rabin [101] and Ben-Or [17] on randomized agreement were the first to use probabilistic methods to circumvent the impossibility, rather than invoking timing assumptions limitations. The above works were followed by many other papers [6, 7, 29, 32, 33, 90, 112]. The idea (as given by Ben-Or [17]) is that if a process during a round receives a majority vote on a single value, it decides in favor of this value. Otherwise, it changes its proposed value in the next round by flipping a coin. The correctness proof places special focus on proving that the algorithm will terminate with high probability; in other words, that executions that do not terminate occur as events with probability 0. Contrary to other cases, the efficiency of randomized agreement protocols has been criticized as having high communication (due to comparatively high number of communication steps) and computational complexity (due to customary employment of cryptography) [90].

Whilst being novel in using randomization to bypass FLP, Ben Or's approach induced an exponential number of rounds (unless $f = O(\sqrt{n})$. , and tolerated no more than a fifth of the processors being malicious, which is not optimal. Rabin [101] introduced a shared coin, such that a sequence of coin tosses is distributed to all processors before the agreement protocol is initiated. The algorithm assumes message authentication with digital signatures. It can tolerate $f < n/10$ of malicious processors in an asynchronous setting, but it achieves agreement within 4 rounds. Toueg [112] gave an optimal algorithm tolerating $f < n/3$ faulty processors with the same setting. In this case more rounds were required, yet these were again constant in number.

At the beginning of the millenium, Cachin et al. [33] presented SINTRA which achieved replication on Byzantine-tolerant atomic broadcast. In particular, it employs threshold cryptography for: (i) digital signatures, (ii) coin-tossing, and (iii) public-key encryption (although for (i) it is stated that standard digital signatures suffice). Based on this it implements broadcast primitives and binary agreement, which themselves are used to achieve multi-valued agreement resulting to atomic broadcast. The resulting protocol is a secure causal atomic broadcast protocol. The threshold schemes for coin tossing comes from Cachin et al. [32] where Byzantine agreement is achieved in constant time and the protocol is optimal in the number of malicious processors that it can tolerate. In this parallel work, a trusted dealer is required in an initial setup phase. From then on, the system can implement a transaction processing service that is considered to suffice for an unlimited number of requests. The threshold scheme is not used directly by the agreement component, but indirectly through its employment by the coin-tossing and digital signatures schemes.

In a very recent work by Abraham et al. [2], the authors solve asynchronous multi-valued Byzantine agreement. Validity is given by an external validity function (as in the case of Tendermint and in a trend following [31]), thus the problem solved it tipped a Validated Asynchronous Byzantine Agreement (VABA). Previous results [32] gave reductions of VABA from binary agreement (which is agreement on a single value from 2 possible values rather than from multiple values as is the case of VABA). Abraham et al. take a different approach through a view-change-based solution (much used in partially synchronous algorithms). It avoids binary agreement by running n threads of leader election and then randomly electing a leader retrospectively. Timeouts (much needed in most partially synchronous protocols) are replaced by the knowledge

of progress of $n - f$ leader-election threads. This VABA solution also uses cryptographic abstractions such as threshold signatures and threshold coin-tossing. The protocol is secure against an adaptive adversary, and optimal in resilience (tolerating $3f + 1$) and in word communication with $O(n^2)$ messages (where a message is roughly the size of one or two threshold signatures).

In another recent work, Cohen et al. [41] provide the first, as suggested, sub-quadratic binary Byzantine agreement algorithm. The authors employ public-key cryptography to build a shared coin and use verifiable random functions. The sub-quadratic word complexity comes at the cost of not having optimal resilience ($n \approx 4.5f$), and also having both safety and termination guaranteed with high probability.

3.2.2 Blockchain-oriented randomized BFT

Because of increased interest on BFT for distributed ledgers many BFT protocols have appeared. Some of these employ randomization.

HoneyBadgerBFT [86]: This was the first work on a randomized BFT solution that was leveraged towards blockchains. The paper emphasizes its decision to focus on randomization rather than placing timing assumption/failure detectors to circumvent the FLP result. To this end, it provides specific examples of executions where PBFT-like protocols requiring some weak synchrony assumption may fail to progress whilst in the same scenarios asynchronous protocol can progress. Moreover, it is suggested that randomized BFT protocols can recover and progress faster than their weak-synchrony counterparts when messages are delivered. The HoneyBadger service is proposed as the first BFT atomic broadcast protocol that provides optimal asymptotic efficiency in the asynchronous setting. Structurally it follows SINTRA. The two main lines of improvements done on SINTRA are: (a) removal of redundant work among processors by ensuring fairness through the use of threshold public encryption, and (b) removal of suboptimal instantiation of the Asynchronous Common Subset (ACS) module.

The ACS essentially lets processors decide on a common subset of requests. Honeybadger's improvement is achieved through the use of erasure codes and by the reduction of ACS to reliable broadcast rather than multi-value validated Byzantine agreement. The reduction of ACS to reliable broadcast is due to Ben Or et al. [17]. The ACS also requires binary agreement, and it uses the one by Mostefaoui et al. [94] which uses a cryptographic common coin. The reliable broadcast algorithm is due to Bracha [25] combined with erasure codes to enhance efficiency. Honeybadger employs cryptography both for the common coin and to for the threshold encryption scheme that it uses in the top-level Honeybadger protocol. It assumes a trusted dealer that delivers public keys and secret shares in an initial setup phase. HoneyBadger's analysis shows an $O(N)$ communication bit/transaction complexity on its BAB protocol which is better than all previous protocols, although the total communication complexity is the same as PBFT. The experimental analysis compares Honeybadger with PBFT and shows improvements in the number of transactions per second. It is clearly stated that Honeybadger is targeted towards scenarios where computational power is ample and communication throughput is the scarce resource, since this is regarded as the standard case for blockchain settings. Honeybadger's implementation uses 80-bit protocol is used instead of the standard 128-bit [60].

BEAT [60] harnesses Honeybadger's modularity by specifying alternatives for some of Honeybadger's modules (threshold encryption scheme, information dispersal service, etc.). In this way it presents 3 variant protocols that are suggested to outperform Honeybadger, whilst also providing optimizations. In particular, BEAT-0 changes the threshold encryption (which is acclaimed to be more secure and efficient than Honeybadger's), the coin flipping mechanism and the erasure coding support. BEAT-1 uses the plain Bracha's protocol [25] rather than the erasure-coded version of HoneyBadger. This provides an advantage in cases of low contention and batch size. BEAT-2 pushes the cost of threshold encryption to the client side reducing latency (with weaker liveness guarantees), whilst also achieving causal ordering. BEAT- 3 and BEAT-4

are presented as "BFT storage systems" replacing reliable broadcast with asynchronous verifiable information dispersal based on fingerprinted cross-checksum. The authors differentiate between BFT storage (only implementing reads and writes) from BFT SMR that can support arbitrary operations. BEAT's suite of protocols targets a more per setting/per application mentality rather than a one-size-fits-all approach. Their experimental results reflect this approach with the expected improvements appearing in the corresponding experiments (whilst differentiating batch size, contention, node numbers). Nevertheless, the improvements are not always spectacular, especially as the number of malicious processors is increased.

Hashgraph [13] departs from the Honeybadger/BEAT approach where there are two protocols one for broadcast and one for voting. Instead they propose an algorithm that incorporates both. In particular, communication takes place in a gossip manner. A processor p randomly selects another processor q, to whom it communicates any information that p considers as unknown to q. Such information is stored in a data structure called a "hashgraph". This allows for a unique protocol incorporating both communication and virtual voting. The information on the local hashgraph information allows processors to divide the requests into decision rounds, based on how strongly the information gossiped appears among the processors in the hashgraph. The protocol again uses public key cryptography.

EPIC [78] focuses on tackling an adaptive adversary, in the sense that the set of f processors that may act maliciously during the system's lifetime may be different at any instance of the computation. This is said to differ from the static adversary tackled by HoneyBadger and BEAT where the set of malicious processors is static during the computation. With this focus, EPIC also changes modules of HoneyBadger and tests the impact of these changes.

Whilst the probabilistic approach bears many benefits, it also has its downsides. The fact that determinism is lost, implies that the algorithm's verification and analysis should now combine both the (deterministically-orientated) assertion-based reasoning, as well as probabilistic analysis [64, 65]. More importantly randomized algorithms will guarantee the correctness of the protocol but will strive to provide guarantees on whether it terminates or not. Termination is guaranteed with high probability, but it is not certain as is the case with deterministic algorithms [10]. Bypassing cryptography: While cryptographic paradigms constitute a main tool in most BFT solutions, there are several claims that cryptography is expensive computationally. A line of work striving to achieve signature-free BFT is an alternative to the aforementioned norm. Ousting cryptography is bound to generate increased burdens on communication with more messages required to compensate the loss of guarantees that cryptography provided. Nonetheless, the trade-off may in some cases be well worth it, and only the setting and the application can flip the coin towards cryptography or not. We note here that the signature-free protocols still require basic authentication to ensure that a message received by a processor p from the common channel with some processor q, was not fed into the channel by some third processor.

3.2.3 Signature-free Randomized Consensus

As already mentioned, many works tackling Byzantine asynchronous SMR do this on top of atomic broadcast and/or consensus protocols. It is customary for these to be based on binary agreement. A specific group of BFT/consensus/atomic broadcast protocols avoids the use of computationally expensive encryption methods (such as the commonly used threshold signatures) [103]. More specifically, it is (often) assumed that authenticated communication links exist (i.e., a processor p_i with a bidirectional channel with another processor p_j, will receive messages guaranteed to originate from p_j not from another processor), but no further cryptographic abstractions are used. For example a message from p_i forwarded from p_j to processor p_k cannot be confirmed either as originating from p_i, or having remained unaltered. This approach, while avoiding the computational burden of added cryptography, nevertheless, incurs extra costs on message exchanges in

order to ensure the validity of incoming messages.

Randomized Byzantine binary consensus. The already mentioned work by Ben-Or [17] does not employ advanced cryptographic paradigms. On the other side, it can take an exponential expected number of round to terminate, since it only requires a *local coin*, i.e., one that provides a processor with a random value independent of corresponding values given to other processors. *Common coins* on the other hand can overcome this inefficiency, and require only a constant number of expected rounds. Probably the first work using a common coin for randomized binary consensus (a basic building block for SMR in many cases) was by Rabin [101]. This though required signatures and was not optimal in resilience ($f < 10/n$), but had $O(n^2)$ message complexity and constant number of expected rounds. Signature-free solutions like the ones of Berman and Garay [19] and Friedman et al. [64] also have a message complexity of $O(n^2)$, but are again not optimally resilient, namely $f < 5/n$.

Many of the early common coins required a *trusted dealer*, which, a priori, provides the participating processors with a sequence of bits, in a setup phase. This trusted dealer assumption can be bypassed by using *distributed key generation* like the one of [71], which implements an eventually perfect common coin. Nevertheless, it is not known if a common coin can be obtained in a signature-free way in the asynchronous setting. An interesting experimental comparison between local and common coins can be found in [89]. The experiments showed that the local coin managed a very good performance given the exponential theoretical bound on the rounds, and managed to surpass the computationally "heavier" shared coin in many cases. Yet more experiments will be needed to give decisive results.

Bracha [26] on one hand and Srikanth and Toueg [110] published in the same year (1987) probably the first signature-free optimal resilience binary Byzantine consensus protocols. A signature-free asynchronous binary consensus protocol with optimal resilience is also given by Correia et al. [44] as an appendix, since it is required for their atomic broadcast protocol. Mostefaoui et al. [94] provide an asynchronous binary consensus (which for the first time) achieves optimal resilience, quadratic message complexity and constant expected time, and also constant complexity of message size. The binary consensus algorithm sits on top of a "double-synchronized binary-value broadcast", which is constructed from basic broadcast primitives. The overall idea of this broadcast protocol, is that instead of tracking on which processor broadcast a specific value, it only considers how many distinct processes have broadcast this same value. Moreover, it ensures that a value delivered by a correct process, was not broadcast by only malicious processes. The consensus algorithm uses a common coin. In contrast to a first version of the algorithm [93], which required a perfect common coin and did not accept message reordering, this version only requires a weak coin and can tolerate message reordering. Tyler [47] experimentally evaluated this work, and has also proposed two alternative algorithms with the same qualities [48]. In a previous work, Mostefaoui and Raynal [95] propose a randomized binary consensus algorithm that is based on validated broadcast, which is more expensive communication-wise than reliable broadcast.

Building signature-free SMR and atomic broadcast on top of binary consensus. It is common to construct atomic broadcast services modularly on top of multi-valued consensus and/or vector consensus, themselves using a binary consensus protocol. Song and Van Renesse [109] propose a multi-valued consensus protocol that can call any binary consensus routine and "strongly" completes in one step, where the "strongly one-step" execution is rigorously defined in the paper to be distinct from similar metrics of other papers. The solution given is not optimal in resilience. Mostefaoui and Raynal [96] suggest another optimal resilience signature-free reduction of multi-valued consensus to binary consensus.

Milosevic et al. [87] elaborate on the actual theory of reducing atomic broadcast to consensus. They

consider four different versions of the validity property of consensus and prove how each one of these is equivalent, weaker or stronger to atomic broadcast. They also provide a reduction of their own. The Cachin et al. [31] in SINTRA employ signatures to achieve atomic broadcast. Liang and Vaidya [77] consider a synchronous time model. Patra et al. [99] presented an optimal resilience algorithm that is based on an common coin scheme suggested to be more efficient than previous ones.

Correia et al. [44] provide a signature-free, asynchronous, leaderless, and optimal in resilience protocol (i.e., $n = 3f + 1$) reducing atomic broadcast to vector-consensus and subsequently to multi-valued and then to binary consensus. Bottom-up, the reliable broadcast protocol is assumed as existing. A binary consensus algorithm is given as an appendix of the paper. It is a signature-free optimal resilience version of Bracha's algorithm [25], although it is noted that the coin-tossing scheme used (which is the one given by Cachin et al. [32]) does not fully avoid cryptography since it is based the Diffie-Hellman problem.

The paper focus of the paper is on the reduction of atomic broadcast to multi-valued consensus. We briefly describe the protocol, since it is indicative of how such protocols are structured. The multi-valued consensus algorithm starts with an initialization phase and a reliable broadcast phase where processors send their proposed value. Once $n - f$ proposals have been received, if there exists a unique value supported by $n - 2f$ processors this is broadcast again, otherwise $\perp$ is sent. Provided on whether or not a value common to $n - f$ processors was delivered, a binary value of 1 or 0 is agreed upon by calling the binary broadcast protocol. The algorithm returns $\perp$ if binary consensus returns 0, or otherwise (binary consensus returns 1) the processor waits until at least $n - 2f$ messages are received that support a specific value.

Vector consensus is agreement on a single vector of values rather than on a single value [58]. The guarantees provided beyond agreement and eventual termination, concern validity: i.e., the vector of every correct processor p_i must (i) have at least $f + 1$ values proposed by correct processors, and (ii) the value $V_i[j]$ of the vector that corresponds to correct processor p_j's value as delivered by p_i must be either p_j's true value or $\perp$. The main idea is that as messages responding to an initial call for vector values accumulate, repeating calls to multi-valued consensus are performed with the vector as the value. When the multi-valued consensus returns a non-$\perp$ value (i.e., an agreed upon vector) then this is returned as the agreed vector.

Atomic broadcast sits on top of vector consensus (although it could be build straight on top of multi-valued consensus, but still the vector consensus functionality would have to appear). After the initialization phase, whenever there are pending messages to be atomically broadcast, the protocol starts with a reliable broadcast of the message and of the message number (which is a monotonically increasing integer counter). Then the vector consensus algorithm is called with the set of hashes of the messages to be delivered. Once all the messages that appear $f + 1$ times in the agreed vector have been received locally, they are atomically delivered per the agreed order. The remaining messages are left for the next round of atomic broadcast, where rounds are again counted with an integer counter.

RITAS [90] is a stack of protocols that provide randomized Byzantine agreement. It is mostly based on the signature-free protocols of Correia et al. [44]. In particular, the protocols of multivalued and vector consensus are from [44], and minor modifications allow for atomic broadcast to be based directly on atomic broadcast and not on vector consensus. RITAS provides implementations of these protocols. It also proposes a reliable broadcast and an adapted binary consensus protocol from [26]. As such, it theoretically terminates in an exponential number of communication steps [18, 26].

As the plethora of BFT papers from the past and the present, BFT (deterministic or randomized) is a well-established and enduring replication technique. It also offers a wide scope for the interested researcher. Nevertheless, BFT guarantees tolerance against malicious behavior only when the system's assumptions are obeyed. We turn towards another notion in fault tolerance, which aims at recovering the intended behavior of an algorithm when the system's assumptions are violated.

4 Self-stabilization

4.1 Introduction

Fault-tolerant systems, let them be distributed ledger infrastructure, cloud-based systems, databases, or other, will provide fault-tolerance guarantees only if specific assumptions hold. It is possible that in some instances system assumptions are violated. For example, a system might be tolerant to some number of failures of its processing entities, or to less than a third of those exhibiting malicious behavior. This is a rather most of the times uncontrollable. The same holds for assumptions on bounded churn rates, i.e., bounded rates of processors' joins and leaves/crashes. Another example is systems that offer guarantees "with high probability", e.g., error detection.

While such assumptions may be sufficient for long periods of a system's lifetime, some rare violations cannot be completely ruled out. Albeit, each of these sporadic instances may have detrimental effects, occasionally lethal [118]. (Note that their "extreme" rarity is actually disputed [59], i.e., they might occur more often than their theoretical estimates.) For example, a soft error (some accidental bit-flip) may force a counter to acquire its maximal value, and thus drive the system to either non-progress or to a permanent violation of the system's safety properties. A corrupt program counter or program variable can bring the system to an arbitrary state from which it cannot recover, since it was not anticipated by the system's designers [24, 118]. The system remains useless, requiring human intervention to recover and personnel to always be on-call. Self-stabilizing systems [52, 54, 107] are designed to automatically recover the system back to its working state and desired behavior, from any given state it may end up to after an unanticipated failure. Such systems have a comprehensive approach towards faulty system states. These are states that system designers usually consider as impossible to reach or fail to consider at all. In this way, self-stabilizing systems guarantee convergence to a legitimate system state starting from any possible system state, and closure when this legitimate state has been reached, and until the guarantees of the system are violated again [8].

Impressively, this approach can even cope with the standard designers' assumption that the system and its variables are initiated to a consistent system state. In general, automatic recovery reduces the cost of recovery, and most possibly the off-time, making systems more available, and provides added-value as far as their maintainability is concerned [83]. The self-stabilization research community has produced many results for a plethora of problems that are fundamental to distributed computing. SMR, as might be expected, has drawn the attention of many members of the self-stabilization community that aim to enhance existing solutions with the stabilization property. In the sequel, we will review related work.

For completeness, we add a couple of lines on the historical background of self-stabilization: Edsger Dijkstra [52], [50, p. 313] was the first to identify the self-stabilization property for distributed algorithms while solving the mutual exclusion task. Leslie Lamport [74] commenting on Dijkstra's paper, held it to be Dijkstra's "most brilliant work", and "a milestone in work on fault tolerance", as well as a "very fertile field for research". Lamport's characterization drew attention to self-stabilization and the distributed computing community has seen a mounting number of published research papers on self-stabilization (including an annual conference dedicated to self-stabilization). Many fundamental protocols that enable the existence of the internet (e.g., routing protocols) are self-stabilizing [35, 38, 100].

4.2 Self-Stabilizing BFT

The few attempts to tackle the problem of self-stabilizing BFT are very recent. Binun et al. [21] assume a semi-synchronous setting to bypass the FLP impossibility, i.e., the system's lower and upper message delivery delays are known. The algorithm has optimal resilience with $n = 3f + 1$. They employ the

self-stabilizing Byzantine-tolerant clock synchronization algorithm from [57] to enforce a new instance of Byzantine agreement upon every clock pulse. The algorithm requires that n instances of BFT are initiated in order to reach to an agreed vector of histories. Once this history is received, the changes instructed by the requests in the history are applied to the replica. The BFT version avoids the use of private/public key cryptography by using a rotating leader and information secure methods.

A second work by Dolev et al. [55], follows the PBFT protocol [36]. The setting is asynchronous and the algorithm uses failure detection to monitor whether the leader is acting maliciously. A major challenge in self-stabilizing BFT is that a processor which is in an arbitrary state, cannot make apart between corruption caused due to this arbitrary state and malicious messages from malicious agents. This is especially challenging.

The algorithm comprises of a view change module that monitors the view of the system. The view is a bounded integer counter that corresponds to the identifier of a processor, such that the processor with the corresponding identifier is considered the leader. This module is responsible for the convergence of the processors to a unique default view in the case where there is a transient fault. It is also responsible to change the view when the failure detector requires it, i.e., either when the leader fails to process pending requests, or when it is not responsive. To this end it leverages on a Θ-failure detector that counts and compares message exchanges in order to avoid using timeouts. The replication module follows the three phase protocol of PBFT, with the additional task of monitoring for local state corruption, in which case a reset is triggered.

Ongoing work by Lundström et al. [80, 81] provides self-stabilizing reliable broadcast and binary consensus (with the "indulgence" and "zero-degradation" characteristics). These algorithms can form the basis for asynchronous randomized SMR, but they are not Byzantine tolerant.

4.3 Randomized Self-stabilization

There has been extensive work on algorithms that handle both self-stabilization and randomization. Nevertheless, a distinction must be drawn between a self-stabilizing randomized algorithm, which is an algorithm that is probabilistic (usually) towards its termination and deterministic in its convergence (e.g., [113]), and a randomized self-stabilizing algorithm. The latter is randomized towards stabilization, i.e., it convergences with some high probability. For example Herman [68] presents a token-passing algorithm in a synchronous anonymous ring that achieves convergence with probability 1 using local coin flips.

There are two definitions for a randomized or probabilistic self-stabilizing algorithm:
a) Expectation-based: An algorithm is randomized self-stabilizing for a task if starting with any system configuration and considering any fair scheduler then the algorithm will reach a safe configuration within an expected number of rounds bounded by some constant (possibly dependent of the number of the system's processors) [54, 69].
b) Probability-based: A protocol is said to be probabilistic self-stabilizing if eventually it converges to a legitimate configuration with probability 1 [68, 116]. The above definitions suggest that convergence is probabilistic.

Beauquier et al. [15] provide a complete model framework for proving the self-stabilization of randomized protocols. (The framework was used in several previous papers as well, e.g., [14]). The model clearly distinguishes between the scheduler (in this case a non-deterministic one) and the algorithm's behavior (a randomized one). The approach is demonstrated on two randomized self-stabilizing algorithms on rings, one for leader election and one on token circulation, with the protocols being optimal in space. Whilst Beauquier et al. [15] work with shared memory, Mayer et al. [85] present a similar leader election algorithm in the message-passing model.

Ben-Or et al. [18] construct a Byzantine tolerant digital clock synchronization algorithm that has optimal

(i.e., constant) convergence time and optimal resilience. They implement a common coin in a synchronous setting based on a global beat system. It is left as an open problem whether their method can be tuned to solve the problem in an asynchronous setting.

On the theoretical side, Dolev and Tzatchar [56] target self-stabilizing protocols that employ randomization to achieve convergence, and confine the use of randomization only to the period of convergence and not after this. This adaptive method is demonstrated on the token passing algorithm by Herman [68] and on the clock synchronization algorithm by Ben-Or et al. [18]. Yamashita [116] and Devismes et al. [51] suggested a way to construct probabilistic self-stabilizing algorithms from *weak*-stabilizing algorithms that guarantee stabilization only in the best case. In a recent work Volker [113] presented a scheme transforming local randomized algorithms to self-stabilizing randomized algortithms that are deterministic towards their convergence.

5 Towards a Randomized Self-Stabilizing BFT Solution - Challenges

We have already given the motivation for the development of self-stabilizing systems. Together with all the vulnerabilities that a system can have, we underline a specific assumption that may be violated specifically in a Byzantine-tolerant system: Based on a well-known impossibility result [76] that less than a third of processors may be malicious. This assumption though cannot be controlled in any way. Its violation is a transient fault and can lead the system to an arbitrary state. As such a self-stabilizing system can recover the system to an expected state. There is existing work on stabilizing consensus and agreement algorithms [5, 12, 16, 53, 108]. Nevertheless, it usually considers non-Byzantine, or synchronous environments, and whenever asynchronous Byzantine settings are considered they do not align with the suggested approach on a randomized self-stabilizing Byzantine-tolerant SMR protocol.

Main Challenges. The suggested objective must circumvent a series of impossibility results [61, 63, 76]. We note the following challenges and limitations.

1. Given that most solutions of BFT, Byzantine consensus and atomic broadcast are dependent on cryptography, with most recent solutions using threshold encryption, this is an important obstacle for self-stabilization at the current timeframe. To this point there is scant research on self-stabilizing cryptographic protocols [28].

2. Randomization requires a stream of random bits. Building a common coin with a fixed number of rounds (like in Rabin [101]) requires secret sharing or other similar methods employing cryptographic paradigms that to this moment are non-existent in a self-stabilizing form (and may indeed be proved impossible in the future). A starting point could be made by employing Ben Or's local coin approach, which, nevertheless, induces a theoretical exponential number of rounds to terminate. We have already noted that, in many cases, the difference between local and shared coin algorithms is suggested to be only theoretical and not practical (Section 3.2.3).

3. Some non-stabilizing protocols require unbounded local memory or channel capacity. This is incompatible with self-stabilization which requires bounded capacity links [54, Chapter 3.2].

4. Distinguishing malicious behavior is not easy in such a system, since corrupt values in communication links may result from either malicious processors or arbitrary faults [61]. Since the approach is randomized and not based on detecting malicious behavior, detection is focused on self-stabilization and understanding corruption of the system's state.

A final word. The current survey followed the state-of-the-art in distributed ledgers, and the revived interest in Byzantine-tolerant state machine replication. We underlined the need for a holistic fault-tolerance approach for the critical socio-economic services that these systems are projected to deploy. In one sense, this approach culminates in Byzantine-tolerant self-stabilizing protocols. Performance and ease of implementation are important issues that define the future of any such solution, with results suggesting that the additional performance cost of self-stabilization is far outweighed by the added guarantees [65]. The focus on randomized protocols, informs us of a range of existing randomized solutions that may prove suitable as a basis to produce corresponding self-stabilizing solutions.

References

[1] Ittai Abraham, Dahlia Malkhi, et al. The blockchain consensus layer and bft. *Bulletin of EATCS*, 3(123), 2017.

[2] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 337–346. ACM, 2019.

[3] Yackolley Amoussou-Guenou, Antonella Del Pozzo, Maria Potop-Butucaru, and Sara Tucci Piergiovanni. Correctness of tendermint-core blockchains. In *22nd International Conference on Principles of Distributed Systems, OPODIS 2018, December 17-19, 2018, Hong Kong, China*, pages 16:1–16:16, 2018.

[4] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolic, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*, pages 30:1–30:15, 2018.

[5] Dana Angluin, Michael J. Fischer, and Hong Jiang. Stabilizing consensus in mobile networks. In *Distributed Computing in Sensor Systems, Second IEEE International Conference, DCOSS 2006, San Francisco, CA, USA, June 18-20, 2006, Proceedings*, pages 37–50, 2006.

[6] Luciana Arantes, Roy Friedman, Olivier Marin, and Pierre Sens. Probabilistic byzantine tolerance for cloud computing. In *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, sep 2015.

[7] Luciana Arantes, Roy Friedman, Olivier Marin, and Pierre Sens. Probabilistic byzantine tolerance scheduling in hybrid cloud environments. In *Proceedings of the 18th International Conference on Distributed Computing and Networking - ICDCN '17*. ACM Press, 2017.

[8] Anish Arora and Mohamed G. Gouda. Closure and convergence: a foundation of fault-tolerant computing. *IEEE Transactions on Software Engineering*, 19(11):1015–1027, Nov 1993.

[9] James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003.

[10] James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003.

[11] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knezevic, Vivien Quéma, and Marko Vukolic. The next 700 BFT protocols. *ACM Trans. Comput. Syst.*, 32(4):12:1–12:45, 2015.

[12] Jacques M. Bahi, Mohammed Haddad, Mourad Hakem, and Hamamache Kheddouci. Self-stabilizing consensus average algorithm in distributed sensor networks. In *Transactions on Large-Scale Data- and Knowledge-Centered Systems IX*, pages 28–41. Springer Berlin Heidelberg, 2013.

[13] Leemon Baird and Atul Luykx. The hashgraph protocol: Efficient asynchronous BFT for high-throughput distributed ledgers. In *2020 International Conference on Omni-layer Intelligent Systems, COINS 2020, Barcelona, Spain, August 31 - September 2, 2020*, pages 1–7. IEEE, 2020.

[14] Joffroy Beauquier, Maria Gradinariu, and Colette Johnen. Memory space requirements for self-stabilizing leader election protocols. In Brian A. Coan and Jennifer L. Welch, editors, *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, PODC '99, Atlanta, Georgia, USA, May 3-6, 1999*, pages 199–207. ACM, 1999.

[15] Joffroy Beauquier, Maria Gradinariu, and Colette Johnen. Randomized self-stabilizing and space optimal leader election under arbitrary scheduler on rings. *Distributed Comput.*, 20(1):75–93, 2007.

[16] Luca Becchetti, Andrea E. F. Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Stabilizing consensus with many opinions. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 620–635, 2016.

[17] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 27–30, 1983.

[18] Michael Ben-Or, Danny Dolev, and Ezra N. Hoch. Fast self-stabilizing byzantine tolerant digital clock synchronization. In Rida A. Bazzi and Boaz Patt-Shamir, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Canada, August 18-21, 2008*, pages 385–394. ACM, 2008.

[19] Piotr Berman and Juan A. Garay. Randomized distributed agreement revisited. In *Digest of Papers: FTCS-23, The Twenty-Third Annual International Symposium on Fault-Tolerant Computing, Toulouse, France, June 22-24, 1993*, pages 412–419. IEEE Computer Society, 1993.

[20] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. State machine replication for the masses with BFT-SMART. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*, pages 355–362, 2014.

[21] Alexander Binun, Thierry Coupaye, Shlomi Dolev, Mohammed Kassi-Lahlou, Marc Lacoste, Alex Palesandro, Reuven Yagel, and Leonid Yankulin. Self-stabilizing byzantine-tolerant distributed replicated state machine. In *Stabilization, Safety, and Security of Distributed Systems - 18th International Symposium, SSS 2016, Lyon, France, November 7-10, 2016, Proceedings*, pages 36–53, 2016.

[22] K. Birman, D. Freedman, Q. Huang, and P. Dowell. Overcoming cap with consistent soft-state replication. *Computer*, 45(2):50–58, Feb 2012.

[23] Olivier Boireau. Securing the blockchain against hackers. *Network Security*, 2018(1):8–11, 2018.

[24] Danny Boyle and Robert C. Mendick. Worldwide airport chaos after computer check-in systems crash, 2017.

[25] Gabriel Bracha. An asynchronou [(n-1)/3]-resilient consensus protocol. In Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro, editors, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 154–162. ACM, 1984.

[26] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987.

[27] Sean Braithwaite, Ethan Buchman, Igor Konnov, Zarko Milosevic, Ilina Stoilkovska, Josef Widder, and Anca Zamfir. Tendermint blockchain synchronization: Formal specification and model checking. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part I*, volume 12476 of *Lecture Notes in Computer Science*, pages 471–488. Springer, 2020.

[28] D. Brownstein, S. Dolev, and M. V. Kumaramangalam. Self-stabilizing secure computation. *IEEE Transactions on Dependable and Secure Computing (Under publication)*, pages 1–1, 2020.

[29] Christian Cachin. State machine replication with byzantine faults. In Charron-Bost et al. [39], pages 169–184.

[30] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[31] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. *IACR Cryptol. ePrint Arch.*, 2001:6, 2001.

[32] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, may 2005.

[33] Christian Cachin and Jonathan A. Poritz. Secure intrusion-tolerant replication on the internet. In *2002 International Conference on Dependable Systems and Networks (DSN 2002), 23-26 June 2002, Bethesda, MD, USA, Proceedings*, pages 167–176. IEEE Computer Society, 2002.

[34] Christian Cachin and Marko Vukolic. Blockchain consensus protocols in the wild (keynote talk). In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 1:1–1:16, 2017.

[35] Marco Canini, Iosif Salem, Liron Schiff, Elad Michael Schiller, and Stefan Schmid. Renaissance: A self-stabilizing distributed SDN control plane. In *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*, pages 233–243, 2018.

[36] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186, 1999.

[37] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[38] Ho-Yen Chang, Shyhtsun Felix Wu, and Y. Frank Jou. Real-time protocol analysis for detecting link-state routing protocol attacks. *ACM Trans. Inf. Syst. Secur.*, 4(1):1–36, 2001.

[39] Bernadette Charron-Bost, Fernando Pedone, and André Schiper, editors. *Replication: Theory and Practice*, volume 5959 of *Lecture Notes in Computer Science*. Springer, 2010.

[40] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

[41] Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a coincidence: Sub-quadratic asynchronous byzantine agreement WHP. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[42] Miguel Correia. From byzantine consensus to blockchain consensus. *Essentials of Blockchain Technology (2019)*, 41, 2019.

[43] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. How to tolerate half less one byzantine nodes in practical distributed systems. In *23rd International Symposium on Reliable Distributed Systems (SRDS 2004), 18-20 October 2004, Florianpolis, Brazil*, pages 174–183. IEEE Computer Society, 2004.

[44] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. *Comput. J.*, 49(1):82–96, 2006.

[45] Miguel Correia, Giuliana Santos Veronese, Nuno Ferreira Neves, and Paulo Veríssimo. Byzantine consensus in asynchronous message-passing systems: a survey. *IJCCBS*, 2(2):141–161, 2011.

[46] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition, 2011.

[47] Tyler Crain. Experimental evaluation of asynchronous binary byzantine consensus algorithms with t<n/3 and o(n^2) messages and O(1) round expected termination. *CoRR*, abs/2004.09547, 2020.

[48] Tyler Crain. Two more algorithms for randomized signature-free asynchronous binary byzantine consensus with t<n/3 and o(n^2) messages and O(1) round expected termination. *CoRR*, abs/2002.08765, 2020.

[49] Flaviu Cristian, Houtan Aghili, H. Raymond Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. *Inf. Comput.*, 118(1):158–179, 1995.

[50] Nell B Dale and John Lewis. *Computer science illuminated*. Jones & Bartlett Learning, 2011.

[51] Stéphane Devismes, Sébastien Tixeuil, and Masafumi Yamashita. Weak *vs.* self *vs.* probabilistic stabilization. *Int. J. Found. Comput. Sci.*, 26(3):293–320, 2015.

[52] Edsger W Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[53] Benjamin Doerr, Leslie Ann Goldberg, Lorenz Minder, Thomas Sauerwald, and Christian Scheideler. Stabilizing consensus with the power of two choices. In *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures - SPAA '11*. ACM Press, 2011.

[54] Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

[55] Shlomi Dolev, Chryssis Georgiou, Ioannis Marcoullis, and Elad Michael Schiller. Self-stabilizing byzantine tolerant replicated state machine based on failure detectors. In *Cyber Security Cryptography and Machine Learning - Second International Symposium, CSCML 2018, Beer Sheva, Israel, June 21-22, 2018, Proceedings*, pages 84–100, 2018.

[56] Shlomi Dolev and Nir Tzachar. Randomization adaptive self-stabilization. *Acta Informatica*, 47(5-6):313–323, 2010.

[57] Shlomi Dolev and Jennifer L. Welch. Self-stabilizing clock synchronization in the presence of byzantine faults. *J. ACM*, 51(5):780–799, 2004.

[58] Assia Doudou and André Schiper. Muteness detectors for consensus with byzantine processes. In Brian A. Coan and Yehuda Afek, editors, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, page 315. ACM, 1998.

[59] K. Driscoll, B. Hall, Håkan Sivencrona, and P. Zumsteg. Byzantine fault tolerance, from theory to reality. In *Computer Safety, Reliability, and Security, 22nd International Conference, SAFECOMP 2003, Edinburgh, UK, September 23-26, 2003, Proceedings*, pages 235–248, 2003.

[60] Sisi Duan, Michael K. Reiter, and Haibin Zhang. BEAT: asynchronous BFT made practical. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 2028–2041. ACM, 2018.

[61] Swan Dubois, Maria Potop-Butucaru, Mikhail Nesterenko, and Sébastien Tixeuil. Self-stabilizing byzantine asynchronous unison. *J. Parallel Distrib. Comput.*, 72(7):917–923, 2012.

[62] Muhammad Fayyaz and Tanya Vladimirova. Survey and future directions of fault-tolerant distributed computing on board spacecraft. *Advances in Space Research*, 58(11):2352 – 2375, 2016.

[63] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[64] Roy Friedman, Achour Mostéfaoui, and Michel Raynal. Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. *IEEE Trans. Dependable Secur. Comput.*, 2(1):46–56, 2005.

[65] Chryssis Georgiou, Robert Gustafsson, Andreas Lindhé, and Elad Michael Schiller. Self-stabilization overhead: A case study on coded atomic storage. In Mohamed Faouzi Atig and Alexander A. Schwarzmann, editors, *Networked Systems - 7th International Conference, NETYS 2019, Marrakech, Morocco, June 19-21, 2019, Revised Selected Papers*, volume 11704 of *Lecture Notes in Computer Science*, pages 131–147. Springer, 2019.

[66] Guy Golan-Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. SBFT: A scalable and decentralized trust infrastructure. In *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019*, pages 568–580. IEEE, 2019.

[67] Maurice Herlihy. Blockchains and the future of distributed computing. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, page 155, 2017.

[68] Ted Herman. Probabilistic self-stabilization. *Inf. Process. Lett.*, 35(2):63–67, 1990.

[69] Amos Israeli and Marc Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In Cynthia Dwork, editor, *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, August 22-24, 1990*, pages 119–131. ACM, 1990.

[70] Pankaj Jalote. *Fault tolerance in distributed systems*. Prentice Hall, 1994.

[71] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1751–1767. ACM, 2020.

[72] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst.*, 27(4):7:1–7:39, 2009.

[73] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[74] Leslie Lamport. 1983 invited address solved problems, unsolved problems and non-problems in concurrency. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing (PODC'84)*, pages 1–11, New York, NY, USA, 1984. ACM.

[75] Leslie Lamport. Byzantizing paxos by refinement. In *Proceedings of the 25th International Symposium on Distributed Computing (DISC'11)*, pages 211–224, 2011.

[76] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[77] Guanfeng Liang and Nitin H. Vaidya. Error-free multi-valued consensus with byzantine failures. In Cyril Gavoille and Pierre Fraigniaud, editors, *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 11–20. ACM, 2011.

[78] Chao Liu, Sisi Duan, and Haibin Zhang. EPIC: efficient asynchronous BFT with adaptive security. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020*, pages 437–451. IEEE, 2020.

[79] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolic. XFT: practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pages 485–500, 2016.

[80] Oskar Lundström, Michel Raynal, and Elad Michael Schiller. Self-stabilizing indulgent zero-degrading binary consensus. In *International Conference on Distributed Computing and Networking 2021*, ICDCN '21, page 106–115, New York, NY, USA, 2021. ACM.

[81] Oskar Lundström, Michel Raynal, and Elad Michael Schiller. Self-stabilizing uniform reliable broadcast. In *Proceedings of the 8th International Conference on Networked Systems (NETYS'20)*, page (Accepted), 2021.

[82] Sujaya Maiyya, Victor Zakhary, Divy Agrawal, and Amr El Abbadi. Database and distributed computing fundamentals for scalable, fault-tolerant, and consistent maintenance of blockchains. *PVLDB*, 11(12):2098–2101, 2018.

[83] Mahyar R. Malekpour. A byzantine-fault tolerant self-stabilizing protocol for distributed clock synchronization systems. In *Stabilization, Safety, and Security of Distributed Systems, 8th International Symposium, SSS 2006, Dallas, TX, USA, November 17-19, 2006, Proceedings*, pages 411–427, 2006.

[84] Jean-Philippe Martin and Lorenzo Alvisi. Fast byzantine consensus. *IEEE Trans. Dependable Sec. Comput.*, 3(3):202–215, 2006.

[85] Alain J. Mayer, Rafail Ostrovsky, Yoram Ofek, and Moti Yung. Self-stabilizing symmetry breaking in constant space. *SIAM J. Comput.*, 31(5):1571–1595, 2002.

[86] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 31–42, New York, NY, USA, 2016. ACM.

[87] Zarko Milosevic, Martin Hutle, and André Schiper. On the reduction of atomic broadcast to consensus with byzantine faults. In *30th IEEE Symposium on Reliable Distributed Systems (SRDS 2011), Madrid, Spain, October 4-7, 2011*, pages 235–244. IEEE Computer Society, 2011.

[88] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[89] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo. Experimental comparison of local and shared coin randomized consensus protocols. In *2006 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06)*, pages 235–244, 2006.

[90] Henrique Moniz, Nuno Ferreira Neves, Miguel Correia, and Paulo Veríssimo. RITAS: services for randomized intrusion tolerance. *IEEE Trans. Dependable Secur. Comput.*, 8(1):122–136, 2011.

[91] Camilo Mora, Randi L Rollins, Katie Taladay, Michael B Kantar, Mason K Chock, Mio Shimada, and Erik C Franklin. Bitcoin emissions alone could push global warming above 2°. *Nature Climate Change*, 8(11):931, 2018.

[92] Sir Amyas Morse. Investigation: Wannacry cyber attack and the nhs, 2018.

[93] Achour Mostéfaoui, Moumen Hamouma, and Michel Raynal. Signature-free asynchronous byzantine consensus with t 2<n/3 and o(n^2) messages. In Magnús M. Halldórsson and Shlomi Dolev, editors, *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 2–9. ACM, 2014.

[94] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with t $<$ n/3, o(n2) messages, and O(1) expected time. *J. ACM*, 62(4):31:1–31:21, 2015.

[95] Achour Mostéfaoui and Michel Raynal. Signature-free broadcast-based intrusion tolerance: Never decide a byzantine value. In Chenyang Lu, Toshimitsu Masuzawa, and Mohamed Mosbah, editors, *Principles of Distributed Systems - 14th International Conference, OPODIS 2010, Tozeur, Tunisia, December 14-17, 2010. Proceedings*, volume 6490 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2010.

[96] Achour Mostéfaoui and Michel Raynal. Signature-free asynchronous byzantine systems: from multivalued to binary consensus with t$<$ n/3, o(n^2) messages, and constant time. *Acta Informatica*, 54(5):501–520, 2017.

[97] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[98] Svein Ølnes, Jolien Ubacht, and Marijn Janssen. Blockchain in government: Benefits and implications of distributed ledger technology for information sharing. *Government Information Quarterly*, 34(3):355–364, 2017.

[99] Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Asynchronous byzantine agreement with optimal resilience. *Distributed Comput.*, 27(2):111–146, 2014.

[100] Radia J. Perlman. Fault-tolerant broadcast of routing information. *Computer Networks*, 7:395–405, 1983.

[101] Michael O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 403–409. IEEE Computer Society, 1983.

[102] HariGovind V. Ramasamy and Christian Cachin. Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast. *IACR Cryptol. ePrint Arch.*, 2006:82, 2006.

[103] Michel Raynal. Signature-free communication and agreement in the presence of byzantine processes (tutorial). In Emmanuelle Anceaume, Christian Cachin, and Maria Gradinariu Potop-Butucaru, editors, *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*, volume 46 of *LIPIcs*, pages 1:1–1:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[104] Michel Raynal. *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*. Springer, 2018.

[105] Kenji Saito and Hiroyuki Yamada. What's so different about blockchain? - blockchain is a probabilistic state machine. In *36th IEEE International Conference on Distributed Computing Systems Workshops, ICDCS Workshops, Nara, Japan, June 27-30, 2016*, pages 168–175, 2016.

[106] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.

[107] Marco Schneider. Self-stabilization. *ACM Computer Survey*, 25(1):45–67, March 1993.

[108] Manfred Schwarz, Kyrill Winkler, and Ulrich Schmid. Fast consensus under eventually stabilizing message adversaries. In *Proceedings of the 17th International Conference on Distributed Computing and Networking - ICDCN '16*. ACM Press, 2016.

[109] Yee Jiun Song and Robbert van Renesse. Bosco: One-step byzantine asynchronous consensus. In Gadi Taubenfeld, editor, *Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings*, volume 5218 of *Lecture Notes in Computer Science*, pages 438–450. Springer, 2008.

[110] T. K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Comput.*, 2(2):80–94, 1987.

[111] Andrew S. Tanenbaum and Maarten van Steen. *Distributed systems - principles and paradigms (2nd edition)*. Pearson Education, 2007.

[112] Sam Toueg. Randomized byzantine agreements. In *Proceedings of the third annual ACM symposium on Principles of distributed computing - PODC '84*. ACM Press, 1984.

[113] Volker Turau. Making randomized algorithms self-stabilizing. In Keren Censor-Hillel and Michele Flammini, editors, *Structural Information and Communication Complexity - 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1-4, 2019, Proceedings*, volume 11639 of *Lecture Notes in Computer Science*, pages 309–324. Springer, 2019.

[114] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. Spin one's wheels? byzantine fault tolerance with a spinning primary. In *28th IEEE Symposium on Reliable Distributed Systems (SRDS 2009), Niagara Falls, New York, USA, September 27-30, 2009*, pages 135–144, 2009.

[115] Marko Vukolic. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *Open Problems in Network Security - IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers*, pages 112–125, 2015.

[116] Masafumi Yamashita. Probabilistic self-stabilization and biased random walks on dynamic graphs. *Int. J. Netw. Comput.*, 2(2):147–159, 2012.

[117] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 347–356. ACM, 2019.

[118] Junko Yoshida. Toyota case: Single bit flip that killed, 2013. Available at `https://www.eetimes.com/toyota-case-single-bit-flip-that-killed` (accessed *December* 12, 2020).