Atomic Broadcasts and Consensus: A Survey

Philipp Sedlmeier, Johannes Schleger*, Max Helm*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: philipp.sedlmeier@tum.de, schleger@net.in.tum.de, helm@net.in.tum.de

***Abstract*—**

Atomic Broadcast and Consensus constitute important problems in distributed systems and have occupied computer scientists over the last four decades. As of late, they receive a new wave of attention. This paper provides an overview of the theoretical foundations of both topics. Furthermore, it presents recent approaches to the quest for the improvement of such protocols.

***Index Terms*—consensus, atomic broadcast, fault tolerance**

1. Introduction

Today, one can find an abundance of distributed systems that rely on cooperating processes. For these processes, agreeing on data as the basis of their computations is fundamental. For example, the engine control and the flight surface control of an airplane's flight control system need to agree on whether to continue or abort a landing [1].

Malfunctions of a computer system may lead to disastrous outcomes; sticking to the airplane example, to a plane crash. This calls for fault-tolerant computers and computer networks. Fault-tolerance can, for example, be ensured by the use of process replication, in particular a synchronously replicated storage. This approach relies heavily on atomic broadcasts [2].

In this paper, we will provide a short comparison of recent concepts in these fields. In Section 2, we will investigate the topic's key terms and their relationship. We will present some related work in Section 3. Then, we will examine recent approaches in protocol development in Section 4. We will follow this with a discussion about if and how we can compare these protocols in Section 5.

2. Background

There exists an ample amount of literature about atomic broadcast and consensus [3]. However, there is a large divergence in the underlying definitions. Thus, we will provide the definitions of some essential terms in Section 2.1. Subsequently, we will give a short overview of theoretical results regarding the solvability of such problems in Section 2.2. Eventually, we will examine how the different concepts can be reduced to each other in Section 2.3.

2.1. Terms

Atomic Broadcast. The terms *Atomic Broadcast* and *Total Order Broadcast* are used interchangeably [3]. Informally, it requires all correct processes to "deliver the same messages in the same order" [4]. Formally, there are two major problem definitions. Following Cristian et al., an atomic broadcast protocol must fulfill the three properties *Atomicity*, *Order* and *Termination* [5]. However, all three properties depend on physical times, namely the points in time when each processor delivers an update.

Nowadays, definitions omitting such references are preferred [2]. Défago et al. emphasize the total order and identify the following four properties [3]:

- **Validity**: if a correct process broadcasts a message m, then it eventually delivers m.
- **Uniform Agreement**: if a process delivers m, all correct processes eventually deliver m.
- **Uniform Integrity**: for any message m, every process delivers m at most once, and only if m was previously broadcast by its sender.
- **Uniform Total Order**: if two processes p and q deliver messages m and n, then p delivers m before n if, and only if, q delivers m before n.

Nevertheless, they also point out that the agreement and total order properties are not necessarily specified uniformly and may also just apply to correct processes.

Consensus. The problem of *Consensus* is a problem of *agreement* between multiple processes on some value that one or more of those processes have proposed. The requirements for a consensus algorithm are summarized by Coulouris et al. [6]:

- **Agreement**: the decision value of all correct processes is the same.
- **Integrity**: if all correct processes proposed the same value, any correct process that has decided has chosen that value.
- **Termination**: Eventually, each correct process decides.

Integrity is also known as *validity* and sometimes defined differently; for instance, that the decision value initially must have been proposed by one of the processes [6].

Lamport et al. propose two variants of the consensus problem. In **interactive consistency**, each process computes a vector of values with an element for each process [7]. They require every correct process to compute the same vector, and every vector element corresponding to a correct process is that process's private value. The **Byzantine generals** problem involves one distinguished process that supplies a value the others are to agree

doi: 10.2313/NET-2020-11-1_19

upon [6], as is informally, but vividly illustrated by a commanding army general that sends messages to his lieutenants [8].

Permission. In the classical approach to consensus protocols, the communicating participants are already known beforehand. Such protocols are known as *permissioned*, in contrast to *permissionless* protocols, where participants can join or leave freely and neither their exact number nor their identity is known [9].

Failures. In the previous definitions, we already introduced the notion of a *correct* process. A correct process is any process that does not sustain process failures. These can be divided into four classes [3]:

- **Crash failure**: a process stops performing any activity.
- **Omission failure**: a process omits performing some actions, e.g. sending a message.
- **Timing failure**: a process violates timing assumptions of the system model[1].
- **Byzantine failure**: a process displays arbitrary or even malicious behavior.

These failure classes are nested in the above order, i.e. $CF \subset OF \subset TF \subset BF$ [5]. Indeed, most of the literature focuses only on crash and Byzantine failures.

2.2. Solvability

Consensus and atomic broadcast are easily solvable if the participating processes cannot fail [6]. Otherwise, the possibility of reaching consensus between processes is in question. We will shortly summarize the most influential results on this issue.

Fischer et al. showed that in an asynchronous system, every consensus algorithm has the possibility of nontermination, given the crash failure of only a single process [10][2]. Dolev et al. identified synchrony conditions and examined how they affect the number of faults that can be tolerated [11]. Dwork et al. introduced the concept of partial synchrony and determine the solvability of consensus for multiple partially synchronous models [12]. To bypass the problem altogether, Chandra and Toueg introduced unreliable failure detectors that can identify faulty processes and can be used to solve consensus, as long as the faulty processes are in the minority [4].

For any algorithm exist lower bounds on some properties. For example, if at most f processes sustain a crash failure, every algorithm reaching an agreement requires at least $f + 1$ rounds of information exchange [11]. In a system with n processes, of which f are faulty, consensus is solvable if and only if $n \geq 3f + 1$ (with Byzantine faults) and if and only if $n \geq 2f + 1$ (with non-Byzantine faults) [1], [13].

For a vivid description of the Byzantine case, we refer the avid reader to [8]. The basic intuition is that as long as $n \leq 3f$ holds, a node may be able to detect faulty behavior, but is not able to distinguish which node caused the faulty behavior and which node is a "victim" as well.

1. Naturally, timing failures can only occur in synchronous systems, as a system is considered to be *asynchronous* if we make no timing assumptions at all [4].

2. Due to the authors' initials, this result is informally known as *FLP impossibility*. Note, that it does not mean reaching consensus in such a system is not possible at all, there is just no deterministic solution.

2.3. Problem Reduction

It has been shown that the previous concepts can be reduced to each other in many cases. We can see in [6] that a solution for any of the three variants of the consensus problem from Section 2.1, i.e. *Consensus*, *Interactive Consistency* and *Byzantine Generals*, can easily be used to construct a solution to one of the two other variants.

Chandra and Toueg proved consensus and atomic broadcast to be equivalent problems in asynchronous systems with crash failures. They also claim equivalence under arbitrary, i.e. Byzantine failures, but omit any proof [4]. Indeed, the relation between the two problems seems to be more complicated.

The first comprehensive study on this topic seems to be by Milosevic et al. They show that the equivalence of consensus and atomic broadcast does **not** hold in general, but the definition of *validity* determines whether they are equivalent, or one is harder than the other [14].

3. Related Work

Many theoretical results on consensus and related issues have been summarized by Fischer, for example in [1], [15]. The emergence of cryptocurrencies in the last decade, most notably Bitcoin, has attracted interest and progress in the development of the underlying consensus protocols. A survey of blockchain consensus protocols can, for example, be found by Xiao et al., who identify their core concepts and conduct a performance analysis [9].

Due to the scope of the topic, surveys on atomic broadcast protocols mostly concentrate on particular aspects. For example, Cason et al. characterize the latency variability of different atomic broadcast protocols [16]. The most extensive study so far seems to be one by Défago et al. that surveys - and classifies - around 60 algorithms [3]. Their classification is accompanied by a performance analysis concerning the message ordering strategies [17]. However, we did not find more recent studies that are nearly as extensive.

4. Available Protocols

We observe that many recently proposed protocols aim to improve particular aspects of already existing protocols. Consequently, we will investigate the issues that standard protocols suffer from, and present solutions that have been proposed recently.

In this survey, we will only cover permissioned protocols. Furthermore, all of these protocols assume a partially synchronous system [12]. We will examine crash-fault tolerant protocols (CFT) in Section 4.1 and those that tolerate Byzantine faults (BFT) in Section 4.2. We will present a new concept of fault tolerance in Section 4.3.

4.1. Crash-Fault Tolerant Protocols

Paxos. The basis for most implementations of state machine replication until today is **Paxos**, first published in 1998 by Lamport [18]. Despite its prevalence, there are some substantial drawbacks to Paxos. To reach an agreement, more than half of the processes need to be running and communicating synchronously [18]. It is prone to failures that are common in some systems [19]. Furthermore, it depends on one process that acts as a leader. While Paxos is able to recover from a crash of this primary, this process is quite slow [19].

Raft. Another frequent critique of Paxos is that it is notoriously difficult to understand and not a good basis for practical systems. Therefore, Ongaro and Ousterhout developed **Raft** as an alternative, making use of problem decomposition and state space reduction [20]. Its new features include the concept of a strong leader that is authoritative for the distributed log entries. The leader election is performed through a heartbeat mechanism. That means, the current leader periodically sends heartbeat messages to its followers; if a follower does not receive such messages over a period of time, it begins an election for a new leader. To resolve or even prevent split votes in leader elections, Raft uses randomized timeouts. In contrast to Paxos, the leader election mechanism is a part of the consensus protocol itself [20].

However, while Raft's safety does not depend on timing assumptions, its availability does[3]. In particular, *broadcast time* $<$ *leader-election timeout* $<$ *mean time between failures* must hold. Indeed, Howard verifies Raft's efficiency in a well-understood network environment, where the parameters can be set accordingly [21]. She also states this is not the case in an internet-scale environment yet, but proposes modifications to that effect.

4.2. Byzantine-Fault Tolerant Protocols

Protocols that can tolerate Byzantine failures as well are widely considered to be badly scalable. Not only do they require more nodes than their CFT counterparts[4], node failures are also often assumed to be independent [22]. Yet, in order to achieve this, each node has to run with different operating systems and so forth. BFT protocols usually have higher time and message complexities as well. Nevertheless, research and development in such protocols have surged with the interest in permissioned blockchains [23].

Practical Byzantine Fault Tolerance (**PBFT**) by Castro and Liskov is the first practical implementation of a BFT consensus protocol [22]. The literature disagrees on whether PBFT is just inspired by Paxos or is its Byzantine version [9], [23]. Anyway, it is still "regarded as the 'baseline' for practical BFT implementations" [24]. However, Amir et al. showed the vulnerability of PBFT - and other leader-based protocols - to performance attacks by a small number of Byzantine nodes that can seriously impair the system's performance [25].

3. Otherwise, it would contradict the FLP impossibility result [10].
4. Namely at least $3f + 1$, as we have seen in Section 2.2

Scalability. Thai et al. proposed **HiBFT**, an extension of PBFT, that is supposed to be scalable up to hundreds of nodes [26]. Under the assumption that the system is organized in a hierarchical structure, multiple nodes are composed into logical groups. Then, not all nodes communicate with each other, but only the group leaders. Hence, HiBFT reduces the number of computationally expensive signature verifications and message complexity. Indeed, first results showed better performance in throughput and latency of HiBFT compared to PBFT, despite a sextupled number of nodes [26].

Performance. Zyzzyva, proposed by Kotla et al. [27], aims at reducing the replication overhead by omitting one communication phase by optimistic speculation. It executes client requests immediately, without running an agreement protocol first. This boosts the protocol's performance and reduces its message complexity - compared to PBFT - in gracious executions. Indeed, it is considered the "state of the art" concerning performance. However, Zyzzyva relies on the clients to resolve the cases where something went wrong; a client needs to detect whether it has received the same reply from all replicas [27]. Furthermore, safety violations of the protocol have recently been discovered [28].

Simplified View Change. Yin et al. presented **HotStuff** [28], which aims to achieve *optimistic responsiveness*, i.e. the designated leader needs to wait only for $n-f$ responses after global stabilization time (GST) [12] is reached. Hence, the protocol is designed to reach consensus fast, i.e. at the pace of the actual network delay. More importantly, it reduces the message complexity during view changes to linearity in the number of nodes even in the presence of leader failures. To achieve this, HotStuff adds a phase to the view change process and merges it into the regular protocol. This leads to a slightly higher latency, but also a higher throughput compared to other BFT protocols [28]. The modified version *Chained HotStuff* is essentially a pipelined version, where a quorum certificate can serve in different phases simultaneously. This version serves also as the basis of the LibraBFT consensus protocol, whose authors cite HotStuff's reduced communication costs as a main reason for using it [29].

Robustness. Clement et al. discovered that while "recently developed BFT state machine replication protocols are quite fast, they don't tolerate Byzantine faults very well" [30]. They state, that even single server or client failures can render such systems practically useless. Thus, they proposed **Aardvark**, a protocol specifically designed to be robust, i.e. that provides acceptable and predictable performance under all circumstances, including the occurrence of failures. To contain the effect of a Byzantine primary process, the system monitors its performance and changes the view if it is performing slowly. They found that Aardvark's performance is within 40% of the best performance of other state-of-the-art protocols on the same hardware during gracious executions, while it outperforms the same protocols by far under various attacks [30].

Leaderless BFT. Instead of improving BFT protocols by improving the leader-change mechanisms, another recent

doi: 10.2313/NET-2020-11-1_19

approach is the development of completely leaderless protocols. **AllConcur**, proposed by Poke et al., is a leaderless atomic broadcast protocol where the nodes "exchange messages concurrently through an overlay network, described by a digraph G" [31]. The algorithm makes use of a failure detector and an early termination mechanism. However, the algorithm's liveliness property only holds if the number of failures is bounded by the vertex connectivity of G and if the failure detector is complete and accurate. The authors claim that AllConcur's throughput beats that of a Paxos implementation by orders of magnitude; unfortunately, their comparison is not comprehensive and seems to cover only one specific case [31].

Crain et al. proposed **DBFT** that replaces a leader by a *weak coordinator* that does not impose its value on its fellow processes [32]. Thus, non-faulty processes can decide on a value without its help and a faulty coordinator cannot prevent the other processes from reaching consensus. Unfortunately, the authors do not provide a performance evaluation of the protocol, apart from reporting a quadratic message complexity. They claim, however, that DBFT is used by one of the fastest blockchains to date [33].

4.3. Cross-Fault Tolerant Protocols

Liu et al. argue that the overhead for multiple properties inherent to BFT is mostly due to the "assumption of a powerful adversary that can fully control not only the Byzantine faulty machines, but at the same time also the message delivery schedule across the entire network" [34]. However, they claim that this scenario is rather unrealistic in most cases. Hence, they propose the concept of **Cross-Fault tolerance**, **XFT** for short. XFT requires the same number of replicas as CFT, i.e. $2f + 1$, and provides all its reliability guarantees. Thus, it is strictly stronger than CFT. In addition, it provides safety and liveliness when Byzantine faults occur, as long as a majority of the replicas are correct and can communicate with each other synchronously.

In the same paper, they provide a XFT-SMR protocol, *XPaxos*. Its performance outperforms PBFT and Zyzzyva, while coming close to the performance of an optimized Paxos [34]. However, it doesn't scale with the number of faults and suffers from the same performance shortcomings in the case of failures as other leader-based protocols. Thus, Garg et al. recently proposed the multi-leader XFT consensus protocol **Elpis**. It introduces a concept of per-object ownership, where ownership of accessible objects is assigned to the nodes. Then, not a single leader is responsible for ordering all commands, but each node is responsible for ordering the commands that concern the objects it has been assigned to as owner. This ownership can also be changed dynamically. The authors claim that Elpis achieves a performance twice as high as XPaxos [35].

5. Comparing Protocols

As we have seen in Section 4, many protocols aim to improve or remedy particular shortcomings of already existing protocols. Even though all of these protocols solve the same problem - i.e. the problem of consensus or atomic broadcast - the underlying application determines which protocols can be used. For example, HiBFT can be used for a permissioned blockchain only if its replicas are - or can be - ordered in a hierarchical structure.

Singh et al. argue that comparing different BFT protocols and choosing an appropriate one is difficult because they are evaluated under different and benign conditions [24]. Therefore, they propose a simulation environment that provides identical and realistic conditions. Their comparison yields insight into which characteristics of a system environment favor or disadvantage a specific protocol. Nevertheless, they also conclude that there is no one-size-fits-all protocol and that such a protocol may be impossible to build at all [24]. A similar conclusion was drawn earlier by Défago et al. in their performance analysis of atomic broadcast protocols [17].

The discussion whether there is a protocol superior to the others is not restricted to BFT consensus alone. While Paxos and Raft are the two dominating algorithms for distributed consensus, it is an open discussion which is the better one, as Howard and Mortier point out [36].

Unfortunately, there are still very few comparisons of consensus or atomic broadcast protocols available. Those that are available focus on blockchain consensus and do not make use of a simulation environment [9], [37]. The finding from [24] that there is no single superior protocol available seems to have gained significantly more popularity. Among others, it inspired the development of **Hyperledger Fabric**, a blockchain platform with a modular consensus component, so that the implementation of consensus can be adjusted to the present use case [38].

We should also note that some problems are inherent to consensus protocols in general, regardless of their specific application. For example, a high message complexity is a burden on every protocol that solves BFT consensus. Thus, developments that reduce the cost of communication in distributed systems may benefit all consensus protocols similarly. To that end, Goren and Moses recently examined silent information transfer in the presence of failures [39].

6. Conclusion

As we have seen, there exists a vast and still growing number of consensus and atomic broadcast protocols. These protocols vary not only in design and performance but also in the underlying assumptions. Which protocol to use depends on the specific setting and is still an open discussion. Thus, modular and extensible projects may become even more relevant in the future.

References

[1] M. J. Fischer, "The Consensus Problem in Unreliable Distributed Systems (A Brief Survey)," in *Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory*, M. Karpinski, Ed. Springer, 1983.

[2] X. Défago, "Atomic Broadcast," in *Encyclopedia of Algorithms*, M.-Y. Kao, Ed. Boston: Springer, 2008.

[3] X. Défago, A. Schiper, and P. Urbán, "Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey," *ACM Computing Surveys*, vol. 36, no. 4, Dec. 2004.

[4] T. D. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *J. ACM*, vol. 43, no. 2, Mar. 1996.

doi: 10.2313/NET-2020-11-1_19

[5] F. Cristian, H. Aghili, R. Strong, and D. Dolev, "Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement," *Information and Computation*, vol. 118, no. 1, 1995.

[6] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed. Boston: Addison-Wesley, 2012, ch. Consensus and related problems.

[7] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *J. ACM*, vol. 27, no. 2, Apr. 1980.

[8] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, Jul. 1982.

[9] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A Survey of Distributed Consensus Protocols for Blockchain Networks," *ArXiv*, vol. abs/1904.04098, 2019.

[10] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *J. ACM*, vol. 32, no. 2, Apr. 1985.

[11] D. Dolev, C. Dwork, and L. Stockmeyer, "On the Minimal Synchronism Needed for Distributed Consensus," *J. ACM*, vol. 34, no. 1, Jan. 1987.

[12] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *J. ACM*, vol. 35, no. 2, Apr. 1988.

[13] L. Lamport, "Lower bounds for asynchronous consensus," *Distributed Computing*, vol. 19, no. 2, Oct. 2006.

[14] Z. Milosevic, M. Hutle, and A. Schiper, "On the Reduction of Atomic Broadcast to Consensus with Byzantine Faults," in *2011 IEEE 30th International Symposium on Reliable Distributed Systems*, 2011.

[15] M. J. Fischer, "A Theoretician's View of Fault Tolerant Distributed Computing," in *Fault-Tolerant Distributed Computing*, B. Simons and A. Spector, Eds. New York: Springer, 1990.

[16] D. Cason, P. J. Marandi, L. E. Buzato, and F. Pedone, "Chasing the Tail of Atomic Broadcast Protocols," in *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*, 2015.

[17] X. Défago, A. Schiper, and P. Urbán, "Comparative Performance Analysis of Ordering Strategies in Atomic Broadcast Algorithms," *IEICE Trans. on Information and Systems*, vol. E86-D, no. 12, Dec. 2003.

[18] L. Lamport, "The Part-Time Parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, May 1998.

[19] H. Howard, "Distributed consensus revised," University of Cambridge, Tech. Rep. UCAM-CL-TR-935, Apr. 2019.

[20] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia: USENIX Association, Jun. 2014.

[21] H. Howard, "ARC: Analysis of Raft Consensus," University of Cambridge, Tech. Rep. UCAM-CL-TR-857, Jul. 2014.

[22] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. USENIX Association, 1999.

[23] C. Cachin and M. Vukolić, "Blockchain Consensus Protocols in the Wild," in *31st International Symposium on Distributed Computing (DISC 2017)*, A. W. Richa, Ed., vol. 91. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[24] A. Singh, T. Das, P. Maniatis, P. Druschel, and T. Roscoe, "BFT Protocols under Fire," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2008.

[25] Y. Amir, B. Coan, J. Kirsch, and J. Lane, "Prime: Byzantine Replication under Attack," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 4, 2011.

[26] Q. Thai, J.-C. Yim, T.-W. Yoo, H.-K. Yoo, J.-Y. Kwak, and S.-M. Kim, "Hierarchical Byzantine fault-tolerance protocol for permissioned blockchain systems," *The Journal of Supercomputing*, vol. 75, Jun. 2019.

[27] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative Byzantine Fault Tolerance," *ACM Transactions on Computer Systems*, vol. 27, no. 4, Jan. 2010.

[28] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT Consensus in the Lens of Blockchain." *arXiv: Distributed, Parallel, and Cluster Computing*, 2018.

[29] LibraBFT Team, "State Machine Replication in the Libra Blockchain," May 2020.

[30] 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*. USENIX Association, 2009.

[31] M. Poke, T. Hoefler, and C. W. Glass, "AllConcur: Leaderless Concurrent Atomic Broadcast," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2017.

[32] T. Crain, V. Gramoli, M. Larrea, and M. Raynal, "DBFT: Efficient Leaderless Byzantine Consensus and its Application to Blockchains," *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, 2018.

[33] T. Crain, C. Natoli, and V. Gramoli, "Evaluating the Red Belly Blockchain," *ArXiv*, vol. abs/1812.11747, 2018.

[34] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolić, "XFT: Practical Fault Tolerance beyond Crashes," *ArXiv*, vol. abs/1502.05831, 2016.

[35] M. Garg, S. Peluso, B. Arun, and B. Ravindran, "Generalized Consensus for Practical Fault Tolerance," in *Proceedings of the 20th International Middleware Conference*. ACM, 2019.

[36] H. Howard and R. Mortier, "Paxos vs Raft: Have We Reached Consensus on Distributed Consensus?" in *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*. New York: Association for Computing Machinery, 2020.

[37] S. Wan, M. Li, G. Liu, and C. Wang, "Recent advances in consensus protocols for blockchain: a survey," *Wireless Networks*, Nov. 2019.

[38] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *Proceedings of the Thirteenth EuroSys Conference*. New York: Association for Computing Machinery, 2018.

[39] G. Goren and Y. Moses, "Silence," *J. ACM*, vol. 67, no. 1, Jan. 2020.