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

Lewis Tseng(✉)

Coordinated Science Laboratory, Department of Computer Science,
University of Illinois at Urbana-Champaign, Urbana, USA
ltseng3@illinois.edu

Abstract. Fault-tolerant consensus has been studied extensively in the literature, because it is one of the important distributed primitives and has wide applications in practice. This paper surveys important works on fault-tolerant consensus in message-passing networks, and the focus is on results from the past decade. Particularly, we categorize the results into two groups: new problem formulations and practical applications. In the first part, we discuss new ways to define the consensus problem, which include larger input domains, enriched correctness properties, different network models, etc. In the second part, we focus on real-world systems that use Paxos or Raft to reach consensus, and Byzantine Fault-Tolerant (BFT) systems. We also discuss Bitcoin, which can be related to solving Byzantine consensus in anonymous systems, and compare Bitcoin with BFT systems and Byzantine consensus algorithms.

Keywords: Consensus · Paxos · Bitcoin · BFT · Byzantine · Crash

1 Introduction

Fault-tolerant consensus has received significant attentions over the past three decades [14,50] since the seminal work by Lamport, Shostak, and Pease [43,67] – some important results include solving consensus in an optimal way and identifying bounds on time and communication complexity under different models – please refer to [14,50,70] for these fundamental results. In this paper, we survey recent efforts on fault-tolerant consensus in message-passing networks, with the focus on results from the past decade. References [18,26,69] presented early surveys on the topic. To complement theses prior surveys, our paper focus on the following two directions:

– *Exploration of new problem formulations*: Lots of different consensus problems have been proposed in the past ten years in order to solve more complicated tasks and accommodate different system and network requirements. New problem formulations include enriched correctness properties, different fault models, different communication networks, and different input/output domains. For this part, we focus on the comparison of recently proposed problem formulations and relevant techniques.

J. Suomela (Ed.): SIROCCO 2016, LNCS 9988, pp. 92–108, 2016.
DOI: 10.1007/978-3-319-48314-6_7

– *Exploration of practical applications*: Consensus has been applied in many practical systems. Here, we focus on three types of applications: (i) crash-tolerant consensus algorithms (mainly Paxos [40] and Raft [63]) and their applications in real-world systems, (ii) Practical Byzantine Fault-Tolerance (PBFT) [20] and subsequent works on improving PBFT, and (iii) Bitcoin [2] and its comparison with Byzantine consensus algorithms and Byzantine Fault-Tolerance (BFT) systems.

For lack of space, discussions on some results are omitted here. Further details can be found in [77].

Classic Problem Formulations 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 [14,50], 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:

– *Exact* [40,67]: fault-free nodes have to agree on exactly the *same* output.
– *Approximate* [30,32]: 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.
– *k-set* [22,68]: the number of distinct outputs at fault-free nodes is $\leq k$.

Validity property is also required for consensus algorithms to produce meaningful output(s), since the property defines the acceptable relationship between inputs and output(s). Some popular validity properties include: (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 inputs at fault-free nodes. A consensus algorithm is said to be correct if it satisfies termination, agreement and validity properties given that enough number of nodes are fault-free throughout the execution of the algorithm. In this paper, we focus on three types of node failures – Byzantine, crash, and omission faults.

The other key component of the consensus problem formulation is *system synchrony*, i.e., a model specifying the relative speed of nodes and the network delay. There are also three main categories [14,16,31,50]:

– *Synchronous*: each node proceeds in a lock-step fashion, and there is a known upper bound on the network delay.

– *Partially synchronous*: there exists a *partially synchronous* period from time to time. In such a period, fault-free nodes and the network stabilize and behave (more) synchronously.[1]
– *Asynchronous*: no known bound exists on nodes' processing speed or the network delay.

2 Exploration of New Problem Formulations

In the past decade, researchers proposed many new consensus problems to handle more complicated tasks and/or environments. We categorize these efforts into four groups: (i) input/output domain, (ii) communication network and synchrony assumptions, (iii) link fault models, and (iv) enriched correctness properties, such as early-stopping and one-step properties. In this survey, we focus on the discussion of works on different input/output domain and communication networks. Please refer to [77] for works in other groups. In this section, we assume that the system consists of n nodes, and up to f of them may crash or become Byzantine faulty. Byzantine faulty nodes may have an arbitrary behavior.

2.1 Input/Output Domain

Multi-valued Consensus. In the original *exact* Byzantine consensus problem [43,67], both input and output are binary values. Later, references [51,80] 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. Perhaps a bit surprisingly, for L-bit inputs, 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 [35]. Their algorithm is based on the reduction technique and has the following steps: (i) hash the inputs to much smaller values using universal hash function, (ii) apply (classic) Byzantine consensus algorithm using these hash values as inputs, and (iii) achieve consensus by obtaining the input value from nodes that have the same hash values (if there is enough number of such nodes) [35]. 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 [47]. One key contribution is to introduce a lightweight fault detection (or fault diagnosis) mechanism using coding [47]. Their coding-based fault diagnosis is efficient for large inputs because the inputs are divided into batches of small values, and in each batch, either consensus (on the small value

[1] Note that there are also other definitions of partial synchrony. We choose to present this particular definition, since many BFT systems only satisfy liveness under this particular definition. Please refer to [12,31] for more models on partial synchrony.

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 nodes is bounded, consensus on most batches can be achieved with small communication complexity [47].

Subsequently, variants of reduction technique were applied to solve consensus problems with large inputs in asynchronous systems. References [65,66] provided multi-valued algorithms with small error probability. Afterwards, Patra improved the results and proposed an error-free algorithm [64]. 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 using variants of fault diagnosis mechanisms.

Typically, to achieve optimal communication complexity, the number of batches is in the same order of L. Consequently, the number of messages is large, since by assumption, L is a large value (compared with n). Instead of achieving optimal bit complexity, Mostéfaoui and Raynal focused on a different goal – minimizing number of messages in asynchronous systems [56,58]. Their algorithm relies on two new all-to-all communication abstractions, which have an $O(n^2)$ message complexity (i.e., $O(n^2L)$ bits) and a constant time complexity. The first communication abstraction allows the fault-free to reduce the number of input values to a small constant c, which ranges from 3 to 6 depending on the bound on the number of faulty nodes. The second abstraction allows each fault-free node to obtain a set of inputs such that, if the set at a fault-free node contains a single value, then this value belongs to the set at any other fault-free nodes. The algorithm in [56,58] consists of four phases: (i) nodes exchange input values in the first three phases with the first phase based on the first communication abstraction, and the two subsequent phases based on the second, and (ii) nodes use binary consensus in the final phase to determine whether it is safe to agree on the value learned from phase 3.

Multi-valued consensus has also been studied under the crash fault model. Mostéfaoui et al. proposed multi-valued consensus algorithms in both synchronous and asynchronous systems [9]. Later, Zhang and Chen proposed a more efficient multi-valued consensus algorithm in asynchronous systems [90].

High-Dimensional Input/Output. In the Byzantine vector consensus (or multi-dimensional consensus) [52,82], 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 [52] and Vaidya and Garg [82] addressed approximate vector consensus in the presence of Byzantine faults. These papers yielded lower bounds on the number of nodes, and algorithms with optimal resilience in asynchronous [52,82] as well as synchronous systems [82]. The algorithms in [52,82] are generalizations of the optimal iterative approximate Byzantine consensus for scalar inputs in asynchronous systems [11]. The algorithms in [52,82] require sub-routines for geometric computation in the d-dimensional space to obtain each node's local state in each iteration, whereas, a simple average operation suffices when $d = 1$ (i.e., classic approximate

consensus) [11]. These two papers [52] and [82] independently addressed the same problem, and developed different algorithms – mainly on different geometric computation techniques – which also resulted in different proofs.

Subsequent work by Vaidya [81] explored the approximate vector consensus problem in incomplete *directed* graphs. Later, Tseng and Vaidya [78] 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 [78] bears some similarity to the ones in [11,52,82]; 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 show the correctness of iterative consensus algorithms when the output is a polytope [78].

2.2 Communication Network

The fault-tolerant consensus problem has been studied extensively in complete networks (e.g., [14,30,40,50,67]) and in undirected networks (e.g., [29,33]). 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 the assumptions on the communication network and enriched the problem space in four main directions: directed graphs, dynamic graphs, unknown/anonymous networks and partial synchrony. Here, we focus on the works on directed graphs. Please find the discussion on the later three directions in [77].

Directed Graphs. Researchers started to explore various consensus problems 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. Significant 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. [83] proved *tight* conditions for achieving approximate Byzantine consensus in synchronous and asynchronous systems using *iterative* algorithms. The tight condition for achieving approximate crash-tolerant consensus using iterative algorithms in asynchronous systems was also proved in [76].

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., [44–46,89]. LeBlanc and Koutsoukos [45] addressed a continuous time version of the consensus problem with malicious faults in complete graphs. LeBlanc et al. [44] have obtained *tight* necessary and sufficient conditions for tolerating up to f faults in the network.

The aforementioned approximate algorithms (e.g., [44,74,83]) are generalizations of the iterative approximate consensus algorithm in complete network [30,32]. However, to accommodate directed links, the proofs are more involved. Particularly, for the sufficiency part, one has to prove that all fault-free nodes must be able to receive a non-trivial amount of a state at some fault-free node in finite number of iterations. The necessity proofs in the work on directed graphs (e.g., [44,83]) are generalizations of the indistinguishability proof [13,33]. 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 certain faulty behaviors for up to f nodes such that different groups of fault-free nodes have to agree on different outputs.

There were also works on using general algorithms to achieve consensus – an algorithm is *general* if nodes are allowed to have topology knowledge and the ability to route messages (send and receive messages using multiple node-disjoint paths). Furthermore, unlike iterative algorithms (e.g., [11,30]), the state maintained at each node in general algorithms is not constrained to a single value. Tseng and Vaidya [79] 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. The tight condition for achieving approximate Byzantine consensus in asynchronous systems remains open. Lili and Vaidya [74] proved tight conditions for achieving approximate Byzantine consensus using general algorithms.

The exact consensus algorithms in [79] require that some "common information" has to be propagated to all fault-free nodes even if some nodes may fail. Generally speaking, the algorithms in [79] proceed 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, because some "common information" are guaranteed to be received by all nodes that have not failed yet. The algorithm in [74] can be viewed as an extension of the iterative algorithm that tolerates Byzantine faults in directed networks [83], and it utilized the routing information and network knowledge to tolerate more failures than the algorithm in [83] does.

3 Exploration of Practical Applications

Fault-tolerant consensus has been adopted in many practical systems. We start with real-world systems that are designed to tolerate crash node faults, particularly, those based on two families of algorithms – Paxos [40] and Raft [63]. Then, we discuss efforts on designing BFT (Byzantine Fault-Tolerance) systems. Finally, we compare Bitcoin-related work [60] with BFT systems and Byzantine consensus. In [77], we also discuss systems tolerating "arbitrary state corruption faults".

3.1 Paxos and Raft

Here, we discus exact consensus algorithms developed for asynchronous systems. Consensus algorithm needs to satisfy validity, agreement and termination as discussed in Sect. 1. However, it is impossible to achieve exact consensus in asynchronous systems [34]. Hence, the termination property is relaxed – progress (or liveness) is only ensured when there exist some time periods that enough messages are received within time.

Paxos [40–42,54] is the well-known family of consensus protocols tolerating crash node faults in asynchronous systems. Since Paxos was first proposed by Lamport [40,41], variants of Paxos were developed and implemented in real-world systems, such as Chubby lock service used in many Google systems [17,25], and membership management in Windows Azure [19].[2] Yahoo! also developed ZaB [71], a protocol achieving atomic broadcast in network equipped with FIFO channels, and used ZaB to build the widely-adopted coordination service, ZooKeeper [38]. ZooKeeper is later used in many practical storage systems, like HBase [4] and Salus [86]. Recently, many novel mechanisms have been proposed to improve the performance of Paxos, including quorum lease [55], diskless Paxos [75], even load balancing [54], and time bubbling (for handling nondeterministic network input timing) [28]. While the original Paxos [40,41] is theoretically elegant, practitioners have found it hard to implement Paxos in practice [21]. One difficulty mentioned in [21] is that membership/configuration management is non-trivial in practice, especially, when Multi-Paxos, and disk corruptions are considered. (Multi-Paxos is a generalization of Paxos which is designed to optimize the performance when there are multiple inputs to be agree upon [21].)

In 2014, Ongaro and Ousterhout from Stanford proposed a new consensus algorithm – Raft [63]. 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 [63]: "*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 element deeply with leader election protocol and membership/configuration management protocol [63]. After their publication, Raft has quickly gained popularity, and been used in practical key-value store systems such as etcd [3] and RethinkDB [7]. Please refer to their website [6] for a list of papers and implementations.

3.2 Byzantine Fault Tolerance (BFT)

Generally speaking, Byzantine Fault-Tolerance (BFT) systems implement deterministic state machines over different machines (or *replicas*) to tolerate Byzantine node failures. In other words, BFT systems realize the State Machine Replication systems [72] that tolerate Byzantine faults. The main challenge is to design

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

a system such that it behaves like a centralized server to the clients in the presence of Byzantine faults. More precisely, the system is given requests from the clients, and the goals of a BFT system 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, safety is guaranteed at all time, and liveness is guaranteed only in the *grace periods*, i.e., when messages are delivered in time.

Since Castro and Liskov published their seminal work PBFT (Practical Byzantine Fault-Tolerance) [20], significant efforts have been devoted to improving BFT systems. There were mainly two directions of the improvements: (i) reducing the overhead like communication costs, or replication costs, and (ii) providing higher throughput or lower latency (in the form of round complexity). Below, we focus on different techniques for improving the performance. Please refer to [77] for the discussions on other works in this area, including hardening existing crash-fault-tolerant systems, hardware-based BFTs, BFTs with relaxed properties, BFT storage systems, and BFTs over intercloud.

Improving Performance. Castro and Liskov's work on Practical Byzantine Fault-Tolerance (PBFT) showed for the first time that BFT system is useful in practice [20]. PBFT requires $3f + 1$ replicas, where f is the upper bound on the number of Byzantine nodes in the system. Subsequently, Quorum-based solutions Q/U [10] and HQ [27] have been proposed, which only require one round of communication in contention-free case by allowing clients directly interact with all the replicas to agree on an execution order. Contention-free case means the time when all the following conditions hold: (i) no replica fails, (ii) the network has stable performance, and (iii) there is no contention on the proposed input value. The quorum-based solutions reduce latency (number of rounds) in some cases, but was shown to be more expensive in other cases [39]. Hence, Zyzzyva [39] focuses on increasing performance in failure-free case (when no replica fails) by allowing speculative operations that increase throughput significantly and adopting a novel roll-back mechanism to recover operations when failures are detected. Zyzzyva requires $3f + 1$ replicas; however, a single crash failure would significantly reduce the performance by forcing Zyzzyva to run in the slow mode – where no speculative operation can be executed [39]. Thus, Kotla et al. also introduced Zyzzyva5, which can be executed in the fast mode even if there are crash failures, but Zyzzyva5 requires $5f + 1$ replicas [39]. Subsequently, Scrooge [73] reduces the replication cost to $4f$ by requiring the participation from clients which help detect replicas' misbehaviors. Moreover, Scrooge runs in the fast mode even if there are crashed nodes.

Clement et al. observed that a single Byzantine replica or client can significantly impact the performance of HQ, PBFT, Q/U and Zyzzyva [24]. Thus, they proposed a new system Aardvark, which provides good performance when Byzantine failures happen by sacrificing the performance in the failure-free case [24]. 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 due to the integration of speculative operations [23].

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 [37]. Their tunable design is useful, since it allows the system administrators to explore the performance tradeoff space. 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 [37], Guerraoui et al. showed how to construct new BFT systems with different parameters; particularly, they proposed (i) *AZyzzyva* which composes Zyzzyva and PBFT together to have more stable performance than Zyzzyva does and faster failure-free performance than PBFT's performance, and (ii) *Aliph* which has three components: PBFT, Quorum-based protocol optimized for contention-free case, and Chain-based protocol optimized for high-contention case without failures and asynchrony [37].

For computation-heavy workload, Yin et al. proposed a novel idea that separates agreement protocol from executions of clients' requests [88]. 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 [87]. 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 [87].

3.3 Bitcoin

Bitcoin is a digital currency system proposed by Satoshi Nakamoto [60] 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), it can be viewed as a cryptocurrency. Even though Bitcoin has large latencies (on the order of an hour), and the theoretical peak throughput is up to 7 transactions per second [85], Bitcoin is still one of the most popular cryptocurrencies. Here, we briefly discuss the core mechanism of Bitcoin and compare it 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.[3] Then, a procedure 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 procedure 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 – all fault-free miners will have the same Blockchain eventually [60]. That is, fault-free 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 the same unit of money twice. In Bitcoin, the consistent global state – the 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), the money recipient can simply check whether the money has already been spent in the Blockchain and reject the money if it has already been used.[4] In [60], Satoshi Nakamoto presented a simple analysis that showed with high probability, Bitcoin's participants maintain a total order of the transactions if adversary's computation power is less than 1/3 of the total computation power in the Bitcoin network. As a result, no double-spending attack is possible with high probability if adversary's computation power is bounded. However, the models under consideration were not well-defined and the analysis was not rigorous in [60]. Thus, significant efforts have been devoted to formally proving the correctness of Bitcoin mechanism or improving the design and performance. Please refer to a nice textbook [61] 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 the problem formulation of Byzantine consensus (as described in Sect. 1) and the assumptions of Bitcoin [36,53,60]. For example, 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

[3] Here, we follow the convention: (i) Bitcoin network includes all the anonymous participants in the Bitcoin system and the network that supports the anonymous communication; and (ii) throughout the discussion, "Bitcoin" refers to the system/network, whereas, "bitcoin" refers to the basic unit of the cryptocurrency.

[4] One technical issue here is that the Blockchain has the "eventually consistent" feature. The exact mechanism to handle the issue is beyond the scope of this survey. Please refer to a nice textbook [61] for some mechanisms.

is synchronized enough, and there is a notion 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 [8,59]. However, the discussion was quite informal [59]. To the best of our knowledge, Miller and LaViola were the first 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. [36] extracted and analyzed the core mechanism of Bitcoin [36], namely Bitcoin Backbone. They first identified and formalized two 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 [59], but satisfy agreement and validity assuming that the adversary's computation power (puzzle-solving power) is bounded by 1/3. Their algorithm can also be used to solve Byzantine consensus with strong validity [62]. Finally, they proposed a more complicated consensus protocol, which was proved to be secure assuming high network synchrony and that the adversary's computation power is strictly less than 1/2. In [36], Garay et al. focused on how to use Bitcoin-inspired mechanism to solve Byzantine consensus.

Comparison with BFT Systems. Conceptually, BFT and Bitcoin have similar goals: (i) *BFT*: clients' requests are executed in a total order distributively; and (ii) *Bitcoin*: a total order of blocks are maintained by participants distributively. Therefore, it is interesting to compare BFT with Bitcoin as well. Below, we address fundamental differences between the two.

- *Environment*: As discussed above, 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 well-controlled, and replicas' 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 variants of cryptographic tools.
- *Features*: In [85], 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 programmed 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 [60].
- *Correctness property*: As addressed in Sect. 3.2, BFT systems satisfy safety in asynchronous network and satisfy liveness when network is synchronous enough (in grace period). As shown in [36,60], Bitcoin requires network synchronous enough for ensuring correctness (when network delay is negligible compared to computation time).
- *Applications*: Bitcoin or Blockchain-based systems inspire lots of exciting applications beyond cryptocurrency, e.g., smart contract, identity/ownership management, digital access/contents, etc. In contrasts, applications for BFT systems are more traditional in the sense that there already exist those applications (that tolerate only crash faults), and BFTs help improve the fault-tolerance level.

In [85], Marko Vukolic proposed an interesting research direction on finding the synergies between Bitcoin and BFT systems, since both systems have their limitations and advantages. On one hand, the poor performance of POW-based mechanism limits the applicability of Blockchain in other domains like smart contract application [15,85]. On the other hand, BFT systems are not widely adopted in practice due to their poorer scalability and lack of killer applications [48,84]. SCP is a recent system that utilizes hybrid POW/BFT architecture [49]. However, further exploration of the synergy between Bitcoin and BFT systems is an interesting research direction.

4 Summary and Future Directions

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, e.g., [14,18,26,50,69], we survey prior works from 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 relationship with Byzantine consensus and BFT systems.

Future Directions. The future research directions below focus on one theme: *bridging the gap between theory and practice.* As discussed in the first part of the 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, and it is also difficult for practitioners to decide which algorithms are most appropriate to solve their problems. Thus, making these results more coherent and more practical (e.g., giving rule-of-thumbs for picking algorithms) would be an important and interesting task.

In the second part, we discuss the efforts of applying fault-tolerant consensus in real-world systems. Unfortunately, the difficulty in implementing or even

understanding the consensus algorithms prevents wider applications of consensus algorithms. Therefore, simplifying the conceptual design and verifying the implementation is also a key task. Raft [63] is one good example of how simplified design and explanation could help gain popularity and practicability. Another major task is to understand and analyze more thoroughly the real-world distributed systems. As suggested in [36,85], BFT systems and Bitcoin are not yet well-understood. The models presented in [36,53] and other works mentioned in [85] were only the first step toward this goal. Only after enough research and understanding, could we improve the state-of-art mechanisms. For example, as mentioned in [61], Bitcoin's core mechanism depends on the incentive mechanism to reward miners; however, not much work has analyzed Bitcoin from the perspective of game theory.

Acknowledgment. We would like to thank the anonymous reviewers for encouragement and suggestions. We also acknowledge Nitin H. Vaidya for early feedback and Michel Raynal for pointers to several new works.

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. etcd. https://github.com/coreos/etcd
4. HBase. http://hbase.apache.org/
5. Leslie Lamport - A.M. Turing award winner. http://amturing.acm.org/award_winners/lamport_1205376.cfm
6. Raft. https://raft.github.io/
7. Rethinkdb. https://www.rethinkdb.com/
8. Dugcampbell's blog, 07 2015. http://www.dugcampbell.com/byzantine-generals-problem/
9. Mostefaoui, A., Raynal, M., Tronel, F.: From binary consensus to multivalued consensus in asynchronous message-passing systems. IPL **73**(5), 207–212 (2000)
10. Abd-El-Malek, M., Ganger, G.R., Goodson, G.R., Reiter, M.K., Wylie, J.J.: Fault-scalable Byzantine fault-tolerant services. SOSP **39**(5), 59–74 (2005)
11. Abraham, I., Amit, Y., Dolev, D.: Optimal resilience asynchronous approximate agreement. In: Higashino, T. (ed.) OPODIS 2004. LNCS, vol. 3544, pp. 229–239. Springer, Heidelberg (2005). doi:10.1007/11516798_17
12. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Partial synchrony based on set timeliness. Distrib. Comput. **25**(3), 249–260 (2012)
13. Attiya, H., Ellen, F.: Impossibility results for distributed computing. Synth. Lect. Distrib. Comput. Theor. **5**(1), 1–162 (2014). Morgan & claypool
14. Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations, and Advanced Topics. Parallel and Distributed Computing. Wiley, Hoboken (2004)
15. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Sok: research perspectives and challenges for bitcoin and cryptocurrencies. In: 2015 IEEE Symposium on Security and Privacy, pp. 104–121, May 2015
16. Bouzid, Z., Mostfaoui, A., Raynal, M.: Minimal synchrony for Byzantine consensus. In: Symposium on Principles of Distributed Computing, PODC (2015)

17. Burrows, M.: The chubby lock service for loosely-coupled distributed systems. In: Proceedings of the Operating Systems Design and Implementation, OSDI (2006)
18. Cachin, C.: State machine replication with Byzantine faults. In: Charron-Bost, B., Pedone, F., Schiper, A. (eds.) Replication. LNCS, vol. 5959, pp. 169–184. Springer, Heidelberg (2010). doi:10.1007/978-3-642-11294-2_9
19. Calder, B., et al.: Windows azure storage: a highly available cloud storage service with strong consistency. In: SOSP (2011)
20. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: Proceedings of the Operating Systems Design and Implementation, OSDI (1099)
21. Chandra, T.D., Griesemer, R., Redstone, J.: Paxos made live: an engineering perspective. In: Symposium on Principles of Distributed Computing, PODC (2007)
22. Chaudhuri, S.: More choices allow more faults: set consensus problems in totally asynchronous systems. Inf. Comput. **105**(1), 132–158 (1993)
23. Clement, A., Kapritsos, M., Lee, S., Wang, Y., Alvisi, L., Dahlin, M., Riche, T.: Upright cluster services. In: SOSP (2009)
24. Clement, A., Wong, E., Alvisi, L., Dahlin, M., Marchetti, M.: Making Byzantine fault tolerant systems tolerate Byzantine faults. In: NSDI (2009)
25. Corbett, J.C., et al.: Spanner: Google's globally-distributed database. In: OSDI (2012)
26. Correia, M., Veronese, G.S., Neves, N.F., Veríssimo, P.: Byzantine consensus in asynchronous message-passing systems: a survey. IJCCBS **2**(2), 141–161 (2011)
27. Cowling, J., Myers, D., Liskov, B., Rodrigues, R., Shrira, L.: HQ replication: a hybrid quorum protocol for Byzantine fault tolerance. In: OSDI (2006)
28. Cui, H., Gu, R., Liu, C., Chen, T., Yang, J.: Paxos made transparent. In: SOSP (2015)
29. Dolev, D.: The Byzantine generals strike again. J. Algorithms **3**(1), 14–30 (1982)
30. Dolev, D., Lynch, N.A., Pinter, S.S., Stark, E.W., Weihl, W.E.: Reaching approximate agreement in the presence of faults. J. ACM **33**(3), 499–516 (1986)
31. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
32. Fekete, A.D.: Asymptotically optimal algorithms for approximate agreement. In: PODC (1986)
33. Fischer, M.J., Lynch, N.A., Merritt, M.: Easy impossibility proofs for distributed consensus problems. In: PODC (1985)
34. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**, 374–382 (1985)
35. Fitzi, M., Hirt, M.: Optimally efficient multi-valued Byzantine agreement. In: PODC (2006)
36. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46803-6_10
37. Guerraoui, R., Knežević, N., Quéma, V., Vukolić, M.: The next 700 bft protocols. In: EuroSys (2010)
38. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.: Zookeeper: wait-free coordination for internet-scale systems. In: USENIX ATC (2010)
39. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: speculative Byzantine fault tolerance. In: SOSP (2007)
40. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. **16**(2), 133–169 (1998)
41. Lamport, L.: Paxos made simple. SIGACT News **32**(4), 51–58 (2001)

42. Lamport, L.: Fast Paxos. Distrib. Comput. **19**(2), 79–103 (2006)
43. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
44. LeBlanc, H., Zhang, H., Koutsoukos, X., Sundaram, S.: Resilient asymptotic consensus in robust networks. IEEE J. Sel. Areas Commun. **31**(4), 766–781 (2013). Special Issue on In-Network Computation
45. LeBlanc, H., Koutsoukos, X.: Consensus in networked multi-agent systems with adversaries. In: HSCC (2011)
46. LeBlanc, H., Koutsoukos, X.: Low complexity resilient consensus in networked multi-agent systems with adversaries. In: HSCC (2012)
47. Liang, G., Vaidya, N.: Error-free multi-valued consensus with Byzantine failures. In: PODC (2011)
48. Liu, S., Cachin, C., Quéma, V., Vukolic, M.: XFT: practical fault tolerance beyond crashes. CoRR abs/1502.05831 (2015)
49. Luu, L., et al.: Scp: a computationally-scalable Byzantine consensus protocol for blockchains. National University of Singapore, Technical report (2015)
50. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, San Francisco (1996)
51. Lynch, N., Fischer, M., Fowler, R.: Simple and efficient Byzantine generals algorithm. In: Symposium on Reliability in Distributed Software and Database Systems (1982)
52. Mendes, H., Herlihy, M.: Multidimensional approximate agreement in Byzantine asynchronous systems. In: STOC (2013)
53. Miller, A., LaViola Jr., J.J.: Anonymous Byzantine consensus from anonymous Byzantine consensus from moderately-hard puzzles: a model for bitcoin. University of Central Florida, Technical report (2012)
54. Moraru, I., Andersen, D.G., Kaminsky, M.: There is more consensus in egalitarian parliaments. In: SOSP (2013)
55. Moraru, I., Andersen, D.G., Kaminsky, M.: Paxos quorum leases: fast reads without sacrificing writes. In: SOCC (2014)
56. Mostéfaoui, A., Raynal, M.: Signature-free asynchronous Byzantine systems: from multivalued to binary consensus with t < n/3, O(n2) messages, and constant time. In: Scheideler, C. (ed.) SIROCCO 2015. LNCS, vol. 9439, pp. 194–208. Springer, Switzerland (2015). doi:10.1007/978-3-319-25258-2_14
57. Mostéfaoui, A., Raynal, M.: Intrusion-tolerant broadcast and agreement abstractions in the presence of Byzantine processes. IEEE Trans. Parallel Distrib. Syst. **27**(4), 1085–1098 (2016). http://dx.doi.org/10.1109/TPDS.2015.2427797
58. Mostéfaoui, A., Raynal, M.: Signature-free asynchronous Byzantine systems: from multivalued to binary consensus with t < n/3, O(n2) messages, and constant time. Acta Informatica (2016). doi:10.1007/s00236-016-0269-y
59. Nakamoto, S.: the proof-of-work chain is a solution to the Byzantine generals' problem. In: The Cryptography Mailing List, November 2008. http://www.mail-archive.com/cryptography@metzdowd.com/msg09997.html
60. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (October 2008). bitcoin.org
61. Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S.: Bitcoin and Cryptocurrency Technologies. Princeton University Presss, Princeton (2016)
62. Neiger, G.: Distributed consensus revisited. IPL **49**(4), 195–201 (1994)
63. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: 2014 USENIX Annual Technical Conference (USENIX ATC 2014) (2014)

64. Patra, A.: Error-free multi-valued broadcast and Byzantine agreement with optimal communication complexity. In: Fernàndez Anta, A., Lipari, G., Roy, M. (eds.) OPODIS 2011. LNCS, vol. 7109, pp. 34–49. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25873-2_4
65. Patra, A., Rangan, C.P.: Communication optimal multi-valued asynchronous broadcast protocol. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 162–177. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14712-8_10
66. Patra, A., Rangan, C.P.: Communication optimal multi-valued asynchronous Byzantine agreement with optimal resilience. In: Fehr, S. (ed.) ICITS 2011. LNCS, vol. 6673, pp. 206–226. Springer, Heidelberg (2011). doi:10.1007/978-3-642-20728-0_19
67. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
68. de Prisco, R., Malkhi, D., Reiter, M.: On k-set consensus problems in asynchronous systems. IEEE Trans. Parallel Distrib. Syst. **12**(1), 7–21 (2001)
69. Raynal, M.: Consensus in synchronous systems: a concise guided tour. In: Pacific Rim International Symposium on Dependable Computing (2002)
70. Raynal, M.: Concurrent Programming: Algorithms, Principles, and Foundations. Springer, Heidelberg (2013)
71. Reed, B., Junqueira, F.P.: A simple totally ordered broadcast protocol. In: Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware, LADIS (2008)
72. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. **22**(4), 299–319 (1990)
73. Serafini, M., Bokor, P., Dobre, D., Majuntke, M., Suri, N.: Scrooge: reducing the costs of fast Byzantine replication in presence of unresponsive replicas. In: Dependable Systems and Networks (DSN) (2010)
74. Su, L., Vaidya, N.: Reaching approximate Byzantine consensus with multi-hop communication. In: Pelc, A., Schwarzmann, A.A. (eds.) SSS 2015. LNCS, vol. 9212, pp. 21–35. Springer, Heidelberg (2015). doi:10.1007/978-3-319-21741-3_2
75. Trencseni, M., Gazsó, A., Reinhardt, H.: Paxoslease: diskless Paxos for leases. CoRR abs/1209.4187 (2012)
76. Tseng, L.: Fault-tolerant consensus in directed graphs and convex hull consensus. Ph.D. thesis. University of Illinois at Urbana-Champaign (2016)
77. Tseng, L.: Recent results on fault-tolerant consensus in message-passing networks. CoRR abs/1608.07923 (2016)
78. Tseng, L., Vaidya, N.H.: Asynchronous convex hull consensus in the presence of crash faults. In: Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing, PODC (2014)
79. Tseng, L., Vaidya, N.H.: Fault-tolerant consensus in directed graphs. In: PODC (2015)
80. Turpin, R., Coan, B.A.: Extending binary Byzantine agreement to multivalued Byzantine agreement. IPL **18**(2), 73–76 (1984)
81. Vaidya, N.H.: Iterative Byzantine vector consensus in incomplete graphs. In: Chatterjee, M., Cao, J., Kothapalli, K., Rajsbaum, S. (eds.) ICDCN 2014. LNCS, vol. 8314, pp. 14–28. Springer, Heidelberg (2014). doi:10.1007/978-3-642-45249-9_2
82. Vaidya, N.H., Garg, V.K.: Byzantine vector consensus in complete graphs. In: PODC (2013)
83. Vaidya, N.H., Tseng, L., Liang, G.: Iterative approximate Byzantine consensus in arbitrary directed graphs. In: PODC (2012)

84. Vukolić, M.: The Byzantine empire in the intercloud. SIGACT News **41**(3), 105–111 (2010)
85. Vukolić, M.: The quest for scalable blockchain fabric: proof-of-work vs. BFT replication. In: Camenisch, J., Kesdoğan, D. (eds.) iNetSec 2015. LNCS, vol. 9591, pp. 112–125. Springer, Heidelberg (2016). doi:10.1007/978-3-319-39028-4_9
86. Wang, Y., Kapritsos, M., Ren, Z., Mahajan, P., Kirubanandam, J., Alvisi, L., Dahlin, M.: Robustness in the salus scalable block store. In: NSDI (2013)
87. Wood, T., Singh, R., Venkataramani, A., Shenoy, P., Cecchet, E.: ZZ and the art of practical BFT execution. In: EuroSys (2011)
88. Yin, J., Martin, J.P., Venkataramani, A., Alvisi, L., Dahlin, M.: Separating agreement from execution for Byzantine fault tolerant services. SOSP **37**(5), 253–267 (2003)
89. Zhang, H., Sundaram, S.: Robustness of complex networks with implications for consensus and contagion. In: Proceedings of the 51st IEEE Conference on Decision and Control, CDC (2012)
90. Zhang, J., Chen, W.: Bounded cost algorithms for multivalued consensus using binary consensus instances. Inf. Process. Lett. **109**(17), 1005–1009 (2009)