Recent Results on Fault-Tolerance Consensus in Message-Passing Networks

Lewis Tseng

Department of Computer Science
University of Illinois at Urbana-Champaign

Email: ltseng3@illinois.edu

May 2016*

Abstract

This paper surveys recent results on fault-tolerant consensus in message-passing networks. We focus on two categories of works: (i) new problem formulations (including input domain, fault model, network model...etc.), and (ii) practical applications. For the second part, we focus on Crash Fault-Tolerant (CFT) systems that use Paxos or Raft, and Byzantine Fault-Tolerant (BFT) systems. We also briefly discuss Bitcoin, which can be related to solving Byzantine consensus in anonymous systems.

---

*Revised on July 2016.

1 Introduction

Fault-tolerant *consensus* has received significant attentions over the past three decades [19, 84] since the seminal work by Lamport, Shostak, and Pease [102, 76]. The problem has been studied extensively – topics include solving consensus and/or proving time/communication complexity under different model assumptions, communication mechanisms, and correctness conditions. Please refer to [19, 84] for some classic results on fault-tolerant consensus. In this paper, we survey recent efforts on fault-tolerant consensus in message-passing networks. References [46, 104, 31, 84, 19] have presented abundant discussions on this topic, especially, on the techniques and comparison of different consensus algorithms. To complement theses prior works, we present this survey from two new angles:

- *Exploration of New Problem Formulations*: Plenty of different consensus problems have been introduced in the past ten years for solving more complicated tasks and accommodating different systems and networks requirements. New problem formulations include additional correctness properties, different fault models, different communication networks, or different input/output domains. For this part, we focus on the comparison of recently proposed problem formulations.
- *Exploration of Practical Applications*: Since Castro and Liskov published their seminal work, PBFT (Practical Byzantine Fault-Tolerance) [36], significant effort has been devoted to make *Byzantine Fault-Tolerance* (BFT) practical and efficient. We will discuss works on improving BFT. Moreover, due to the recent popularity of Bitcoin [2], industry and academia have renewed interested in Byzantine consensus. We also survey relevant results. For the second part, we focus on the practical challenges and applications.

1.1 Classic Definitions of Fault-tolerant Consensus

We consider the consensus problem in a point-to-point message-passing network, which is modeled as an *undirected* graph. Without specifically mentioning, the communication network is assumed to be *complete* in this survey, i.e., each pair of nodes can communicate with each other directly. In the fault-tolerant consensus problem [19, 84], each node is given an *input*, and after a finite amount of time, each fault-free node should produce an *output* – consensus algorithms should satisfy the *termination* property. Additionally, the algorithms should also satisfy appropriate *validity* and *agreement* conditions. There are three main categories of consensus problems regarding different agreement properties:

1. *Exact* consensus [102, 73]: fault-free nodes have to agree on exactly the *same* output.
2. *Approximate* consensus [53, 55]: fault-free nodes have to agree on “roughly” the *same* output – the difference between outputs at any pair of fault-free nodes is bounded by a given constant ϵ $(\epsilon > 0)$ of each other.
3. *k-set* consensus [41, 49]: the number of distinct outputs at fault-free nodes is at most k.

Validity property is also required for consensus algorithms to produce meaningful outputs, since the property defines the acceptable relationship between inputs and output(s). Typical validity includes: (i) *strong validity*: output must be an input at some fault-free node, (ii) *weak validity*: if all fault-free nodes have the same input v, then v is the output, and (iii) *validity* (for approximate consensus): output must be bounded by the input at fault-free nodes. A consensus algorithm is said to be correct if it satisfies termination, agreement and validity properties given that certain number of nodes may become faulty. In this paper, we focus on three types of node failures – Byzantine, crash, and omission faults. The only exception is Section 2.3 where we focus on link faults.

The other key component of the consensus problem definition is about system *synchrony*, i.e., a model for the relative speed of nodes and the network. There are also three main categories [19, 84, 54, 28]:

1. *Synchronous* systems: each node proceeds in a lock-step fashion, and there is a known upper bound on message delay.
2. *Partially synchronous* systems: there exists a *partial synchronous* period from time to time. In such a period, fault-free nodes and the network stabilize and behave (more) synchronously.[1]
3. *Asynchronous* systems: no known bounds exist on nodes' processing speed or message delays.

In Section 2, we discuss papers that defined new consensus problems which either assume variants of aforementioned properties or introduce new correctness properties. The main purpose is to give a big picture on the problem domains that have been explored in the literature.

1.2 Practical Applications of Fault-tolerant Consensus

In Section 3, we discuss recent efforts focusing on the practical applications. Consensus is an important primitive that has wide applications such as state-machine replication (SMR) [110], and distributed storage. We address three main applications: (i) crash fault-tolerant systems based on variants of Paxos [73, 74] and Raft [97], (ii) Byzantine fault-tolerant (BFT) systems, and (iii) Bitcoin [94, 95] a popular cryptocurrency. For part (ii), we discuss several techniques to improve the performance, including speculative execution, execution/agreement separation, hardware-based solution, dynamic switch among abortable components, and relaxed properties. For part (iii), we focus on the comparison of Bitcoin and Byzantine consensus/BFT systems.

2 Exploration of New Problem Formulations

Researchers have studied the consensus problems under more generalized assumptions. In this section, we focus on the following generalizations: (i) input/output domain, (ii) communication

---

[1] Note that there are also other definitions of partial synchrony. We choose this particular definition, since many BFT systems only satisfy liveness under this particular definition. Please refer to [54, 15] for more models.

network and synchrony assumptions, (iii) link fault models, and (iv) properties other than validity and agreement, such as early-stopping and one-step properties. In this section (with the exception on discussing link fault models), we assume that among n nodes in the system, up to f of them may become Byzantine faulty or crash. Byzantine faulty nodes may have an arbitrary behavior.

2.1 Input/Output Domain

Multi-Valued Consensus In the original *exact* Byzantine consensus problem [102, 76], both input and output are binary values. Later, [83, 124] proposed the multi-valued version in which input may take more than two *real* values. Recently, multi-valued consensus received renewed attentions and researchers proposed algorithms that achieve (asymptotically) optimal communication complexity (number of bits transmitted) in both synchronous and asynchronous systems. Surprisingly, for L-bit input, these algorithms achieve (asymptotic) communication complexity of $O(nL)$ bits when L is large enough.

In synchronous systems, Fitzi and Hirt proposed a Byzantine multi-valued algorithm with small error probability [57]. The main idea is to reduce the inputs to much smaller messages using universal hash function. (Classic) Byzantine consensus is then performed using these reduced hashed values as inputs. Finally, consensus is achieved by obtaining the input value from nodes that have the same hashed values (if there is enough number of such nodes) [57]. Later, Liang and Vaidya combined a different reduction technique (that divides an input into a large number of small values) with novel coding technique to construct an error-free algorithm in synchronous systems [80]. One key contribution is to introduce a lightweight fault detection (or fault diagnosis) mechanism using coding [80]. Such fault diagnosis works because the inputs are divided into batches of small values. In each batch, either consensus (on the small value of this batch) can be achieved with small communication complexity or some faulty nodes will be identified. Once all faulty nodes are identified, then consensus on the remaining batches becomes trivial. Since number of faulty node is bounded, consensus on most batches can be achieved with small communication complexity [80].

Subsequently, variants of reduction technique were applied to solve consensus problems with large inputs in asynchronous systems. References [101, 100] provided multi-valued algorithms with small error probability. Patra improved the results and proposed an error-free algorithm [99]. These algorithms terminate with overwhelming probability; however, the expected time complexity is large because these algorithms first divide inputs to small batches and achieve consensus on each batch with various fault diagnosis mechanisms. Typically, to achieve optimal communication complexity, the number of batches is in the same order of L. In contrast, Mostefaoui and Raynal focused on a different goal [92]. They proposed an asynchronous consensus algorithm by reducing multi-valued consensus to binary consensus, and their algorithm requires $O(n^2)$ messages (i.e., $O(n^2L)$ bits) and constant expected time complexity. The proposed algorithm relies on two components: Rabin's common coin [103] and an all-to-all broadcast communication abstraction to exchange binary values among fault-free nodes [92].

Multi-valued consensus has also been studied under the crash fault model in which nodes may crash; otherwise, they follow the algorithm specification. Mostefaoui et al. considered multi-valued consensus algorithm when nodes may suffer crash failures in both synchronous and asynchronous systems [11]. Later, Zhang and Chen proposed a more efficient multi-valued consensus algorithm in asynchronous systems with crash faults [135]. King and Saia studied a slight different problem called almost-everywhere Byzantine agreement in synchronous systems with a strong adversary that corrupt nodes adaptively [71]. The proposed algorithm has a small error probability.

High-Dimensional Input/Output In the Byzantine vector consensus (or multi-dimensional consensus) [86, 126], each node is given a d-dimensional vector of reals as its input ($d \geq 1$), and the output is also a d-dimensional vector. In complete networks, the recent papers by Mendes and Herlihy [86] and Vaidya and Garg [126] addressed approximate vector consensus in the presence of Byzantine faults.[2] These papers yielded lower bounds on the number of nodes, and algorithms with optimal resilience in asynchronous [86, 126] as well as synchronous systems [126]. The algorithms in [86, 126] are generalizations of the optimal iterative approximate Byzantine consensus for scalar inputs in asynchronous systems [14]. The algorithms in [86, 126] require sub-routines for geometric computation in the d-dimensional space to obtain the local state in each iteration; whereas, a simple average operation suffices in [14]. Subsequent work [125] explored the approximate vector consensus problem in incomplete *directed* graphs. Later, [121] proposed the convex hull consensus problem, in which fault-free nodes have to agree on "largest possible" polytope in the d-dimensional space that may not necessarily equal to a d-dimensional vector (a single point). The asynchronous algorithm in [121] bears some similarity to the ones in [86, 126, 14]; however, Tseng and Vaidya used a different communication abstraction to achieve the "largest possible" polytope. Moreover, Tseng and Vaidya introduced a new proof technique to prove the correctness of iterative consensus algorithms when the output is a polytope [121].

2.2 Communication Network and Synchrony

The fault-tolerant consensus problem has been studied extensively in complete networks (e.g., [102, 73, 19, 84, 53]) and in undirected networks (e.g., [56, 51]). In these works, any pair of nodes can communicate with each other reliably either directly or via at least $2f + 1$ node-disjoint paths (for Byzantine faults) or $f + 1$ node-disjoint paths (for crash faults). Recently, researchers revisited such assumptions and enriched the problem space in three main directions: directed graphs, dynamic graphs, and partial synchrony.

Directed Graphs Recently, researchers started to explore various consensus in arbitrary directed graphs, i.e., two pairs of nodes may not share a bi-directional communication channel, and not every pair of nodes may be able to communicate with each other directly or indirectly. Tseng and Vaidya [123] proved *tight* necessary and sufficient conditions on the underlying communication graphs for achieving (i) exact crash-tolerant consensus in synchronous systems, (ii) approximate crash-tolerant consensus in asynchronous systems, and (iii) exact Byzantine consensus in synchronous systems using *general* algorithms. An algorithm is *general* if nodes are allowed to have topology knowledge and the ability to route messages (send and receive messages

---

[2] These two papers [86] and [126] independently addressed the same problem, and developed different algorithms.

using multiple node-disjoint paths). Furthermore, unlike iterative algorithms [53, 14], the state maintained at each node in general algorithms is not constrained. Lili and Vaidya [116] proved tight conditions for achieving approximate Byzantine consensus using general algorithms.

Much efforts have also been devoted on *iterative* algorithms in incomplete graphs. In iterative algorithms, (i) nodes proceed in iterations; (ii) the computation of new state at each node is based only on local information, i.e., nodes own state and states from neighboring nodes; and (iii) after each iteration of the algorithm, the state of each fault-free node must remain in the convex hull of the states of the fault-free nodes at the end of the previous iteration. Vaidya et al. [127] proved *tight* conditions for achieving approximate Byzantine consensus in synchronous and asynchronous systems using *iterative* algorithms. The tight condition for approximate crash-tolerant consensus in asynchronous systems was also proved in [120].

A more restricted fault model – called "malicious" fault model – in which the faulty nodes are restricted to sending identical messages to their neighbors has also been explored extensively, e.g., [77, 78, 134, 79]. LeBlanc and Koutsoukos [77] addressed a continuous time version of the consensus problem with malicious faults in complete graphs. LeBlanc et al. [79] have obtained *tight* necessary and sufficient conditions for tolerating up to f faults in the network.

The aforementioned approximate algorithms (e.g., [127, 116, 79]) are generalizations of the iterative approximate consensus algorithm in complete network [53, 55]. However, to accommodate directed links, the proofs are more involved. Particularly, for sufficiency, one has to prove that all fault-free nodes must be able to receive the non-trivial amount of a state at some fault-free node in finite number of iterations. The exact consensus algorithms in [123] also require that some information has to be propagated to all fault-free nodes even if some nodes may fail. Generally, the algorithms in [123] proceeds in phases such that in each phase, a group of nodes try to send information to the remaining nodes. The algorithms are designed to maintain validity at all time. Additionally, if no failure occurs in a phase, then agreement can be achieved.

The necessity proofs in the work on directed graphs (e.g., [127, 79, 123]) are generalizations of the indistinguishability proof [18, 56]. The main contributions are to identify how faulty nodes can block the information flow so that (i) fault-free nodes can be divided into several groups, and (ii) there exists a certain faulty behavior such that different groups of fault-free nodes have to agree on different outputs.

Dynamic Graphs Researchers have also explored consensus problem in directed dynamic networks [24, 23, 38, 39, 111], where communication network changes over time. For synchronous systems, Charron-Bost et al. [38, 39] solved *approximate* crash-tolerant consensus in directed dynamic networks using iterative algorithms. In the asynchronous setting, Charron-Bost et al. [38, 39] addressed approximate consensus with crash faults in *complete* graphs.

References [24, 111, 23] considered the *message adversary*, which controls the communication pattern, i.e., the adversary has the power to specify the sets of communication graphs. Biely et al. studied the exact consensus problem [23] and k-set consensus problem [24, 111] in dynamic networks under *message adversary*, and the system is assumed to be synchronous. All the nodes are assumed to be fault-free in [24, 111, 23]. No message is tampered in message adversary model.

The algorithms in aforementioned papers share some similarity with their counter parts in complete graphs, e.g., [84, 19]. The main contributions of these papers are to identify concise definitions of dynamic graphs so that it can be proved that useful information can be propagated to enough number of nodes in the presence of faults.

Partial Synchrony Alistarh et al. [17] considered k-set consensus in partially synchronous systems, and presented the (asymptotically) tight bound on the complexity of set agreement in such systems. Milosevic et al. [88] considered permanent and transient transmission faults in a variation of partially synchronous systems, and proved necessary and sufficient conditions on the number of nodes n to tolerate permanent and transient transmission faults. Hamouma et al. [63] studied the consensus problem when only a few links may be synchronous throughout the execution of the algorithm.

Alistarh et al. [16] addressed a fundamental question of partially synchronous systems: "For how long does the system need to be synchronous to solve crash-tolerant consensus?" The core idea of the algorithm in [16] relies on (i) the mechanism to detect asynchrony, and (ii) determine when to update value safely (without violating validity) based on asynchrony detection. Bouzid et al. [28] studied the problem from a different aspect – how many eventually synchronous links are necessary for achieving consensus. They introduced a notion of eventual $\langle t{+}1 \rangle$bisource which characterize the necessary and sufficient timing condition to solve consensus. This condition requires an existence of fault-free nodes such that it has an eventually synchronous incoming links from f other fault-free nodes, and eventually synchronous outgoing links to f other fault-free nodes. The proposed algorithm in [28] uses two novel components: a new all-to-all communication abstraction for fault-free nodes to eventually agree on a set of values, and an object to ensure that fault-free nodes eventually converge to a single value.

2.3 Link Fault Model

In addition to node failures, significant effort has also been devoted to the problem of achieving consensus in the presence of link failures [40, 25, 107, 108, 109]. Santoro and Widmayer proposed the *transient* Byzantine link failure model: a different set of links can be faulty at different time [107, 108]. The nodes are assumed to be fault-free in the model. They characterized a necessary condition and a sufficient condition for undirected networks to achieve consensus in the transient link failure model; however, the necessary and sufficient conditions do not match: the necessary and sufficient conditions are specified in terms of node degree and edge-connectivity,[3] respectively.

Subsequently, Biely et al. proposed another link failure model that imposes an upper bound on the number of faulty links incident to each node [25]. As a result, it is possible to tolerate $O(n^2)$ link failures with n nodes in the new model. Under this model, Schmid et al. proved lower bounds on number of nodes, and number of rounds for achieving consensus [109]. [122] considered iterative consensus in arbitrary directed graphs under transient Byzantine link failure model.

For exact consensus problem, it has been shown that (i) an undirected graph of $2f + 1$

---

[3] A graph $G = (\mathcal{V}, \mathcal{E})$ is said to be k-edge connected, if $G' = (\mathcal{V}, \mathcal{E} - X)$ is connected for all $X \subseteq \mathcal{E}$ such that $|X| < k$.

node-connectivity[4] is able to tolerate f Byzantine nodes [56]; and (ii) an undirected graph of $2f+1$ edge-connectivity is able to tolerate f Byzantine links [108]. Researchers also showed that $2f+1$ node-connectivity is both necessary and sufficient for the problem of information dissemination in the presence of either f faulty nodes [117] or f *fixed* faulty links [118]. Unlike the "transient" failure model, the faulty links are assumed to be fixed throughout the execution of the algorithm in [118].

Charron-Bost and Schiper proposed the HO (Heard-Of) model that captures node failures and message losses at the same time [40]. To the best of our knowledge, the HO model is the first model unifying system synchrony and node crashes together. The HO model assumes round-based algorithms, which consists of three steps: (i) send messages, (ii) receive messages, and (iii) perform computation (specified by the algorithm). For each round r and each node i, let $HO(i, r)$ denote the set of nodes that node i has "heard of" at round r. Then the model also specifies a set of *communication predicates* over all $HO(i, r)$ to capture failures, message loss, or delayed messages. The benefits of the HO models are: (i) it puts different types of failures in a unified framework, including static, dynamic, permanent or transient faults, and (ii) compared with the classic fault models, the impossibility proofs and correctness proofs (for given algorithms) are in general shorter and simpler [40]. In [40], Charron-Bost and Schiper discussed communication predicates that map to classic problem specification, e.g., "Synchronous system, reliable links, at most f crash failures" or "Partially synchronous system, eventual reliable links, at most f crash failures", and identified the relationships among these communication predicates and solvability of consensus problems specified in the HO model with these predicates. Subsequently, Biely et al. generalized the model to value faults, which can also capture Byzantine node and link faults [26].

2.4 Extra Properties

Early-Stopping Property In synchronous systems, an algorithm has an early-stopping property if the algorithm can terminate early if there is less than f faults in an execution. Suppose that given an execution, an actual number of faults in a system is t, where $t \leq f$. It has been shown that fault-tolerant consensus cannot be achieved in $\leq t+1$ rounds using deterministic algorithms in synchronous systems [84]. That is, the lower bound of round complexity is $\min\{t+2, f\}$ for crash faults [70], omission fault [98], and Byzantine faults [52]. In [52], Dolev and Lenzen proposed a new property, namely early-deciding, which requires fault-free nodes to decide early but the decided nodes may continue to send messages in to help other undecided nodes. They showed that an early-deciding algorithm requires more message complexity than normal consensus algorithms [52]. The proof consists of two parts: (i) find a "pivotal" node that is critical for whether the execution would result in output 0 or output 1, and (ii) ensure that $\Omega(f^2)$ messages have to be exchanged in certain rounds to achieve consensus. As a result, they are able to show that for any $\min\{t+2, f\}$-deciding binary consensus algorithm and any $1 \leq t \leq f/2$, there is an execution such that number of faults is t and fault-free nodes send at least $f^2 t/44$ messages.

---

[4] A graph $G = (\mathcal{V}, \mathcal{E})$ is said to be k-node connected, if $G' = (\mathcal{V} - X, \mathcal{E})$ is connected for all $X \subseteq \mathcal{V}$ such that $|X| < k$.

One-Step Property An asynchronous consensus algorithm has one-step property if there is no contention (i.e., all fault-free nodes propose the same input value), the algorithm terminates within in one communication step. A communication step consists of three events: (i) send messages, (ii) receive messages, and (iii) do local computation and update local state. One-step property is first proposed for crash-tolerant consensus algorithms [29] and later extended to Byzantine consensus algorithms [58, 115]. These algorithms share similar structures: (i) use communication primitives to exchange values, and produce output if there is enough match, and (ii) use traditional consensus algorithms to achieve the consensus if no output is generated in the first phase. Typically, the lower bound on the number of nodes to achieve one-step property is more than classic correctness properties (validity and agreement).

3 Exploration of Practical Applications

We start with systems that tolerate crash node faults, particularly, two families of algorithms – Paxos [73] and Raft [97]. Then, we discuss efforts to tolerate more complex failures and BFT (Byzantine Fault-Tolerance) systems. Finally, we compare Bitcoin-related work [94] with BFT systems and Byzantine consensus. Typically, these systems satisfy correctness (or safety) in asynchronous network; however, to ensure progress (or liveness), there must exist some time periods that all messages are received within time.

3.1 Paxos and Raft

Paxos [73, 74, 75, 89] is the well-known family of consensus protocols tolerating crash node faults in asynchronous network. Since Paxos was first proposed by Lamport [73, 74], variants of Paxos were developed and implemented in practical systems, such as Chubby lock service used in many Google systems [30, 44], and membership management in Windows Azure [35].[5] Yahoo! also developed ZaB [105], a protocol achieving atomic broadcast in asynchronous network with FIFO channels, and used ZaB to build the widely-adopted coordination service, ZooKeeper [67]. ZooKeeper is later used in many practical storage systems, like HBase [5] and Salus [131]. Recently, many novel mechanisms have been proposed to improve the performance of Paxos, including quorum lease [91], diskless Paxos [119], even load balancing [90], and time bubbling (for handling nondeterministic network input timing) [48].

In 2014, Ongaro and Ousterhout from Stanford proposed a new asynchronous consensus algorithm – Raft [97]. Their main motivation was to simplify the design of consensus algorithm so that it is easier to understand and verify the design and implementation. One interesting (social) experiment by Ongaro and Ousterhout was mentioned in [97]: "*In an informal survey of attendees at NSDI 2012, we found few people who were comfortable with Paxos, even among seasoned researchers*". To simplify the (conceptual) design, Raft integrates the consensus solving part deeply with leader election and membership/configuration management [97].[6] After their

---

[5] We would like to thank the anonymous reviewer who pointed out that Windows Azure also uses ZooKeepr to manage virtual machines [1].

[6] While the original Paxos [73, 74] is theoretically elegant, practitioners have found it hard to implement Paxos in practice [37]. One difficulty mentioned in [37] is that membership/configuration management is non-trivial in practice.

publication, Raft has quickly gained popularity, and been used in practical key-value store systems such as etcd [4] and RethinkDB [9]. Please refer to their website [8] for a list of papers and implementations.[7]

3.2 Arbitrary State Corruption

Recently, researchers explored fault model beyond crash node failures. One such fault model is called Arbitrary State Corruption (ASC) [45, 20, 21]. In the ASC fault model, the whole state of a node may transition to an arbitrary state due to incidents like bit flips or hardware error. However, the failure is not caused by a malicious adversary. Thus, it is generally assumed that a message from a faulty node can be detected in the ASC model [20, 21]. Note that the ASC model is a proper subset of Byzantine fault model, since Byzantine nodes can behave arbitrarily, including sending messages in a way that may not be detected.

Correia et al. introduced a library, PASC, which relies on different check mechanisms (e.g., CRC code) to harden crash-fault tolerant algorithms against ASC faults [45]. PASC does not replicate the entire node; rather, it replicates internal states of each node; thus, the overhead is moderate comparing to BFT replication (discussed in Section 3.3). Behrens et al. proposed a framework to harden distributed systems using arithmetic codes, which is able to detect transit and permanent hardware errors with high probability [21]. Subsequently, Behrens et al.[20] observed that the technique in [45] does not manage memory usage efficiently, and the mechanism in [21] incurs large latency due to the component for encoding executions. The authors addressed the aforementioned issues, and used their technique to harden memcached [7] with moderate overhead [20].

3.3 Byzantine Fault Tolerance

Since Castro and Liskov published their seminal work PBFT (Practical Byzantine Fault-Tolerance) [36], significant effort has been devoted to improving *Byzantine Fault-Tolerance* (BFT), including (i) reducing the overhead like communication costs, or replication costs, and (ii) providing higher throughput or lower latency (in the form of round complexity). Generally speaking, BFT system replicates deterministic state machines over different machines (or *replicas*) to tolerate Byzantine node failures. In other words, BFT systems implement the State Machine Replication systems [110] that tolerate Byzantine faults. The main challenge is to design a system such that it behaves like a centralized server to the clients in the presence of Byzantine failures. More precisely, the system is given requests from the clients, and the goals are: (i) the fault-free replicas agree on the total order of the requests, and then the replicas execute the requests following the agreed order (safety); and (ii) clients learn the responses to their requests eventually (liveness). Usually, liveness is guaranteed only in the *grace periods*, i.e., when messages are delivered in time.

---

[7] Paxos has been the de facto standard of consensus algorithms for a long time [6]; however, we feel that it is still of interests to discuss Raft as well, as Raft has gained more and more attentions in academia and industry [8].

Improving Performance Castro and Liskov's work on Practical Byzantine Fault-Tolerance (PBFT) showed for the first time that BFT mechanism is useful in practice [36]. PBFT requires $3f + 1$ replicas, where f is the upper bound on the number of Byzantine failures in the whole system. Subsequently, Quorum-based solutions Q/U [12] and HQ [47] have been proposed, which only require one round of communication in contention-free case (when no replica fails, and the network has stable performance and no contention happens) by allowing clients directly interact with the replicas to agree on an execution order.Such type of mechanisms reduce latency (number of rounds required) in some case, but is shown to be more expensive in other cases [72]. Hence, Zyzzyva [72] focused on increasing performance in failure-free case (when no replica fails) by allowing speculative operations and novel roll-back mechanism. Zyzzyva requires $3f+1$ replicas; however, a single crash failure would significantly reduce the performance by forcing Zyzzyva protocol run in slow mode – where no speculative operation can be executed [72]. Thus, Kotla et al. also introduced Zyzzyva5, which can be executed in fast mode even if there are crash failures, but Zyzzyva5 requires $5f + 1$ replicas [72]. Subsequently, Scrooge [113] reduced the replication cost of Zyzzyva5 by requiring the participation from clients to detect replicas' misbehaviors.

The aforementioned BFT systems are designed to optimize performance for certain circumstances, e.g., HQ for contention-free case and Zyzzyva for failure-free case. Guerraoui et al. proposed a new type of BFT systems that can be constructed to have optimized performance under difference circumstances [61]. Such tunable design is useful, since it provides the flexibility of choosing different performance trade-off according to the network performance and application requirements. Their systems are based on three core concepts: (i) abortable requests, (ii) composition of (abortable) BFT instances, and (iii) dynamic switching among BFT instances. The tunable parameter specifies the progress condition under which a BFT instance should not abort. Some example conditions include contention, system synchrony or node failures. In [61], Guerraoui et al. showed how to construct new BFT systems with different parameter; particularly, they proposed (i) *AZyzzyva* which composes Zyzzyva and PBFT together to have more stable performance than Zyzzyva and faster failure-free performance than PBFT, and (ii) *Aliph* which composes PBFT, Quorum-based protocol optimized for contention-free case, and Chain-based protocol optimized for high-contention cases without failures and asynchrony [61].

For computation-heavy workload, Yin et al. proposed to separate agreement protocol from executions of clients' requests [133]. This separation mechanism reduces the replication cost to $2f + 1$. Note that the system still requires $3f + 1$ replicas to achieve agreement on the order of the clients' requests, but the executions of requests, and data storage only occur at $2f + 1$ replicas. Later, Wood et al. built a system, ZZ, which reduces the replication cost to $f+1$ using virtualization technique [132]. The idea behind ZZ is that $f+1$ active replicas are sufficient for fault detection, and when fault is detected, their virtualization technique allows ZZ to replace the faulty replica by waking up fresh replica and retrieving current system state with small overhead [132].

Clement et al. observed that a single Byzantine replica or client can significantly impact the performance of HQ, PBFT, Q/U and Zyzzyva [43]. Thus, they proposed a new system Aardvark, which provides good performance when Byzantine failures happen by sacrificing the failure-free case performance [43]. Later, Clement et al. also demonstrated how to combine Zyzzyva and Aardvark so that the new system, Zyzzyvark, not only tolerates faulty clients, but also enjoys fast performance in the failure-free case by leveraging speculative operations [42].

Hardening Crash-Tolerant Systems Since most existing systems are designed to tolerate crash faults, there are efforts on hardening existing crash-tolerant systems against Byzantine fault models. Haeberlen et al. was among the first to propose using log-based detection mechanism to hardening crash-tolerant systems [62]. They proposed a library called PeerReview that can be used to detect Byzantine faults, and such detection can be irrefutably linked to faulty nodes – the identity of faulty nodes can be eventually learned by all fault-free nodes. Unfortunately, as discovered by Ho et al. [65], PeerReview can only be used to detect a subset of Byzantine failures. Nysiad [65] transforms crash-tolerant protocols to Byzantine-tolerant protocols by assigning a set of guards to verify each replica's behavior. However, Nysiad needs a logically centralized service to take care of configuration change; thus, Nysiad requires high overhead [45]. UpRight [42] is an architecture to integrate BFT and crash-tolerant systems with small overhead. UpRight has taken ideas from three prior systems: speculative execution [72], robustness to clients' failure [43], and agreement/execution separation [133]. One novelty of UpRight is to introduce shim layers for clients and servers of existing existing crash-tolerant systems that can (i) order clients' requests, and (ii) verify results from servers. Clement et al. used UpRight library to make ZooKeeper [67] tolerate Byzantine faults [42].

Hardware-based BFT Different from the aforementioned software-based BFT mechanisms, other researchers proposed using trusted hardware components to reduce costs. MinBFT [128] uses trusted hardware to build an unique sequential identifier generator, which is then used to verify messages from each replica. With such scheme, MinBFT only requires $2f + 1$ agreeing replicas. CheapBFT [69] relies on an FPGA-based trusted components to authenticate messages, and is able to tolerate all-but-one failures, i.e., it only requires $f + 1$ replicas. Recently, István et al. proposed a novel idea of using FPGA to achieve Byzantine-tolerant consensus and atomic broadcast [68]. Then, they showed how to use such FPGA-based atomic broadcast to make ZooKeeper tolerate Byzantine faults with small decrease of performance (compared to crash-tolerant one). One down side of their mechanism is that the developers need to implement an application-specific network protocol [68].

Relaxed BFT Inspired by the popularity of practical eventually consistent systems (e.g., [3, 50]), some researchers also proposed relaxed safety and liveness properties for BFT systems. CLBFT sacrifices liveness for higher safety, i.e., tolerating more replica failures, by increasing the quorum size (in proportion of the number of replicas) [106]. Zeno [114] chose eventual consistency to provide higher availability when network partition happens. Depot [85] only ensures a fork-join-causal consistency (a model slightly weaker than causal consistency) to eliminate trust for safety, i.e., a client needs to trust only himself to ensure the safety property. Prophecy [112] focuses on increasing throughput for read-heavy workloads; however, Prophecy only provides delay-once consistency (a new consistency model weaker than strong consistency [112]), and relies on a trusted component to detect misbehaviors. Liu et al. proposed the concept of XFT (cross fault-tolerance), which relax the degree of fault-tolerance [81]. Particularly, XFT is correct only when all the following hold: (i) only crash faults happen in asynchronous periods;

and (ii) non-crash faults (Byzantine faults) happen only in synchronous periods (grace periods). By relaxing the guarantees, the authors build XPaxos which has comparable performance of crash-fault-tolerant systems and tolerates Byzantine faults (in grace periods) [81].

BFT Storage System There are also BFT systems specifically designed for storage systems. Goodson et al. proposed an erasure-coded storage system tolerating Byzantine replicas and clients using $4f + 1$ replicas [60]. The main technique to detect faulty client writing different values to different replicas is having the next fault-free client detect the inconsistency (This scheme is possible due to the benefit of coding). Based on this system, Abd-El-Malek et al. proposed a lazy verification protocol to reduce client's workload, which shifts the work to storage replicas during idle time [13]. However, the scheme still requires $4f + 1$ replicas, and consumes high bandwidth [64]. Later, Hendricks et al. built another erasure-coded storage system [64] which relies on a short checksum comprised of cryptographic hashes and homomorphic fingerprints to optimize the throughput in the contention-free case (when no replica fails, and the network has stable performance and no contention happens). The system requires $3f + 1$ replicas. Recently, Cachin et al. built a BFT storage system, MDStore, which only requires $2f + 1$ replicas under the assumption that the client is always fault-free when writing data [32, 33]. MDStore system had two novelties: (i) separation of data and metadata storage, and (ii) provide metadata service using only lightweight cryptographic hash functions. MDStore tolerates any number of Byzantine readers and crash-faulty writers and up to f Byzantine faulty replicas.

Cloud-of-Clouds The idea of building BFT storage systems over intercloud (or cloud-of-clouds) becomes popular lately, since as discussed in [129], the assumption of failure independence holds naturally due to the different cloud administrators, geographical locations and implementations from different cloud service providers. Cachin et al. proposed a layered architecture for BFT storage systems over intercloud, ICStore (abbreviating InterCloud Storage) [34]. One novelty of ICStore is to provide different dependability goals: (i) confidentiality, (ii) integrity, and (iii) reliability and consistency. ICStore's layered architecture allows clients to choose different levels of dependability and performance by choosing by selecting different operation point for each layer [34]. Independently, Bessani et al. proposed DEPSKY, a BFT storage system supporting efficient encoding and confidentiality [22] with $3f + 1$ replicas. However, the liveness property is weakened in DEPSKY, i.e., the read protocol ensures responses only when a finite number of contending writes happen. NCCloud focuses on both fault tolerance and storage repair [66] by designing new regenerating code that has low repair cost and can be used to detect a subset of Byzantine behaviors.

3.4 Bitcoin

Bitcoin is a digital currency system proposed by Satoshi Nakamoto [94] and later gained popularity due to its characteristics of anonymity and decentralized design [2]. Since Bitcoin is based on cryptography tools (Proof-of-Work mechanism), Bitcoin is a type of cryptocurrencies. Even though it has large latencies (on the order of an hour), and the theoretical peak throughput is up to 7 transactions per second [130], Bitcoin is still one of the most popular cryptocurrencies. Here, we briefly discuss its mechanism and relation with Byzantine consensus and BFT systems.

Bitcoin Mechanism The core of Bitcoin is called *Blockchain*, which is a peer-to-peer ledger system, and acts as a virtually centralized ledger that keeps track of all bitcoin transactions. A set of bitcoin transactions are recorded in blocks. Owners of bitcoins can generate new transactions by broadcasting signed blocks to the Bitcoin network.[8] Then, a process called *mining* confirms the transactions and includes the transactions to the Blockchain (the centralized ledger system). Essentially, *mining* is a (randomized) distributed consensus component that confirms pending transactions by including them in the Blockchain. To include a transaction block, a miner needs to solve a "proof-of-work" (POW) or "cryptographic puzzle". The main incentive mechanism for Bitcoin participants to maintain the Blockchain and to confirm new transactions is to reward the participants (or the miners) some bitcoins – the first miner that solves the puzzle receives a certain amount of bitcoins. The main reason that the mining process can be related to consensus is because each miner maintains the chain of blocks (Blockchain) at local storage, and the global state is consistent at all miners (eventually) – eventually, all fault-free miners will have the same Blockchain [94]. That is, anonymous Bitcoin participants need to agree on the total order of the transactions.

One important feature of the cryptocurrency system is to prevent the *double-spending attacks*, i.e., spending some money twice. The consistent global state (order of transactions) can be used to prevent double-spending attacks, since the attackers have no ability to reorganize the order of blocks (i.e., modify the Blockchain, the ledger system). In [94], Satoshi Nakamoto presented a simple analysis that showed with high probability and attackers' computation power less than 1/3 of total computation power, Bitcoin's participants maintain a total order of the transactions; hence, no double-spending attack is possible with high probability. However, the models under consideration were not well-defined and the analysis was not rigorous. Thus, significant effort has been devoted to formally prove the correctness of Bitcoin mechanism or improved the design and performance. Please refer to [95] for a thorough discussion. Below, we focus on the comparison of Bitcoin and Byzantine Consensus/BFT systems.

Comparison with Byzantine Consensus There are several differences between problem formulation of Byzantine consensus (as described in Section 1.1) and the assumptions of Bitcoin [59, 87, 94], such as in Bitcoin, (i) the number of participants is dynamic; (ii) participants are anonymous, and the participants cannot authenticate each other; (iii) as a result of (ii), participants have no way to identify the source of a received message; and (iv) the Bitcoin network is able to synchronize in the course of a round, i.e., the network communication delay is negligible compared to computation time.

It was first suggested by Nakamoto that Bitcoin's POW-based mechanism can be used to solve Byzantine consensus [93, 10]. However, the discussion is quite informal [93]. To the best of our knowledge, Miller and LaViola were the first one to formalize the suggestion and proposed a POW-based model to achieve Byzantine consensus when majority of participants are fault-free. However, the validity is only ensured with non-negligible probability (but not with over-whelming probability). Subsequently, Garay et al. [59] extracted and analyzed the core mechanism of Bitcoin [59], namely Bitcoin Backbone. They first identified and formalized two

---

[8] Here, we follow the convention of Bitcoin literature – Bitcoin network consists of all the anonymous participants in the Bitcoin system. Note that in previous sections, network means the communication network. Also, throughout the discussion, "Bitcoin" means the system, whereas, "bitcoin" means the virtual money.

properties of Bitcoin Backbone: (i) *common prefix property*: fault-free participants will possess a large common prefix of the blockchain, and (ii) *chain-quality property*: enough blocks in the blockchain are contributed by fault-free participants. Then, they presented a simple POW-based Byzantine consensus algorithm which is a variation of Nakamoto's suggestion [93], but satisfy agreement and validity assuming that the adversarys computation power (puzzle-solving power) is bounded by 1/3. Their algorithm can also be used to solve Byzantine consensus with strong validity [96]. Finally, they proposed a more complicated consensus protocol, which was proved to be secure assuming high network synchrony and that the adversarys computation power is strictly less than 1/2. In [59], Garay et al. focused on how to use Bitcoin-inspired mechanism to solve Byzantine consensus, and did not compare Bitcoin with BFT systems.

Comparison with BFT System Conceptually, BFT and Bitcoin have similar goals:

- *BFT*: clients requests are executed in a total order distributively, and
- *Bitcoin*: a total order of blocks are maintained by each participants distributively.

Therefore, it is interesting to compare BFT with Bitcoin as well. Below, we address fundamental differences between the two.

- *Formulation*: As discussed above, model assumptions for BFT are similar to the ones for Byzantine consensus, which are very different from the ones for Bitcoin. One major difference is the anonymous node identity. In BFT, the system environment is more well-controlled, and replica IDs are maintained and managed by the system administrators. In contrast, Bitcoin is a decentralized system where all the participants are anonymous. As a result, BFT systems can use many well-studied tools from the literature, e.g., atomic broadcast, and quorum-based mechanism, whereas, Bitcoin-related systems usually rely on POW (proof-of-work) or various cryptographic puzzles.
- *Features*: In [130], Marko Vukolic mentioned that the features of BFT and Bitcoin are at two opposite ends of the scalability/performance spectrum due to different application goals. Generally, speaking, BFT systems offer good performance (low latency and high throughput) for small number of replicas ($\leq$ 20 replicas), whereas, Bitcoin scales well ($\geq$ 1000 participants), but the latency is prohibitively high and throughput is limited.
- *Incentive*: In BFT system, every fault-free replica/client is assumed to follow the algorithm specification. However, in Bitcoin, participants may choose not to spend their computation power on solving puzzles; thus, there is a mechanism in Bitcoin to reward the mining process [94].
- *Correctness property*: As addressed in Section 3.3, BFT systems satisfy safety in asynchronous network and satisfy liveness when network is synchronous enough (in grace period). As shown in [94, 59], Bitcoin requires network synchronous enough for ensuring correctness (when network delay is negligible compared to computation time).

In [130], Marko Vukolic proposed an interesting research direction on finding the synergies between Bitcoin-related and BFT systems, since both systems have its limitations. On one hand, the poor performance of POW-based mechanism limits the applicability of Blockchain in other domains like smart contract application [130, 27]. On the other hand, BFT systems are not widely adopted in practice due to their poorer scalability and lack of killer applications [81, 129]. SCP is one recent system that utilizes hybrid POW/BFT architecture [82].

4 Conclusion and Future Directions

4.1 Conclusion

Fault-tolerant consensus is a rich topic. This paper is only managed to sample a subset of recent results. To augment previous surveys/textbooks on the same topic [46, 104, 31, 84, 19], we focus on two angles: (i) new consensus problem formulations; and (ii) practical applications. For the second part, we focus on the Paxos- and Raft-based systems, and BFT systems. We also discuss Bitcoin which has close relation with Byzantine consensus and BFT systems.

4.2 Future Directions

The future directions focus on one main theme: *bridging the gap between theory and practice.* As discussed in the first part of paper, researchers have explored wide variety of different (theoretical) problem formulations; however, there is no consolidated or unified framework. As a result, it is often hard to compare different algorithms and models. Worse, it is even harder for practitioners to decide which algorithms or problem formulations to choose. Thus, making these results more coherent and more practical (e.g., giving rule-of-thumbs for picking algorithms) would be an important task.

In the second part, we discuss the efforts of applying fault-tolerant consensus in real systems. Unfortunately, the difficulty in implementing or even understanding the consensus algorithms prevents from wider applications. How to simplify the design and verify the implementation is also a key task. Raft [97] is one good example of how simplified design and explanation could help gain popularity. Another major task is to understand and analyze more thoroughly the popular distributed systems. As suggested in [130, 59], BFT systems and Bitcoin are not yet well-understood. The models presented in [59, 87] and other works mentioned in [130] are only the first step toward this goal. Only after enough research, could we improve the state-of-art mechanisms.

Acknowledgment

We would like to thank the anonymous reviewers for encouragement and suggestions.

References

[1] Apache zookeeper on windows azure. `https://msopentech.com/opentech-projects/apache-zookeeper-on-windows-azure-2/#`.

[2] Bitcoin.org. `https://bitcoin.org/en/`.

[3] Cassandra. `http://cassandra.apache.org/`.

[4] etcd. `https://github.com/coreos/etcd`.

[5] HBase. `http://hbase.apache.org/`.

[6] Leslie Lamport - A.M. Turing award winner. `http://amturing.acm.org/award_winners/lamport_1205376.cfm`.

[7] memcached. `http://memcached.org`.

[8] Raft. `https://raft.github.io/`.

[9] Rethinkdb. `https://www.rethinkdb.com/`.

[10] dugcampbell's blog `http://www.dugcampbell.com/byzantine-generals-problem/`, 07 2015.

[11] A. Mostefaoui, M. Raynal, and F. Tronel. From binary consensus to multivalued consensus in asynchronous message-passing systems. In *Information Processing Letters*, volume 73, pages 207–212. 2000.

[12] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-scalable byzantine fault-tolerant services. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, SOSP '05, pages 59–74, New York, NY, USA, 2005. ACM.

[13] M. Abd-El-Malek, G. R. Ganger, M. K. Reiter, J. J. Wylie, and G. R. Goodson. Lazy verification in fault-tolerant distributed storage systems. In *SRDS*, pages 179–190. IEEE Computer Society, 2005.

[14] I. Abraham, Y. Amit, and D. Dolev. Optimal resilience asynchronous approximate agreement. In *OPODIS*, pages 229–239, 2004.

[15] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Partial synchrony based on set timeliness. *Distributed Computing*, 25(3):249–260, 2012.

[16] D. Alistarh, S. Gilbert, R. Guerraoui, and C. Travers. How to solve consensus in the smallest window of synchrony. In *Proceedings of the 22Nd International Symposium on Distributed Computing*, DISC '08, pages 32–46, Berlin, Heidelberg, 2008. Springer-Verlag.

[17] D. Alistarh, S. Gilbert, R. Guerraoui, and C. Travers. Of choices, failures and asynchrony: The many faces of set agreement. *Algorithmica*, 62(1):595–629, 2012.

[18] H. Attiya and F. Ellen. *Impossibility Results for Distributed Computing.* Morgan & Claypool, June 2014.

[19] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* Wiley Series on Parallel and Distributed Computing, 2004.

[20] D. Behrens, C. Fetzer, F. P. Junqueira, and M. Serafini. Towards transparent hardening of distributed systems. In *Proceedings of the 9th Workshop on Hot Topics in Dependable Systems*, HotDep '13, pages 4:1–4:6, New York, NY, USA, 2013. ACM.

[21] D. Behrens, S. Weigert, and C. Fetzer. Automatically tolerating arbitrary faults in non-malicious settings. *Dependable Computing, Latin-American Symposium on*, 0:114–123, 2013.

[22] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 31–46, New York, NY, USA, 2011. ACM.

[23] M. Biely, P. Robinson, and U. Schmid. Agreement in directed dynamic networks. In *Structural Information and Communication Complexity*, volume 7355 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin Heidelberg, 2012.

[24] M. Biely, P. Robinson, U. Schmid, M. Schwarz, and K. Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. *CoRR*, abs/1408.0620, 2014.

[25] M. Biely, U. Schmid, and B. Weiss. Synchronous consensus under hybrid process and link failures. *Theor. Comput. Sci.*, 412(40):5602–5630, Sept. 2011.

[26] M. Biely, J. Widder, B. Charron-Bost, A. Gaillard, M. Hutle, and A. Schiper. Tolerating corrupted communication. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, pages 244–253, New York, NY, USA, 2007. ACM.

[27] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121, May 2015.

[28] Z. Bouzid, A. Mostfaoui, and M. Raynal. Minimal synchrony for byzantine consensus. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 461–470, New York, NY, USA, 2015. ACM.

[29] F. Brasileiro, F. Greve, A. Mostefaoui, and M. Raynal. *Consensus in One Communication Step*, chapter Parallel Computing Technologies: 6th International Conference, PaCT 2001 Novosibirsk, Russia, September 3–7, 2001 Proceedings, pages 42–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[30] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 335–350, Berkeley, CA, USA, 2006. USENIX Association.

[31] C. Cachin. *State Machine Replication with Byzantine Faults*, chapter Replication: Theory and Practice, pages 169–184. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[32] C. Cachin, D. Dobre, and M. Vukolic. Bft storage with 2t+1 data replicas. *CoRR*, abs/1305.4868, 2013.

[33] C. Cachin, D. Dobre, and M. Vukolić. *Separating Data and Control: Asynchronous BFT Storage with $2t+1$ Data Replicas*, pages 1–17. Springer International Publishing, Cham, 2014.

[34] C. Cachin, R. Haas, and M. Vukolic. Dependable storage in the intercloud. research report rz 3783. Technical report, Research Report RZ 3783, IBM Research, Aug. 2010.

[35] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas. Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 143–157, New York, NY, USA, 2011. ACM.

[36] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[37] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live: An engineering perspective. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, pages 398–407, New York, NY, USA, 2007. ACM.

[38] B. Charron-Bost, M. Függer, and T. Nowak. Approximate consensus in highly dynamic networks. *CoRR*, abs/1408.0620, 2014.

[39] B. Charron-Bost, M. Függer, and T. Nowak. Approximate consensus in highly dynamic networks: The role of averaging algorithms. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 528–539, 2015.

[40] B. Charron-Bost and A. Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

[41] S. Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132 – 158, 1993.

[42] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche. Upright cluster services. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 277–290, New York, NY, USA, 2009. ACM.

[43] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'09, pages 153–168, Berkeley, CA, USA, 2009. USENIX Association.

[44] J. C. Corbett et al. Spanner: Google's globally-distributed database. In *Proc. USENIX Conference on Operating Systems Design and Implementation (OSDI)*, pages 251–264, 2012.

[45] M. Correia, D. G. Ferro, F. P. Junqueira, and M. Serafini. Practical hardening of crash-tolerant systems. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 41–41, Berkeley, CA, USA, 2012. USENIX Association.

[46] M. Correia, G. S. Veronese, N. F. Neves, and P. Veríssimo. Byzantine consensus in asynchronous message-passing systems: a survey. *IJCCBS*, 2(2):141–161, 2011.

[47] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. Hq replication: A hybrid quorum protocol for byzantine fault tolerance. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 177–190, Berkeley, CA, USA, 2006. USENIX Association.

[48] H. Cui, R. Gu, C. Liu, T. Chen, and J. Yang. Paxos made transparent. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 105–120, New York, NY, USA, 2015. ACM.

[49] R. de Prisco, D. Malkhi, and M. Reiter. On k-set consensus problems in asynchronous systems. *IEEE Trans. Parallel Distrib. Syst.*, 12(1):7–21, Jan. 2001.

[50] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *Proc. ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 205–220, 2007.

[51] D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1), March 1982.

[52] D. Dolev and C. Lenzen. Early-deciding consensus is expensive. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 270–279, New York, NY, USA, 2013. ACM.

[53] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33:499–516, May 1986.

[54] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr. 1988.

[55] A. D. Fekete. Asymptotically optimal algorithms for approximate agreement. In *Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, PODC '86, pages 73–87, New York, NY, USA, 1986. ACM.

[56] M. J. Fischer, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. In *Proceedings of the fourth annual ACM symposium on Principles of distributed computing*, PODC '85, pages 59–70, New York, NY, USA, 1985. ACM.

[57] M. Fitzi and M. Hirt. Optimally efficient multi-valued byzantine agreement. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 163–168, New York, NY, USA, 2006. ACM.

[58] R. Friedman, A. Mostéfaoui, and M. Raynal. Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. *IEEE Trans. Dependable Sec. Comput.*, 2(1):46–56, 2005.

[59] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 281–310, 2015.

[60] G. R. Goodson, J. J. Wylie, G. R. Ganger, and M. K. Reiter. Efficient byzantine-tolerant erasure-coded storage. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, DSN '04, pages 135–, Washington, DC, USA, 2004. IEEE Computer Society.

[61] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić. The next 700 bft protocols. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 363–376, New York, NY, USA, 2010. ACM.

[62] A. Haeberlen, P. Kouznetsov, and P. Druschel. Peerreview: Practical accountability for distributed systems. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 175–188, New York, NY, USA, 2007. ACM.

[63] M. Hamouma, A. Mostefaoui, and G. Trédan. *Byzantine Consensus with Few Synchronous Links*, chapter Principles of Distributed Systems: 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings, pages 76–89. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[64] J. Hendricks, G. R. Ganger, and M. K. Reiter. Low-overhead byzantine fault-tolerant storage. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 73–86, New York, NY, USA, 2007. ACM.

[65] C. Ho, R. V. Renesse, M. Bickford, and D. Dolev. Nysiad: Practical protocol transformation to tolerate byzantine failures. In *Proceedings of the tth USENIX Symposium on Networked Systems Design and Implementation*, 2008.

[66] Y. Hu, H. C. H. Chen, P. P. C. Lee, and Y. Tang. Nccloud: Applying network coding for the storage repair in a cloud-of-clouds. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, pages 21–21, Berkeley, CA, USA, 2012. USENIX Association.

[67] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[68] Z. István, D. Sidler, G. Alonso, and M. Vukolic. Consensus in a box: Inexpensive coordination in hardware. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pages 425–438, Berkeley, CA, USA, 2016. USENIX Association.

[69] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel. Cheapbft: Resource-efficient byzantine fault tolerance. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 295–308, New York, NY, USA, 2012. ACM.

[70] I. Keidar and S. Rajsbaum. A simple proof of the uniform consensus synchronous lower bound. *Inf. Process. Lett.*, 85(1):47–52, Jan. 2003.

[71] V. King and J. Saia. Breaking the o(n2) bit barrier: Scalable byzantine agreement with an adaptive adversary. *J. ACM*, 58(4):18:1–18:24, July 2011.

[72] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 45–58, New York, NY, USA, 2007. ACM.

[73] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[74] L. Lamport. Paxos made simple. *SIGACT News*, 32(4):51–58, Dec. 2001.

[75] L. Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, 2006.

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

[77] H. LeBlanc and X. Koutsoukos. Consensus in networked multi-agent systems with adversaries. *14th International conference on Hybrid Systems: Computation and Control (HSCC)*, 2011.

[78] H. LeBlanc and X. Koutsoukos. Low complexity resilient consensus in networked multi-agent systems with adversaries. *15th International conference on Hybrid Systems: Computation and Control (HSCC)*, 2012.

[79] H. LeBlanc, H. Zhang, X. Koutsoukos, and S. Sundaram. Resilient asymptotic consensus in robust networks. *IEEE Journal on Selected Areas in Communications: Special Issue on In-Network Computation*, 31:766–781, April 2013.

[80] G. Liang and N. Vaidya. Error-free multi-valued consensus with byzantine failures. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '11, pages 11–20, New York, NY, USA, 2011. ACM.

[81] S. Liu, C. Cachin, V. Quéma, and M. Vukolic. XFT: practical fault tolerance beyond crashes. *CoRR*, abs/1502.05831, 2015.

[82] L. Luu, V. Narayanan, K. Baweja, C. Zheng, S. Gilbert, and P. Saxena. Scp: A computationally-scalable byzantine consensus protocol for blockchains. Technical report, National University of Singapore, 2015.

[83] N. Lynch, M. Fischer, and R. Fowler. "simple and efficient byzantine generals algorithm.". In *"Proceedings - Symposium on Reliability in Distributed Software and Database Systems"*, pages 46–52. IEEE, 1982.

[84] N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[85] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish. Depot: Cloud storage with minimal trust. *ACM Trans. Comput. Syst.*, 29(4):12:1–12:38, Dec. 2011.

[86] H. Mendes and M. Herlihy. Multidimensional approximate agreement in Byzantine asynchronous systems. In *STOC '13*, 2013.

[87] A. Miller and J. Joseph J. LaViola. Anonymous byzantine consensus from anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. Technical report, University of Central Florida, 2012.

[88] Z. Milosevic, M. Hutle, and A. Schiper. Tolerating permanent and transient value faults. *Distributed Computing*, 27(1):55–77, 2014.

[89] I. Moraru, D. G. Andersen, and M. Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 358–372, New York, NY, USA, 2013. ACM.

[90] I. Moraru, D. G. Andersen, and M. Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 358–372, New York, NY, USA, 2013. ACM.

[91] I. Moraru, D. G. Andersen, and M. Kaminsky. Paxos quorum leases: Fast reads without sacrificing writes. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 22:1–22:13, New York, NY, USA, 2014. ACM.

[92] A. Mostefaoui, H. Moumen, and M. Raynal. Signature-free asynchronous binary byzantine consensus with $t < n/3, o(n^2)$ messages, and $o(1)$ expected time. *J. ACM*, 62(4):31:1–31:21, sep 2015.

[93] S. Nakamoto. he proof-of-work chain is a solution to the byzantine generals' problem. In *The Cryp- tography Mailing List,* *`tps://www.mail-archive.com/cryptography@metzdowd.com/msg09997.html`*, November 2008.

[94] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* `bitcoin.org`, October 2008.

[95] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and Cryptocurrency Technologies.* Princeton University Presss, 2016.

[96] G. Neiger. Distributed consensus revisited. *Inf. Process. Lett.*, 49(4):195–201, Feb. 1994.

[97] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, June 2014. USENIX Association.

[98] P. R. Parvédy and M. Raynal. Optimal early stopping uniform consensus in synchronous systems with process omission failures. In *Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '04, pages 302–310, New York, NY, USA, 2004. ACM.

[99] A. Patra. *Principles of Distributed Systems: 15th International Conference, OPODIS 2011, Toulouse, France, December 13-16, 2011. Proceedings*, chapter Error-free Multi-valued Broadcast and Byzantine Agreement with Optimal Communication Complexity, pages 34–49. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[100] A. Patra and C. P. Rangan. *Progress in Cryptology – LATINCRYPT 2010: First International Conference on Cryptology and Information Security in Latin America, Puebla, Mexico, August 8-11, 2010, proceedings*, chapter Communication Optimal Multi-valued Asynchronous Broadcast Protocol, pages 162–177. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[101] A. Patra and C. P. Rangan. *Information Theoretic Security: 5th International Conference, ICITS 2011, Amsterdam, The Netherlands, May 21-24, 2011. Proceedings*, chapter Communication Optimal Multi-valued Asynchronous Byzantine Agreement with Optimal Resilience, pages 206–226. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[102] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, Apr. 1980.

[103] M. O. Rabin. Randomized byzantine generals. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, SFCS '83, pages 403–409, Washington, DC, USA, 1983. IEEE Computer Society.

[104] M. Raynal. Consensus in synchronous systems: a concise guided tour. In *Dependable Computing, 2002. Proceedings. 2002 Pacific Rim International Symposium on*, pages 221–228, Dec 2002.

[105] B. Reed and F. P. Junqueira. A simple totally ordered broadcast protocol. In *Proceedings of the 2Nd Workshop on Large-Scale Distributed Systems and Middleware*, LADIS '08, pages 2:1–2:6, New York, NY, USA, 2008. ACM.

[106] R. Rodrigues, P. Kouznetsov, and B. Bhattacharjee. Large-scale byzantine fault tolerance: Safe but not always live. In *Proceedings of the 3rd Workshop on on Hot Topics in System Dependability*, HotDep'07, Berkeley, CA, USA, 2007. USENIX Association.

[107] N. Santoro and P. Widmayer. Time is not a healer. In *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science on STACS 89*, pages 304–313, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[108] N. Santoro and P. Widmayer. Agreement in synchronous networks with ubiquitous faults. *Theor. Comput. Sci.*, 384(2-3):232–249, Oct. 2007.

[109] U. Schmid, B. Weiss, and I. Keidar. Impossibility results and lower bounds for consensus under link failures. *SIAM J. Comput.*, 38(5):1912–1951, Jan. 2009.

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

[111] M. Schwarz, K. Winkler, U. Schmid, M. Biely, and P. Robinson. Brief announcement: Gracefully degrading consensus and k-set agreement under dynamic link failures. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 341–343, New York, NY, USA, 2014. ACM.

[112] S. Sen, W. Lloyd, and M. J. Freedman. Prophecy: Using history for high-throughput fault tolerance. In *NSDI*, pages 345–360. USENIX Association, 2010.

[113] M. Serafini, P. Bokor, D. Dobre, M. Majuntke, and N. Suri. Scrooge: Reducing the costs of fast byzantine replication in presence of unresponsive replicas. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 353–362, 2010.

[114] A. Singh, P. Fonseca, P. Kuznetsov, R. Rodrigues, and P. Maniatis. Zeno: Eventually consistent byzantine-fault tolerance. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'09, pages 169–184, Berkeley, CA, USA, 2009. USENIX Association.

[115] Y. J. Song and R. Renesse. *Distributed Computing: 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings*, chapter Bosco: One-Step Byzantine Asynchronous Consensus, pages 438–450. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[116] L. Su and N. Vaidya. Reaching approximate Byzantine consensus with multi-hop communication. In A. Pelc and A. A. Schwarzmann, editors, *Stabilization, Safety, and Security of Distributed Systems*, volume 9212 of *Lecture Notes in Computer Science*, pages 21–35. Springer International Publishing, 2015.

[117] S. Sundaram and C. Hadjicostis. Distributed function calculation and consensus using linear iterative strategies. *Selected Areas in Communications, IEEE Journal on*, 26(4):650–660, May 2008.

[118] S. Sundaram, S. Revzen, and G. Pappas. A control-theoretic approach to disseminating values and overcoming malicious links in wireless networks. *Automatica*, 48(11):2894–2901, Nov. 2012.

[119] M. Trencseni, A. Gazsó, and H. Reinhardt. Paxoslease: Diskless paxos for leases. *CoRR*, abs/1209.4187, 2012.

[120] L. Tseng. *Fault-Tolerant Consensus in Directed Graphs and Convex Hull Consensus*. PhD thesis, University of Illinois at Urbana-Champaign, 2016.

[121] L. Tseng and N. H. Vaidya. Asynchronous convex hull consensus in the presence of crash faults. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 396–405, New York, NY, USA, 2014. ACM.

[122] L. Tseng and N. H. Vaidya. Iterative approximate consensus in the presence of Byzantine link failures. In *Networked Systems - Second International Conference, NETYS 2014, Marrakech, Morocco, May 15-17, 2014. Revised Selected Papers*, pages 84–98, 2014.

[123] L. Tseng and N. H. Vaidya. Fault-tolerant consensus in directed graphs. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 451–460, New York, NY, USA, 2015. ACM.

[124] R. Turpin and B. A. Coan. Extending binary byzantine agreement to multivalued byzantine agreement. In *Information Processing Letters*, volume 18, pages 73–76. 1984.

[125] N. H. Vaidya. Iterative Byzantine vector consensus in incomplete graphs. In *In International Conference on Distributed Computing and Networking (ICDCN)*, January 2014.

[126] N. H. Vaidya and V. K. Garg. Byzantine vector consensus in complete graphs. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 65–73, New York, NY, USA, 2013. ACM.

[127] N. H. Vaidya, L. Tseng, and G. Liang. Iterative approximate Byzantine consensus in arbitrary directed graphs. In *Proceedings of the thirty-first annual ACM symposium on Principles of distributed computing*, PODC '12. ACM, 2012.

[128] G. Veronese, M. Correia, A. Bessani, L. C. Lung, and P. Verissimo. Efficient byzantine fault-tolerance. *Computers, IEEE Transactions on*, 62(1):16–30, 2013.

[129] M. Vukolić. The byzantine empire in the intercloud. *SIGACT News*, 41(3):105–111, Sept. 2010.

[130] M. 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.

[131] Y. Wang, M. Kapritsos, Z. Ren, P. Mahajan, J. Kirubanandam, L. Alvisi, and M. Dahlin. Robustness in the salus scalable block store. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 357–370, Berkeley, CA, USA, 2013. USENIX Association.

[132] T. Wood, R. Singh, A. Venkataramani, P. Shenoy, and E. Cecchet. Zz and the art of practical bft execution. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 123–138, New York, NY, USA, 2011. ACM.

[133] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for byzantine fault tolerant services. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 253–267, New York, NY, USA, 2003. ACM.

[134] H. Zhang and S. Sundaram. Robustness of complex networks with implications for consensus and contagion. In *Proceedings of CDC 2012, the 51st IEEE Conference on Decision and Control*, 2012.

[135] J. Zhang and W. Chen. Bounded cost algorithms for multivalued consensus using binary consensus instances. *Information Processing Letters*, 109(17):1005 – 1009, 2009.