BOSTON COLLEGE

HONORS THESIS

Byzantine Concensus: Theory and Applications in a Dynamic System

---

Author:
Yifan Zhang

Supervisor:
Lewis Tseng

May 19, 2021

BOSTON COLLEGE

Abstract

Department of Computer Science

Byzantine Concensus: Theory and Applications in a Dynamic System

by Yifan Zhang

This survey paper aims to study the theory and applications of Byzantine general problems. In particular, we compare models and methods being used to study Byzantine broadcast problems in dynamic systems, where nodes may join and leave at any time. Byzantine consensus is one of the most fundamental problems in distributed algorithms. There exist two main variants of Byzantine consensus problems: Byzantine Broadcast (BB) and Byzantine Agreement (BA). Byzantine broadcast problems further have several variants including Byzantine reliable broadcast (BRB) and Byzantine consistent broadcast (BCB). Byzantine consistent broadcast has relaxed conditions compared to the original Byzantine broadcast protocol, and has significant applications in blockchain systems. This paper describes the history of research on Byzantine consensus problems and their applications in a dynamic system, and helps readers understand the common techniques being used in these works.

Acknowledgements

Special thanks to professor Lewis Tseng for inspiring me in choosing this particular topic, believing in me, leading me to through my thesis and answering my questions.

Contents

Chapter 1

Introduction

Consensus is the task of getting all nodes in a group to agree on some specific value based on the votes of each processes. All nodes must agree upon the same value and it must be a value that was submitted by at least one of the nodes. A fundamental problem in distributed computing and multi-agent systems is to achieve overall system stability, where all nodes agree on some value needed for computation, in the presence of a number of faulty processes. This often requires coordinating processes to reach consensus. Byzantine consensus is the problem for n nodes to agree on a value, despite the fact that up to f of them may behave arbitrarily, which is called Byzantine failures. Since Byzantine failures imply no restrictions they can confuse the failure detection systems, which makes fault tolerance difficult. Prior works have done extensive study on Byzantine consensus problems under various models, such as the asynchronous system [22][7][24] and the synchronous [5] system. Besides faulty nodes model, there are studies on reaching consensus under faulty links model [22][25] as well.

There exist a few variants formulations for the Byzantine consensus problem, including Byzantine Broadcast(BB) and Byzantine Agreement(BA) problems. In Byzantine agreement, each party has an input value, and all parties try to decide on the same value. Lamport et al. [12] showed that more than 2/3 of the participating parties must be honest to reach consensus. Since then, Byzantine agreement has been studied under both synchronous and asynchronous settings and in deterministic [9] and randomized [2][20] models.

In Byzantine broadcast, there is a designated sender that tries to broadcast a value to the parties, and the goal is for all parties to learn the value and agree on it. Lamport et al. [12] first introduced the Byzantine broadcast problems and presented protocols and fault tolerance bounds for two settings (both synchronous). Without cryptographic assumptions, Byzantine broadcast and agreement can be solved if $f < n/3$. With assumptions on digital signatures, Byzantine broadcast can be solved if $f < n$ and Byzantine agreement can be solved if $f < n/2$.

The Byzantine broadcast problem further has several variants. In particular, a fundamental primitive for synchronization among a group of parties is reliable broadcast, where a distinguished party broadcasts a value m to the other parties. Two basic types of systems are considered: *synchronous* systems and *asynchronous* systems. Protocols in synchronous systems can be seen as a sequence of rounds. In each round, every node sends messages to other nodes, receives messages sent to it in that round, and then updates its state based on these messages. By contrast, there are no bounds on message delays. Protocols in asynchronous systems can also be viewed as a sequence of rounds. In each round, a node sends messages to all other nodes, waits for only $n-f$ messages of that round, and updates state. Besides, there are two failure types to consider: *Fail-Stop* and *Byzantine*. In Fail-Stop failure, faulty nodes may omit messages at any time, but they stop participating in the protocol after the first time they omit messages. In Byzantine failure, faulty nodes can fail to send messages when they should and can send contradictory messages.

There are two types of protocols to consider for reliable broadcast: deterministic and randomized. In deterministic protocols, no random steps are taken. For deterministic protocols, there is a *termination* requirement that all correct nodes decide by some round r. In randomized protocols, if the sender is honest, then all non-faulty parties should accept m; and if the sender is faulty, all non-faulty parties should decide for the same value or not terminate the protocol at all. There is a considerable literature [3][21] devoted to the implementation of reliable broadcast in presence Byzantine failures in asynchronous systems.

Byzantine Consistent Broadcast (BCB) is probably one of the easiest problems in the Byzantine consensus family, as it only requires honest parties not to decide while others do not have to decide. We are interested in studying Byzantine Consistent Broadcast that requires the following two properties: *consistency* and *validity*. The importance of BCB may have been somewhat overlooked. The most important application for BCB today is Byzantine fault tolerant (BFT) replication systems (also known as blockchains). Among various applications of BCB, we focus on BCB protocol in a dynamic system where nodes can join or leave at anytime.

Byzantine consensus problems in dynamic systems are important because they have more applications in real life because the system is not always static, and nodes in the system may change with respect to time. Hence, we compare the models and methods of previous works on dynamic systems, as well as important models in Bitcoin/Blockchain.

Chapter 2

Preliminaries

We look at some commonly used techniques to study Byantine consensus problems, especially BCB, in dynamic system. Studies in broadcast problems generally assume digital signatures and public-key infrastructure (PKI), and use $\langle x \rangle_r$ to denote a message x signed by party r. We abstract away some details of cryptography but it is important to keep in mind that any PKI system must have some method by which certificate authorities can authenticate users, and that all participants in the PKI system trust that method. Therefore, this prevents faulty nodes from authenticating the messages. Besides, it is assumed that the signature schemes enjoy ideal unforgeability, which means any node r is able to sign specific messages he chose himself or messages provided by an opponent. Unforgeability promises that any node r is able to sign the messages before sending them. It is further assumed that a threshold signature scheme, in which a set of signatures $\langle x \rangle_r$ for a message x from t distinct parties can be combined into a threshold signature for x with the same length as an individual signature. This is because signature from fewer than t parties have no useful information and at least t parties are required for creating a signature. If we set t to be $n - f$ where the number of nodes $n > 3f$ and there are $\leq f$ faulty nodes, a set of signatures with useful information cannot be created by faulty nodes.

In most studies on broadcast problems in a dynamic system, it is assumed that every node knows who's in the system and the protocol proceeds in synchronous rounds. Under synchronous setting, if an honest party sends a message at the beginning of some round, an honest recipient receives the message at the end of that round. To be more specific, a round consists of three phases (i) nodes send messages in the current round, (ii) nodes receives messages sent at the beginning of the current round and (iii) process the received messages and update local state. There are a set of n_t nodes in our model, and n_t is the number of nodes in the system at time t. As we have mentioned earlier, nodes can join and leave the system anytime in a dynamic system. A node that is not faulty throughout the execution is said to be honest and faithfully executed in the protocol. It is only assumed that each node knows who's in the system but it does not know who is faulty. We use the term *quorum* to mean the minimum number of all honest parties in round t, i.e., $n_t - f$. Quorum mechanism is a voting algorithm commonly used in distributed systems to ensure data redundancy and final consistency.

Since synchronous setting is highly unrealistic in reality, we consider Byzantine consensus problems in asynchronous systems where there are no bounds on message delays. The FLP impossibility result [17] proved that it is impossible to achieve agreement, validity, and termination simultaneously in asynchronous systems. As a result, previous works sacrificed one or two of the properties by: (1) either having a randomized guarantee to achieve relaxed versions of consensus; (2) or achieving "weak synchronous" by requiring a stabilizing period in which message delays are moderate. Thus, even though the number of rounds to reach agreement is not bounded, the probability that the protocol does not terminate is zero.

Byzantine general problems under both synchronous and asynchronous settings often adopt protocols with multiple phases. A common technique is to have one phase where nodes echo the sender's proposed value so that they can detect if the sender "equivocates", i.e., proposing different values to different parties. In most dynamic systems, the join/leave protocols of nodes usually require multiple phases of message-passing which generally consist of: new nodes multicasting join/leave request, existing nodes check signature and identities, and existing nodes echoing the new node, etc.

Nevertheless, the number and function of phases can vary depending on the system model. For example, Byzantine broadcast protocols have a designated sender and focus on the message exchange from sender and the rest of nodes in the system, while others Byzantine general problems do not have a sender but instead assign a primary node in the system. Hence, we want to compare different protocols in the next few chapters.

Chapter 3

Byzantine broadcast problems

As mentioned earlier, a designated sender denoted by r_s has an input v_{in} to broadcast to all parties in broadcast. The broadcast problem further has several variants. The consistency and validity conditions of BA, BB and BRB are the same as the conditions for BCB protocol, and the only difference is termination condition. The Byzantine reliable broadcast [4] has a totality condition that is more relaxed than the termination condition for Byzantine Broadcast. As shown in the definitions, BCB has more relaxed requirements compared to the original byzantine broadcast protocol and Byzantine reliable broadcast because it allows some parties to decide while others do not.

Definition 1. (Byzantine Consistent Broadcast (BCB)) A Byzantine consistent broadcast protocol must satisfy (i) consistency: if two honest parties r and r' decide values v and v', then $v = v'$, and (ii) validity: if the sender r_s is honest, then all honest parties decide the input value v_{in} and terminate.

Definition 2. (Byzantine Braodcast(BB)). A *Byzantine broadcast* protocol must satisfy (i) consistency: if two honest parties r and r' decide values v and v', then $v = v'$, (ii) validity: if the sender r_s is honest, then all honest parties decide the input value v_{in} and terminate, and (iii) termination: every honest party decides a value and terminates.

Definition 3. (Byzantine Reliable Broadcast(BRB)). A *Byzantine reliable* broadcast protocol must satisfy (i) consistency: same as above, (ii) validity: same as above, and (iii) totality: if an honest party decides a value, then every honest party decides a value.

Now we compare different methods from studies on Byzantine broadcast problems. As mentioned earlier, there are two main variants of Byzantine consensus: broadcast and agreement. The Byzantine broadcast further has several variants Byzantine reliable broadcast and Byzantine consistent broadcast. Bracha [4] developed a technique that reduces the effect of Byzantine nodes on the system. To be more specific, this technique has two parts: a reliable broadcast primitive and a validation method. The broadcast primitive forces the same messages to all correct processes while the validation method forces the faulty processes to send only messages that could have been sent by correct processes. There are three types of messages in the protocol: $(initial, v)$, $(echo, v)$ and $(ready, v)$ where processes propose, echo, and accept messages respectively.

Many of the early studies on broadcast problems are too idealistic and their solutions are too ineffective to implement in reality. Thus, Castro and Liskov [16] introduced PBFT (Practical Byzantine Fault Tolerance) as the first practical and efficient solution in a weakly synchronous environment, such as the Internet. PBFT solves a more general problem called state machine replication so the setting is different from settings in BB/BA. A state machine stores a state of the system. It receives a set of inputs (commands) and the state machine applies these inputs in a sequential order using a transition function to generate an output and updated state. Now we consider a client-server setting. The server replicas all start with the same state. When they receive concurrent requests from a client, non-faulty replicas must first agree on the sequence of client commands that they receive. After the sequence is agreed upon, the replicas apply commands one by one. Assuming the transition function is deterministic, all honest server replicas maintain an identical state at all times. The requirements for a Fault-tolerant SMR are similar to those for BB and BA. But a major different is that in BB and BA, the nodes executing the protocol are the ones learning the result. In State Machine Replication

(SMR), the replicas engage in the consensus protocol but need to convince clients of the results. For example, if there are f Byzantine replicas, the client needs to communicate with at least $f+1$ replicas to know that it has communicated with at least one honest replica. While Bratcha's reliable broadcast protocol does not use signature schemes, all replicas in SMR know the others' public keys to verify signatures.

There will be a primary node responsible for ordering the requests from the clients. However, if the primary node does not work, *view changes* will be carried out to elect a new primary node. Similar to Bracha's reliable broadcast, this model also adopts a three-phase protocol: *pre-prepare, prepare* and *commit.* The primary node multicasts the sequence of proposed requests from the client to other nodes, like what the sender does in broadcast protocol. Upon receiving a *pre-prepare* message, nodes enter the *prepare* phase to echo the message. Like Bracha's reliable broadcast model, Castro and Liskov's model used a quorum mechanism (which is explained in the previous chapter). Once collecting a quorum of $2f$ *prepare* messages, nodes will *commit* that message. Then replicas execute the message and send a reply to the client. Lastly, the client waits for $f+1$ replies from different replicas with same result and valid signatures. This is the result of the operation.

Similarly, Momose and Ren [18] desgined an algorithm in BCB protocol where everyone interacts with the designated sender only. The protocol proceeds in synchronous rounds. There are four phases in the algorithm: *Propose, Echo, Vote* and *Forward.* Although BCB protocol has more relaxed conditions compared to Byzantine broadcast, BCB has important applications as well. Momose and Ren studied how BCB protocol can be used to close the considerable gaps in the communication complexity of Byzantine consensus. The linear communication complexity when $f < n/2$ can be achieved using an expander graph. An expander graph has a constant number of edges per vertex, which ensures constant communication per party and good connectivity to detect inconsistent values effectively.

They further discovered a quadratic communication lower bound when $f > n/2$ to solve BCB under fault majority and Byzantine broadcast and Byzantine agreement protocols with quadratic communication.

The simplest way to obtain broadcast in a multiple hop network is by using flooding, where the sender sends the message to everyone in its transmission range. Then each device that receives a message for the first time delivers it to the application and also forwards it to all other devices in its range. Although flooding is robust, it is wasteful as too many messages are being sent. Therefore, many broadcast protocols maintain an overlay, which typically covers all nodes in the system while each node has a limited number of neighbors. Given an overlay, broadcast messages are flooded only along the paths of the overlay, and thus reducing the number of messages sent and the number of collisions.

However, having an overlay reduces the robustness of broadcast protocol against failures, especially Byzantine behavior of overlay nodes. To solve this problem, Drabkin et al. [26] present an overlay based Byzantine tolerant broadcast protocol that overcomes Byzantine failures by combining digital signatures, gossiping of message signatures, and failure detectors. The failure detectors collect reports of bad signatures and other observable deviations from the protocol. The information obtained from failure detectors is used to ensure that there are enough correct nodes in the overlay so that the correct nodes of the overlay form a connected graph and that each correct node is within the transmission disk of an overlay node that does not exhibit Byzantine behavior. So we can reduce $f+1$ node independent overlays to one single overlay. At the same time when messages are disseminated, signatures about these messages are being gossiped by all nodes in the system in an unstructured manner. This allows all nodes to learn about the existence of a message even if some of the overlay nodes fail to forward them. Moreover, multiple gossip messages are aggregated into one packet since gossips are sent periodically. The benefit of gossiping is that if a node hears a gossip about a message that it has never received, it can explicitly ask the message both from its overlay neighbor and from the node from which it received the gossip. This greatly reduces the number of messages generated by the protocol since nodes do not need to be sent $f + 1$ times like they do in previous protocols.

In reality, the implementation and specification of Byzantine broadcast protocols can vary based on the environments. In particular, asynchronous reliable broadcast is widely used as building blocks for other distributed computations in reality, such as secure distributed storage. A client starts the *dispersal* protocol as it decides to store a file in a distributed storage system provided by n servers. The file is split into n different blocks, each one being stored by one of the n servers. Cachin and Tessaro [6] adapted asynchronous reliable broadcast to implement a simple asynchronous verifiable information dispersal scheme. A key concept is the *gateway*, which is an non-faulty party through which clients access the servers comprising the storage system. The idea is to replace the gateway by an asynchronous reliable broadcast protocol such as [3] so that the scheme is robust against corrupted clients, and then the server can keep its own block in memory together with the list of hashes.

Chapter 4

Dynamic Systems

Byzantine consensus problems have been extensively researched in dynamic networks. However, many previous works assume that all nodes are *fault-free*, but the network is controlled by a message adversary. Kuhn et al. [8] showed that eventual consensus is hard in the absence of a good initial upper bound on the size of the network. Fugger et al. [15] proved tight lower bounds on the contraction rates of asymptotic consensus algorithms in dynamic networks. More realistic settings consider node failures. Augustine et al. [10] studied Byzantine agreement in dynamic networks and proposed randomized distributed algorithms that achieve almost-everywhere Byzantine agreement. They assumed that the total number of nodes in the network remains constant while both nodes and edges in the expander graph can change arbitrarily.

Instead of changing nodes and edges in an expander graph, Tseng [14] studied eventual consensus and presented a simple algorithm in both static and dynamic systems where nodes leave and join the system. The communication channel is assumed to be *fair-loss*. The message will be eventually delivered only if both ends are fault-free. The eventual property allows us to solve the problem in asynchronous systems with crash or even Byzantine failures. Tseng's algorithm in a dynamic system introduced a *History* variable to store previous values. Tseng's work also showed the importance to clarify the notion of "nodes currently in the system" and n_t in a dynamic system. Nodes that do not execute **Join** function properly at the beginning of round t are not considered to be in the system in round t. Similarly, nodes that do not properly execute **Leave** function at the beginning of round t, including crashed nodes, are still in the system but are faulty nodes in round t. Besides, nodes that do not leave properly, including Byzantine nodes, are considered faulty nodes.

A dynamic system is desirable when it may not be possible or convenient to stop the entire system to allow modification to part of its hardware or software in a large distributed system. Hence, Hao et al. [28] proposed a Dynamic PBFT based on the first practical Byzantine-fault-tolerant protocol (PBFT), so that users could add or take out any node without stopping the whole system. They assume that each node has three states: *Benign, Absent* and *Malicious.* The state of a new node should be initialized to *Benign.* If a node exit actively, the state will change to *Absent.* There is a primary node that is similar to the sender in a broadcast protocol. The primary node can change in a dynamic PBFT model. The **join** function is a three-phase protocol message: (1) new node j registers at the primary node; (2) new node j multicasts a request message to all replicas; (3) nodes verify the signature upon receiving the request. Node **Exit** has two parts: active exit and passive exit. Active exit involves a similar three-phase protocol as **join()**. For passive exit, the system protocol clears them out and not allow them to join anymore. Instead of considering nodes that do not leave properly as faulty nodes as Tseng did, Hao's **leave** protocol makes sure that all nodes leave either actively or being cleared out of the system.

Dynamic system configuration can modify and extend system while it is running. Kramer and Magee [11] introduced a model that permits dynamic incremental modification and extension without stopping the unaffected parts of the system. Previous works either have new nodes join/leave the network, or change the specialities of nodes in the network. However, Kramer and Magee's model can have new nodes join, change existing nodes and even modify new nodes. Specifically, changes to the system include the introduction of new components, modification of new components, and provision of different interconnection patterns.

Changes have to be validated so that they are compatible with the existing system. Based on the dynamic configuration, we can determine the properties required by languages and their environments to support the dynamic configuration.

The following table gives an overview on models that we have discussed so far:

Model	Assumptions	Resilience	Model	Key techniques
BA [13]	No cryptography: Digital signatures:	$f < n/3$ $f < n/2$	synchronous	Quorum mechanism
BB [12]	No cryptography: Digital signatures:	$f < n/3$ $f < n$	synchronous	Quorum mechanism
Reliable Broadcast [4]	No cryptography	$f < n/3$	synchronous	Quorum mechanism
PBFT [16]	PKI	$f < n/3$	asynchronous	Quorum mechanism/ view change
BCB [18]	PKI	$f < n/3$	synchronous	expander graph
Overlay [27]	PKI	$f < n$	asynchronous	gossiping/ failure detector
Eventual consensus [14]	fair-loss channel	$f < n/3$	asynchronous	*History* variable
Dynamic PBFT [28]	PKI	$f < n/3$	asynchronous	Quorum mechanism/ view change

Chapter 5

Bitcoin/Blockchain

Many practical peer-to-peer systems such as Bitcoin is a dynamic system. Nakamoto [19] introduces a peer-to-peer version of eletronic cash that allows online payments to be sent directly from one party to another. This is possible because messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will. Due to the success of Bitcoin, blockchain technologies are taking the world by storm. All nodes in the blockchain have equal status. Theses nodes achieve consensus by using the prior agreement of the rules and following the principle of majority dominance.

The blockchain technology is built on top of four fundamental building blocks, and each block has key properties achieved through specific mechanism: (1) Identifying the source and destination of a transaction: Users serve from digital identities called "address" to send and receive transactions. (2) Transactions: Transactions are generated by the sender and broadcasted to the network of peers. Transactions are invalid unless they have been recorded in the public history of transactions, the blockchain; (3) Condition for auto-processing a transaction: The transfer of any value with the blockchian or the execution of any function through the blockchain should be locked by a logic conditions; (4) Consensus: updates must be agreed by all parties. Outchakoucht et al. [1] claimed that blockchain combined with IoT (Internet of Things) is of great importance for blockchain in the future. He proposed a dynamic and fully distributed security policy based on blockchain. To make the system more secure, he adopted an authorization process where new nodes register, request access to nodes in the blockchain.

In addition to security of blockchain system, we also need to solve Byzantine Generals Problem in the applications of blockchain. PoW (Proof of Work) is the consensus algorithm used in bitcoin. Its main idea is to allocate the accounting rights and rewards through the hashing power competition among the nodes. Nodes in the blockchain agree on an ordered set of blocks, each containing multiple transactions. Furthermore, transactions are grouped into blocks which are then chained together. A ledger is a data structure that consists of an ordered list transactions. A blockchain starts with some initial states, and the ledger records entire history of update operations made to the states. Bitcoin's hash rate refers to the amount of computing and process power being contributed to the network through mining. Hash of the whole data of the previous block, including the hash pointer to the block before that one, is stored in the hash pointer. Based on the information of the previous block, the different nodes calculate the specific solution of a difficult mathematical problem. The first nodes that solves this problem can create the next block and get a certain amount of bitcoin reward.

Nakamoto [19] used HashCash to design the mathematics problem in bitcoin. The specific calculations are: (1) Get the difficulty value that is dynamically adjusted to the hash rate of the whole network; (2) Collect transactions on the network after the production of the last block; (3) Traverse the current target hash value from 0 to 2^{32} and calculate the double SHA256 hash value in step 2; (4) If the node can't work out the hash value at a certain time, it repeats step two. Since the newly created block is linked to the blocks in front of it, the length of the chain is proportional to the amount of workload. All nodes trust the longest chain. If a Byzantine node wants to tamper with the blockchain, he needs to control more than 50% of the world's hashing power to ensure that he can become the first one to generate the latest block and master the longest chain. PoW works efficiently in a dynamic system as the difficulty of the mathematics problem is dynamically adjusted to the hash rate of the whole network. Unlike other models that have strict rules on join/leave protocols in a dynamic system, PoW lets new nodes join easily but makes it difficult for malicious nodes to attack the system. Hence, the PoW can effectively help the system reach consensus in a dynamic system.

The PoS (Proof of Stake) model uses a different process to confirm transactions and reach consensus. PoS is first seen in PPCoin [23], where the digital currency has the concept of coin age. Coin age is the coin value multiplied by the time period after it was created. The longer one holds the coin, the more rights it can get in the network. The accounting rights are allocated based on the formula *proofhash<coin age* target*, where *proofhash* is a composed hash value. If a someone were to attack, he will need to accumulate a large number of coins and hold them long enough to attack the blockchain. While PoW rewards its miner for solving complex equations, in PoS, the individual that creates the next block is based on how much they have 'staked' based on coin age. A winner who creates the next block is randomly chosen based on the amount nodes have staked. Nodes with higher coin age have high chances to be selected. Hence, nodes do not have to compute a mathematic problem like they do in PoW so PoS helps the system achieve consensus with less resources wasted.

Besides, PBFT can also be applied as a consensus algorithm in blockchain. The blockchain using PBFT consists of $3f+1$ server nodes and each node needs to collect at least $2f+1$ messages in the communication. This means the blockchain system can only tolerate 33% malicious nodes. Since the nodes need to communicate with every node to reach the agreement, the scalability is limited. As a result, PBFT is more suitable for a blockchain system with a small number of nodes.

Chapter 6

Conclusion

In conclusion, Byzantine consensus problems are challenging to study but they have important applications such as Bitcoin. Most early works focused on studying BB/BA protocols under a synchronous setting, which is unrealistic in practice. By using important techniques such as threshold signature scheme and having multiple rounds, Byzantine consensus protocols became more resilient and robust. To accommodate real life situations where it is not possible to stop the entire system to allow modification, Byzantine consensus models with different *join*/ *leave* protocols in a dynamic system were introduced. For the important application of dynamic system−−blockchain, various consensus algorithms including Proof-of-Work and Proof-of-Stake have been introduced. Such consensus algorithms in blockchain make it extremely difficult for malicious nodes to attack the system. With the growing popularity of Bitcoin, the study on Byzantine problems in a dynamic system will attract more attention.

Bibliography

[1] H. Es-samaali A. Outchakoucht and J.P. Leroy. In: *International Journal of Advanced Computer Science and Applications, Vol. 8, No.7, 2017* ().

[2] Michael Ben-Or. "Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols". In: *In Proceedings of the second annual ACM symposium on Principles of distributed computing, pages 27–30* (1983).

[3] G. Bracha. "An asynchronous $[(n-1)/3]$-resilient consensus protocol". In: *Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC)* (1984).

[4] G. Bracha and S. Toueg. "Asynchronous consensus and broadcast protocols". In: *Journal of the ACM* (1985). DOI: `https://doi.org/10.1145/4221.214134`.

[5] M. Bravo, G.Chockler, and A. Gotsman. "Making Byzantine Consensus Live". In: *34th International Symposium on Distributed Computing (DISC 2020)* (2020). DOI: `10.4230/LIPIcs.DISC.2020.23`.

[6] C. Cachin and S. Tessaro. "Asynchronous verifiable information dispersal". In: *Asynchronous verifiable information dispersal," 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)* (2005).

[7] N. H. Vaidya D. Sakavalas L. Tseng. "Asynchronous Byzantine Approximate Consensus in Directed Networks". In: *PODC'20: Proceedings of the 39th Symposium on Principles of Distributed Computing* (2020).

[8] Y. Moses F. Kuhn and R. Oshman. "Coordinated consensus in dynamic networks". In: *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing* (2011).

[9] M. Fischer and N. Lynch. "A lower bound for the time to assure interactive consistency". In: *Information processing letters, 14(4):183–186* (1982).

[10] G. Pandurangan J. Augustine and P. Robinson. "Fast Byzantine agreement in dynamic networks". In: *Proceedings of the 2013 ACM symposium on Principles of distributed computing* (2013).

[11] J. Kramer and J. Magee. "Dynamic Configuration for Distributed Systems". In: *IEEE Transactions on Software Engineering* (1985).

[12] R. Shostak L. Lamport and M. Pease. In: *Concurrency: the Works of Leslie Lamport* (2019).

[13] Robert Shostak Leslie Lamport and Marshall Pease. "The Byzantine Generals Problem". In: (1982).

[14] L.Tseng. "Eventual Consensus: Applications to Storage and Blockchain". In: *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)* (2019), pp. 840–846. DOI: 10.1109/ALLERTON.2019.8919675..

[15] T. Nowak M. Fugger and M. Schwarz. "Tight Bounds for Asymptotic and Approximate Consensus". In: *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing* (2018).

[16] B. Liskov M.Castro. "Practical Byzantine Fault Tolerance". In: *OSDI* (1999).

[17] M.S. Paterson M.J. Fischer N.A. Lynch. "Impossibility of distributed consensus with one faulty process". In: *Journal of the ACM* (1985).

[18] A. Momose and L. Ren. "Optimal Communication Complexity of Byzantine Consensus under Honest Majority". In: (2020).

[19] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash system". In: *bitcoin.org* (2008).

[20] Michael O. Rabin. “Randomized byzantine generals”. In: *In Proceedings of the 24th Annual Symposium on Foundations of Computer Science, pages 403–409* (1983).

[21] M. Reiter. “Secure agreement protocols: Reliable agreement in the presence of faults”. In: *Proc. 2nd ACM Conference on Computer and Communications Security* (1994).

[22] N. Santoro and P. Widmayer. “Agreement in synchronous networks with uniqitous faults”. In: *Theor. Comput. Sci* (2007).

[23] S.King and S. Nadal. “PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake”. In: (2012).

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

[25] B. Weiss U. Schmid and I.Keidar. “Impossibility results and lower bounds for consensus under link failures”. In: *SIAM J.Comput* (2009).

[26] M. Segal V. Drabkin R. Friedman. “Efficient Byzantine broadcast in wireless ad-hoc networks”. In: *2005 International Conference on Dependable Systems and Networks (DSN’05)* (2005).

[27] M. Segal V. Drabkin R. Friedman. “Efficient Byzantine Broadcast in Wireless ad-hoc Networks”. In: *2005 International Conference on Dependable Systems and Networks (DSN’05)* (2005).

[28] L. Zhiqiang L. Zhen X. Hao L. Yu and G. Dawu. “Dynamic Practical Byzantine Fault tolerance”. In: *2018 IEEE Conference on Communications and Network Security (CNS)* (2018).