Rapid Blockchain Scaling with Efficient Transaction Assignment

Panagiotis Drakatos, Eleni Koutrouli and Aphrodite Tsalgatidou

August 26, 2021

Rapid Blockchain Scaling with Efficient Transaction Assignment

Panagiotis Drakatos*, Eleni Koutrouli*, Aphrodite Tsalgatidou *
*National and Kapodistrian University of Athens, Dept. of Informatics and Telecommunications
{pandraka,ekou,atsalga}@di.uoa.gr

***Abstract*—The abrupt development of cryptocurrencies and blockchain technology has pointed to the importance of deploying large scale, highly robust Byzantine fault-tolerant schemes to handle critical distributed failures in system networks. Although traditional wisdom is to build systems by following synchronous protocols such as PBFT consensus, these protocols strongly rely on network time limitations and only guarantee liveness when the nodes behave arbitrary. This paper presents the design and implementation of Adrastus, a scalable blockchain system. The main contributions of Adrastus are the use of a consistent hashing mechanism which solves load balancing problems and the efficient assignment of transactions on parallel regions of single-chain consensus systems, referred to as zones, without introducing unnecessary overhead. We claim that the Adrastus blockchain system scales linearly without compromising system security. We present theoretical analysis, discuss our solution, and examine the conditions to meet both safety and liveness of our fault-tolerant system.**

***Index Terms*—Blockchain, Consensus Protocol, Byzantine Fault Tolerance, Scalability, Distributed Systems, System Networks**

I. Introduction

Nowadays cryptocurrencies [1] have been proved as a promising infrastructure for pseudonymous online payments [2]. The blockchain consensus protocol [3] gives the opportunity to every running node to confirm user transactions periodically, batched into smaller patches called blocks. The protocol is responsible for ensuring that all honest nodes conform with the rules and agree on the same total ordering of the confirmed blocks, so the set of the confirmed blocks grows over time.

Blockchain is not just a new technology; it is an ensemble of three well-known areas of the computer industry: game theory, cryptography, and software engineering [4]. The Bitcoin protocol [5], a peer-to-peer electronic cash system proposed in 2008 has introduced some of the fundamental key factors for modern blockchain applications, while it keeps gaining more popularity and attracting attention.

However, low transaction throughput has negative impact on the scalability of cryptocurrency systems and leads to seeking for new opportunities for increasing the numbers of user transactions. Likewise, network latency is the root problem that is highly engaged in the throughput issues. We measure the processing throughout as Transactions per Second (TPS). For example, Visa [6] is capable of handling an average of 2000 TPS with a maximum rate of 4000 TPS whereas, Bitcoin [5] and Ehereum [7] process only 10 TPS on average, which is considered less than 0.2% of the average available bandwidth measured in the P2P networks [8]. Many research efforts focus on solving and improving transaction throughput, resulting in a list of interesting designs for blockchains [8] [9].

Although blockchain technology offers great promises for solving various issues in the computer science industry, it faces various technical challenges that need to be addressed. For instance, even if high throughput is achieved as stated above, workloads require fast communication, descent storage and surely enough computation power, a fact that sets new preliminaries to the consensus node that wants to participate in order to achieve consistency. In this paper we present a scalable blockchain system, named Adrastus[1] which intends to achieve incremental scalability of its consensus protocol while limiting the resource usage of communication storage, computation, and memory cache management.

Additionally, there are further challenges that need to be addressed in order to create a robust and scalable blockchain network infrastructure. The following requirements remain open issues in most of the current well-known present technologies:

- **High scalability:** The ability to work efficiently with increasing numbers of nodes.
- **Security Robustness:** The ability to prevent different kinds of attacks and preserve the security of the transactions.
- **Efficiency:** Performing all the required operations with minimal computational requirements and without the excessive energy consumption.
- **Applicability:** The ability to be used in actual applications and deal with existing challenges.

In this work, we aim to address mainly three of the above challenges: scalability, security, and efficiency. We use the terms "Adrastus BFT consensus" to denote the process where validators attempt to reach a consensus and "Adrastus system" or "Adrastus network" to denote the entire environment of zones,validators and others entities participating in the protocol. We claim that the Adrastus system supports meaningful improvements in each of these issues. More specifically, the network is partitioned in asynchronous zones where each zone is responsible for validating assigned users' transactions. The

[1]The name of our system was inspired by the legendary king of Argos, Adrastus, who was joined by six more leaders and created an army to commence the war of the Seven against Thebes. This name constitutes a union symbol for multiple entities and was therefore, chosen for our system.

assignment process is accomplished under the supervision of a consistent hashing mechanism to distribute transactions load across multiple asynchronous zones. In a consistent hashing mechanism [10], the output range of a hash function is considered as a circular space with fixed values forming a ring. Subsequently, each asynchronous zone is assigned with a random value based on the hash algorithm which determines the position in the circle. In addition, when a new transaction enters the Adrastus network, a unique key is assigned, and it is identified by hashing the transaction key to yield its position on the circle. We have used a synthesis of well-known algorithms to achieve efficiency and scaling by implementing techniques from the design of the Amazon Dynamo database [11], a distributed storage system that has a fast and predictable performance with linear scalability.

The Adrastus has been designed to be scalable and efficient, to achieve safety and liveness and to treat failure without impacting the availability or performance of the entire network, as will be presented in the following sections. Hence, the key contributions of our work are the following:

- We introduce the usage of the consistent hashing mechanism as our main solution to solve load balancing problems and to accomplish the efficient assignment of transactions on parallel regions, referred to as zones, without introducing unnecessary overhead.
- We present the AdrastusBFT protocol that needs fewer communication rounds to achieve consistency and thus we lower the communication latency from $O\big(n^2\big)$ to $O\big(n\big)$ compared with the PBFT [12]. In addition, we include BLS [13] aggregated signature for faster operations on data exchanged.
- We propose an efficient technique based on VRF/VDF random functions to achieve randomness and randomly assign validators into zones.

The rest of the paper is organized as follows. In Section II, we introduce the proposed system architecture while the proposed consistent hashing algorithm and other optimizations are presented in Section III. In Section IV, we analyze the security obtained from the proposed system, in terms of safety and liveness of the protocol. Related work is discussed in Section V. Finally, Section VI concludes the paper.

II. System design

This section introduces the design of the Adrastus system. To begin with, the nodes in our system represent the active validators that have the main responsibility to totally engage in cooperative tasks. The validators are grouped in clusters after we partition the network to create smaller groups. The validators' main role is to verify and transmit user transactions and ensure system consistency. Furthermore, they are responsible for running the consensus protocol, adding blocks to the network, maintaining the current state, and being actively rewarded for their services. In the rest of this paper, we assume that we have a fixed subset of nodes that behave arbitrarily, called Byzantine nodes. The rest of the network consists of all the other nodes known as honest nodes where they compose and follow the protocols according to the specifications. We assume that time is divided into generations. We define a generation g as the fixed time between global configuration events that take place in the network. The time during a generation g is denoted as rounds r. During an individual round, each zone along with its validators has the main responsibility to process users' transactions and finally construct and reach a consensus on the block that they will create in round r according to the Adrastus Byzantine Fault Tolerant (AdrastusBFT) consensus protocol. In the Adrastus protocol, we introduce the main chain directory committee. The main chain directory committee is responsible to combine all micro transaction blocks of each zone into a higher transaction block namely key block which contains all the appropriate information for blocks of all zones at a specific round.

A. *Validators participation*

One of the attacks that every blockchain needs to defend is the Sybil attack [14]. This kind of attack is so important because of its nature that lets an attacker poison the blockchain network and subverts the reputation system that is built by creating a large number of pseudonymous online identities, they used them to gain control of the network for their own benefits. Satoshi's Nakamoto Bitcoin network depends upon miners to solve cryptographic puzzles (PoW) [5] before proceeding with the agreement on the next block, thus avoiding malicious registrations. The Adrastus system adopts a hybrid approach by combining the PoS consensus protocol [15] with the AdrastusBFT consensus protocol (described in section III-D) to prevent malicious attackers from adding nodes to the network which they will not comply with the protocol. If someone wants to participate in the Adrastus network as a representative validator node it demands to stake a number of tokens and wait for verification. Tokens represent an amount of money that the Adrastus blockchain requires from users or from the validator to stake, in order to prove that they are eligible nodes and that they will not easily trick or circumvent the protocol. They are thus used as a trust mechanism. The number of tokens that they stake will regulate the number of voting power that they earn in order to participate in the consensus protocol and get their rewards. We can consider the voting power as a virtual ticket that gives the appropriate permission to the validator to join the consensus and cast their vote. The amount of tokens required for a voting power is algorithmically calculated based on the height of the blockchain and on the active validators. The validators who have staked more amount of money have more chances to be elected as representative organizers because they are considered more trust. At the beginning of each generation, new validators' virtual tickets are randomly assigned to zones. Hence, the new validators join the zones where their voting power has been assigned.

B. *Adrastus Leader election process*

The Adrastus consensus algorithm relies on specific nodes, the supervisors to initiate the protocol and organize the

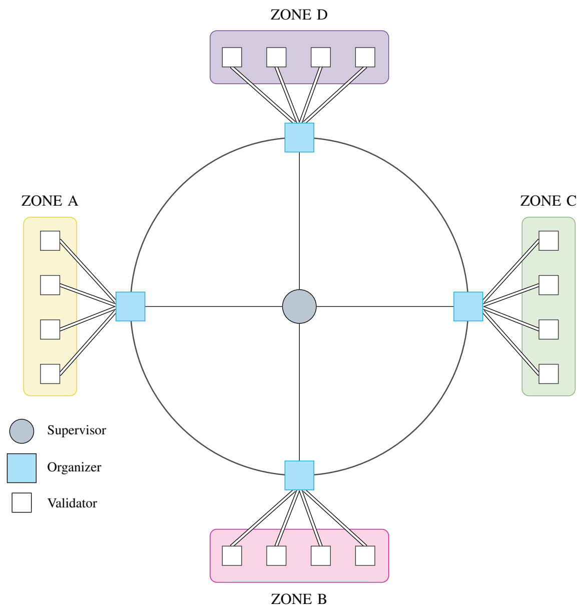

Figure 1: Adrastus System overview.

other nodes of their zone. For that purpose, at the beginning of each generation g, each validator computes a virtual ticket $ticket_{i,e,v} = VRF_{ski}("Supervisor"||struct_e||v)$ where $struct_e$ is the structure containing all the appropriate registered validators and v is the current view number. The current view number is used to count the number of the height of the blockchain protocol. Nodes rely on view number to always get coordinated on the same number. If for any reason they fail to synchronize they agree to increment the view number by one and then they try to reach again the consensus. The validators gossip these tickets between each other based on a specific time-bound δ. After that, they come to an agreement at the lowest valid value they have seen so far from the issuer who constructs it, and they recognize the issuer as the corresponding supervisor of generation g. The Adrastus makes the following assumption: In case that a random supervisor at generation g is not an honest node and thus it behaves maliciously and fails to further obey to the correctness of the protocol agreement, the validators simply ignore him for the rest of the generation g.

C. Randomness Generation on VRF/VDF Randomness

In this subsection, we introduce the protocol that we use to allow validators to reach an agreement on a single ticket described in section II-B. More specifically, the protocol includes the following steps to assign validators into nodes. At the beginning of the protocol, the supervisor of a previously elected generation g sends a "start" message with the hash of the last valid block to the validators. The validators on their turn, after receiving the "start" message, begin to take action by computing a VRF (verifiable random function) [16] that provides publicly verifiable proofs of its output correctness. For this purpose, a validator i calculates a random number ri and a valid proof pi so that

$$(r_i, p_i) = VRF(Sk_i, H(B_{n-1}), v) \tag{1}$$

where sk_i corresponds to the secret key of validator i and v is the current view number that demonstrates the round of the algorithm. The supervisor waits until receiving a quorum of at least $f+1$ valid random numbers from validators who participated in the process and mixes them with a xor operation to get the final randomness $pRnd$. The final randomness pRND is a number that will be used as a seed to randomly assign the nodes to zone. This value is critical because we don't let attacker forecast it as it produced just in time. As a result the attacker has zero time to predict it. Furthermore, the supervisor runs the AdrastusBFT consensus protocol (discussed in section III-D) to reach a consensus on $pRnd$ and commit it in the block B_n.

Nevertheless, the protocol has not ended yet because we do not calculate the actual randomness of the block B_n. The value $pRnd$ is not the actual randomness that we use to elect a new supervisor for the next block, but instead, it will be used as an input into the VDF [16] verifiable delay function $(f : x \to y)$ to start computing the real randomness

$$Rnd = VDF(pRnd, T) \tag{2}$$

of the next block where T is the VDF difficulty which is set dynamically depending on the growth of the system. The reason that we benefit from VDF is that we want to add delay, because VDF takes a specific time to calculate a computational task, even on a parallel computer. The most interesting part which we achieve with this method is that we provably delay the revelation of the actual Rnd by adding delay on the next block time generation and no one, even if the supervisor of the previous block trying to generate the randomness, is not able to predict, or bias the election process until the time for the B_{n+1} blocks comes. Particularly, we have this strict security measure that no one is able to predict the next randomness to prevent any malicious node or supervisor that wants to behave arbitrarily, from owning the randomness seed and uses it for his own purposes for the next block during the process of attaching validators into zones. Shamir and Wagner [17] use this method to build efficient time lock puzzles and take advantage of the exponentiation in a group of an unknown order as the key property to use VDF function. This early revelation of Rnd cannot happen because it takes T time based on the difficulty to compute it. Hence, an attacker will not be able to bias the actual randomness.

Furthermore, after VDF has been computed, the output can be verified by anyone. The Adrastus VDF construction operates as follows: The setup algorithm $Setup(\lambda, T)$ consists of two outputs. A finite abelian group of undefined order G, and an efficient hash function $H : X \to G$. In addition, it must set the public parameters $pp := (G, H, T)$. The evaluation algorithm consists of three computations. (a) a computation of $y \leftarrow H(x)^{2^T} \in G$ by computing T squaring in G starting with $H(X)$, (b) a proof p and (c) the output(y, p).

The supervisor runs the AdrastusBFT consensus protocol (discussed in Section III-D) to reach a consensus on Rnd and commit it on the block B_{n+T}. The remaining question now is how validators can quickly $verify(pp, x, y, p)$ and check that the output y is correct. We have implemented a solution according to the proposal of Wesolowski [18] to prove that $h \leftarrow g^{2^T} \in G$. We let $g := H(X) \in G$ be the base hash function given as input to the VDF solver. In addition, we let $h := y \in G$ be the output of the VDF translated from $y \leftarrow H(x)^{2^T} \in G$ to $h \leftarrow g^{2^T} \in G$. The verifier checks that $g, h \in G$ and outputs $reject$ if not. Correspondingly, the verifier sends a random prime l sampled uniformly from $Primes(\lambda)$ to the prover. We define $Primes(\lambda)$ as the set containing the first 2^λ primes.

III. Adrastus Optimizations

In this section, we propose the consistent hashing mechanism as our base solution to find out the destination zone of a transaction in such a way that it is deterministic and eliminates the unnecessary bottleneck. We believe that this solution is the necessary key for adapting and overcoming the load balancing problems and efficiently assign transactions to zones. In addition to our main goal, our secondary objective is to find a process where the increasing or decreasing number of zones is negligible and consequently does not affect the overall performance. In the following subsections, we present the mechanisms that we introduce in our blockchain network to achieve our objectives outlined in the introductory section.

A. Optimal Zone Computation

In this section, we propose an optimal formula to calculate the number of zones N_{zone} in a generation $g_{i+1}(N_{zone,i+1})$. We define a bound threshold for the maximum number of transactions existing in a block as Θ_{TX}. Furthermore, we use the variable N_{opt} that represents the optimal number of validators into a zone. In addition, we adopt the variable $joinN_i$ that represents the total number of eligible nodes which are in the list to join the newly node pool. In addition, pos value is a positive number that represents how many validators can be accepted in a zone. Moreover, the value $N_{TR,i}$ adjusts the average number of transactions in the blockchain on all zones in generation g_i. The formula that we follow is that we change the total number of zones $N_{zone,i+1}$ in the network only if we face a utilization exceed by the total number of eligible nodes $joinN_i$. Therefore, if the number of validators exceeds the given threshold Θ_{TX} and the average number of transactions per block is larger than the given bounded threshold $N_{TR,i} \geq \Theta_{TX}$ then we call $nsplit$. On the other hand, if the number of validators drops down below the given threshold, then we call $nMerge$ as shown in the function OptimalZoneComputeN.

B. Optimal transaction assignment across zones

In distributed computing, load balancing distribution is the most common problem where dynamic environments should deal with it. The reason this problem occurred is that we need to split the traffic from the clients in a set of tasks to any available server which is capable of fulfilling and make the overall processing more efficient. Therefore, load balancers accomplish to minimize the load across multiple servers that cause a single point of failure in a single server. The first proposed idea is to achieve availability in a distributed environment issued by Amazon in a system called Dynamo [19], which was used for treating and handling network failures by eliminating unnecessary bottleneck. Particularly, the data are partitioned and replicated in virtual nodes so as to minimize the load on a single point of a server. In addition, Google introduced another system [20] that leans to the consistent hashing mechanism to minimize the negative impact of failures in the network; hence, the clients are capable of connecting to a stable server node.

Algorithm 1: OptimalZoneComputeN

```
Result: $N_{zone,i+1}$
$nsplit \leftarrow (N_{zone,i} + 1) * (N_{opt} + pos)$;
$nMerge \leftarrow (N_{zone,i} - 1)$;
if $joinN_i \geq nsplit$ and $N_{TR,i} \geq \Theta_{TX}$ then
    $N_{zone,i+1} \leftarrow joinN_{i+1}/(N_{opt} + pos)$
else if $joinN_i \leq nMerge$ then
    $N_{zone,i+1} \leftarrow joinN_{i+1}/N_{opt}$
```

Figure 2 depicts how the transactions are replicated in distributed zones. More specifically, we assume that we desire to assign clients (referred to as balls) to servers (referred to as bins) in such a way that none of the servers get overloaded. Following this, in the Adrastus distributed environment we wish to allocate a large part of the incoming transactions across zones. Consequently, we refer to the transactions as the balls and to the zones as the bins along with the current supervisor monitors each one of them. In order to further process and accomplish the assignment process, it is necessary to find a hashing style solution, where the given transaction hash id can efficiently find the correct zone to be included.

The solution to this scenario is described as the consistent hashing mechanism [10]. In this mechanism, the active transactions (balls) and zones (bins) are hashed onto a unit circle, in order to create a circular order of bins and balls. Assuming that no collisions are detected, a transaction ball is placed on the successor bin of the nearest neighbor, according to the hash equation, following a clockwise around the circle. One of the nice features of the consistent hashing mechanism [10] is that if a zone (bin) is down or faces a change view state protocol, the system just moves the user's transactions (ball) to the nearest neighbor zone (bin) around the circle. Therefore, rehashing the values and computing new value orders is not needed, thus any unnecessary latency or computation effort is eliminated. Hence, with b balls and m bins, and a random hash function h, it is expected that each bin will be filled with b/m balls. Transactions and zones are both hashed into a unit circle with a unique hash function that is fixed over time.

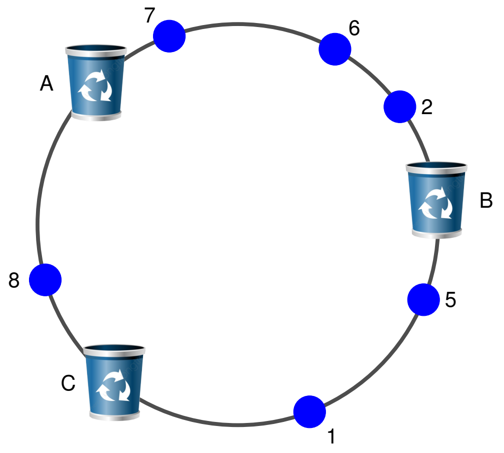

Figure 2: Example of a DHT-based Assignment of Transactions and Zones.

Our suggesting idea is that the [21] [22] consistent hashing mechanism could help us improve our blockchain network and give us the ability to assign transactions to zones on which we are able anytime to reassign a transaction in a zone if the zone is full and is not eligible to serve more tasks. This ability to move a transaction from one zone to another is achieved without rehashing the initial value. Let's suppose now that a new transaction is learned by validators and it is added to the blockchain network for thorough processing. This transaction is hashed and mapped to the zone following it in clockwise order. If the zone is full, then it is forwarded around the circle like a linear probing scheme until it finds a zone that is not full, and then it is placed in that zone so as to be verified and included in the tiny block later on.

Consider the example depicted in Figure 2 where 6 transactions and 3 zones are assigned on the circle, based on two different hash functions. We suppose that the capacity of each zone is a block of 2 transactions. We start allocating transactions in the increasing order of their entrance. Transaction number 1 moves clockwise and goes to zone C. Transaction number 8 goes to zone A. Transactions 7 and 2 go to zone B. Transaction number 5 goes to zone C; Later on, transaction number 6 arrives. It moves clockwise and hits zone B first. However, zone B has capacity 2 and already contains transactions 7 and 2. So transaction 6 keeps moving to reach zone C; however, zone C is also full. Finally, transaction 6 ends up in zone A that has a free slot for it.

In the following we explain how our system prevents double-spent attacks. Let us assume that a transaction t_1 has been created by a user u and the consistent hashing mechanism indicates that the nearest zone to be assigned is z_1. Furthermore, let's consider that a harmful user replays this transaction and creates an identical t_2 with the same id, a fact that simply means the assignment will also be on z_1 because the hash for the same id will produce the twin value in the circle near the zone z_1. The Adrastus network will prevent the execution of the malevolent transaction t_2 because we consider that every transaction is individual and depends on the nonce value which is an auto incremental value that never lets two transactions be executed simultaneously. Instead, the execution will happen with chronological order starting from the lowest nonce and rejecting all others since the t_1 is smaller compared with the t_2. Hence, by incrementing the nonce value, it is not possible for anyone to duplicate the same payment. We also make the assumption that the hash function for the transactions is independent from the hash function of the zones. In fact, with that limited range, we may observe hashing collisions. In order, to avoid this scenario, we reuse the same hash function k times with k different seeds in order to produce a k-independent way of hash function that is strong enough to avoid multiple collision detection. Furthermore, we make the following assumption: if two transactions or two zones hash to the same location on the circle, then the one with the lower id precedes the other. In addition, if a transaction and a zone hash to the same location, we assume that the transaction precedes the zone.

C. Optimal formula for zones' load balancing capacities

A critical concept that we need to take care of is that the allocation should be balanced and not overflow transactions into zones. For example, we do not want any scenario where the random assignment of transactions to zones, based on the consistent hashing mechanism will fill some zones and the others will remain empty. This kind of scenario would be harmful to system performance and to system scaling due to not accomplishing a desirable throughput. Allocation errors can become challenging because when the Adrastus network becomes overflowed with users' transactions, the capacity of each zone reaches constraint limits.While a consistent hashing scheme [21] minimizes the expected number of transaction movements, there is still a high probabilistic chance for allocation errors to occur.

In order to tackle the aformentioned issues, we firstly, introduce a capacity for each zone (bin) that can not exceed a specific limitation. Let's suppose a given load balancing parameter $c = 1 + \epsilon > 1$. We want to ensure that no zone has more than $[cb]$ transactions, when transactions are created and learned from validators. Assuming a linear ordering of the transactions, we let the lowest zones have a capacity with an upper bound at $[cb/m]$ with the respect that the total capacity only needs to change at most $[c]$ zones capacities. We refer to the former zones as the safer zones, which are chosen based on the reputation the supervisors are built upon, and the latter as the small zones. Moreover, we do not let the capacity to drop below a specific threshold. For example, let's consider a scenario where there is a fixed set of zones z, all of which have capacity c and each of t transactions provide u uniformly choice between the zones. The most desirable question is how large the capacity c should be before it becomes overloaded. An interesting solution that fits our needs [22] proves that by using $u = \mathrm{O}\big(log(1/\epsilon)\big)$ choices, we can assign transaction to only $z = (1 + \epsilon)t$ zones, so that the maximum load of 1 is at most $1 + \epsilon$ times the average load. In addition, to introduce more efficient insertions, it is proven that if $u \geq 5 + 3ln(1/\epsilon)$, then a transaction can be assigned in $u^{\mathrm{O}\left(log(1/\epsilon\right)} = (1/\epsilon)^{\mathrm{O}\left(log(log(1/\epsilon))\right)}$ expected time and in $\mathrm{O}\left(n\right)$ assumption time [21]. We assume that each node equipped with the above dynamic algorithm calculates the capacity of each zone an after they come on communication to reach consensus and agree on the number of nodes that will be assigned to each zone.

D. AdrastusBFT Consensus Protocol

A distributed system can be considered as a blockchain network that needs to follow specific rules in order to survive from common failures such as crashes. The Byzantine fault tolerant system [12] was the first system used to eliminate malicious nodes in a network and prevent arbitrary failures of its components by giving the opportunity to the network nodes to take critical actions and change their state to prevent or handle losses and recover from crashes. The Adrastus network can be considered as an asynchronous distributed system whose main responsibility is to guarantee the safety and liveness (described in section IV), even though there are specific nodes that behaved arbitrarily from the protocol. In an asynchronous P2P network, nodes need to communicate with each other to reach a consensus by following a common replication log. As a result, our system needs at anytime ensure that the data transmitted in the network cannot be tampered, delayed, or duplicated by random attackers. For that reason, we use cryptographic techniques to protect the integrity of our data and to prevent spoofing, replay attacks, or detect corrupted messages. Moreover, the nodes are demanded to agree in a common signature called aggregated signature which will be used to verify the integrity of the data transmitted through the Adrastus blockchain network.

The aggregated signatures have been proven very useful because they are cheap and have the potential to save much computation time, and therefore improve the overall performance of the entire network [23]. ECDSA [24] signatures have proved the effectiveness so far. They are good and they are widely used, but the problem with them is that signatures or keys cannot be combined and as a result, every signature has to be verified independently. The main idea of the multi-signature schemes was inspired by ByzCoin BFT [25]. ByzCoin uses communication trees based on collective signing algorithms to collect a multi aggregated signature. Another solution proposed is the EC-Schnorr [26] multi signatures schemes. The block validation becomes faster, but the problem which still exists is that multisig scheme requires several communication rounds. Furthermore, we have to rely on a random number generator. BLS [13] signatures can fix the above problems because they do not need random numbers and also all signatures in the block can be combined to a single signature.

Following this, we mix the BLS [13] signatures with the AdrastusBFT consensus protocol to reach consensus on a single message, with the produced aggregated signature. We follow three phases on each asynchronous zone to validate a

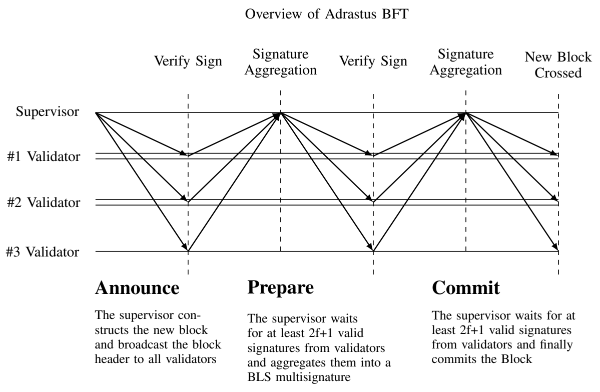
Overview of Adrastus BFT
Verify Sign
Signature Aggregation
Verify Sign
Signature Aggregation
New Block Crossed
Supervisor
#1 Validator
#2 Validator
#3 Validator
Announce
The supervisor constructs the new block and broadcast the block header to all validators
Prepare
The supervisor waits for at least 2f+1 valid signatures from validators and aggregates them into a BLS multisignature
Commit
The supervisor waits for at least 2f+1 valid signatures from validators and finally commits the Block

Figure 3: Adrastus Network communication during a single round of consensus on each asynchronous zone.

block: the announce phase, the prepare phase, and the commit phase. Figure 3 depicts a communication round of the block commitment along with agreement of the BLS signature for the given round.

Announce phase: The nodes on the asynchronous zone are informed about new transactions via a gossip protocol so, later on, these transactions are collected and sent to the current supervisor of each zone. When the current supervisor's timer expires, he constructs a new block with all the transactions that he has been informed so far and broadcasts the block header to the validators of this zone.

Prepare phase: Subsequently, the validators receive the announce phase message from the supervisor and they change their state to prepare phase. All honest nodes check the integrity of the block header and after that, they sign the block header with a BLS signature and send that signature back to the supervisor. The supervisor waits for at least $2f + 1$ valid signatures and aggregates them into a BLS multi-signature scheme. This guarantees that the message proposed by the leader is safe and consistent. Then, the supervisor broadcasts the aggregated multi-signature along with a bitmap structure indicating which validators were involved in the multi-signature process and including the public key and verifiable proofs of the signers $B[i] = 1$.

Commit: At this phase, the validators check that the multi-signature scheme has at least $2f + 1$ valid signers whose public key exists on the bitmap. In addition, they verify the transactions encompassed in the current block content which were broadcasted by the same supervisor during the announce phase. After that, they commit the new block along with the bitmap structure indicating which validators were involved in the process. The supervisor on its turn waits for at least $2f + 1$ valid signatures, and finally creates a bitmap with all the signers that participated in the commit process. In the end, the supervisor commits a new block in the blockchain with all the signatures and bitmaps attached. By this final step, we conclude the improvement process of the BFT protocol.

Table I demonstrates some well-known communication protocols footprints together with the Adrastus network. The

Table I: Consensus Protocol Complexity.

Protocol	Correct Leader	Leader failure(view-change)	leader failures
DLS [27]	$O(n^4)$	$O(n^4)$	$O(n^4)$
PBFT [12]	$O(n^2)$	$O(n^3)$	$O(fn^3)$
SBFT [28]	$O(n)$	$O(n^2)$	$O(fn^2)$
Tendermint [29]	$O(n^2)$	$O(n^2)$	$O(fn^2)$
LibraBFT [30]	$O(n)$	$O(n)$	$O(fn)$
AdrastusBFT	$O(n)$	$O(n^2)$	$O(fn^2)$

numbers that appear in this table, were the result of a literature survey on the distributed fault tolerance protocols, specifically, the asymptotic complexity of the different ways that these protocols use to deal with.

IV. Security Analysis

In this section, we describe how the Adrastus achieves the goals of randomness security, adaptive zone threshold, security of intra zones consensus, safety and liveness, which are presented in the following subsections.

A. Safety of Intra-Zones Consensus

As we mentioned before, the AdrastusBFT consensus protocol achieves safety in the network. With the term safety, we refer to whether or not the agreement on the same block height has been accomplished by all the validators who exist in the consensus process. We suppose that a supervisor the majority, of the time, conforms with the consensus protocol and collectively controls at least $n - t$ valid messages from all the given nodes of any generation g. Let's call that n the maximum number of validators and t the maximum number of corrupted nodes. In the following paragraphs, we first prove safety by assuming that all the validators are fixed over time and the randomness that we use is unbiaseable. Subsequently, we assume that:

Theorem 1. *For every two zones of validators in the same generation, there exists an honest node called supervisor that behaves as a leader and controls both zones.*

Proof. Let's assume that $H_i \geq N - f(i = 1, 2, ..)$ is the total voting power of each zone. The voting power H_i' of each zone, excluding the malicious Byzantine nodes, satisfies $H_i' \geq H_i - f \geq N - 2f$. We notice that if the two sets are disjointed, the voting power of the union $H_1' + H_2' \geq 2N - 4f \geq N - f$ exceeds the total voting power of all the validators. Following this, there exists an honest validator called supervisor, which controls the validators at both zones.

Theorem 2. *The AdrastusBFT consensus protocol achieves safety if the validators in an asynchronous zone have no more than $f > 1/3$ fraction of corrupted nodes.*

Proof. We prove the safety of a proposed blocked header during a random generation g. Let's assume that a supervisor node V is the first node that accepts a block header H_i for round $j > i$. This implies that this specific node has already broadcasted this header to other nodes of the zone. If another honest validator accepts a block header H_i' at round $j' > i$ there must be a valid accept message. Thus, H_i' is false for any non-faulty validator j(including $i = j$) such that $H_i \neq H_i'$. Consequently, this means that no supervisor can construct a safe prepare message for a different header other than H_i because validators accept a message only with a valid proof of certificate. Besides, the supervisor cannot obtain enough votes from honest validators to create an mf+1 proof. Finally, we prove that two non faulty nodes agree on the same block header at each round r of generation g.

B. Liveness

This subsection describes how liveness is achieved in the Adrastus system network. The term liveness corresponds to the meaning of how validators treat failures in the network. For example, we suppose that validators try to reach a consensus for a proposed block, and let's further assume that there is a fixed bound time δ when the operation must make progress. Our protocol needs to ensure if for any reason the reaching consensus process fails, the validators are required to move their current state to a new view and change the log based on a new round. Moreover, we guarantee that there is enough time for validators to wait for the view change protocol. In addition, we make certain that the duration of the view change protocol grows based on the block staking difficulty. We obtain liveness via the following methods

A validator multicasts a message for a view $r + 1$, and then he waits for at least $2f + 1$ messages for this view number $r+1$. Thus, he starts the timer for the view $r+1$ and waits for a time period T in order for the view to expire. If the timer expires before the validator receives $2f + 1$ messages for the view number $r + 1$, then it starts the new view number $r + 2$ but now, in this case, he waits for $2T + D$ time, where D depends on the block difficulty. Consequently, in this way, we avoid delays that become larger than the timeout period when the difficulty increases.

To prevent the nodes from starting the next view change protocol too late we preserve the following: we suppose a case when a random validator identifies a failure in the protocol and chooses to broadcast a new view change message to the other validators of the consensus. Let's further assume that this validator also receives a new view change message from the other nodes with a view number smaller than his view message. If the timer has not expired yet, he will wait to get confirmation until a specific quorum of $2f + 1$ messages is to be meet, in order to be sure that this was the lowest value message. Finally, when the timer expires he follows the smallest value that he has seen so far.

Theorem 3. *The AdrastusBFT consensus protocol achieves liveness if the zones have no less than $m \times f$ malicious byzantine nodes.*

Proof. We first prove that all honest validators on a zone accept all the blocks that are proposed, or they have already accepted them on their blockchain since they behaved honestly for the time of generation g where an organizer behaves honestly. Furthermore, we note that all the messages preserve the integrity and they are inviolated by using digital signatures. Since the organizers of a zone are chosen randomly based on the randomness generation with VDF and VRF, we expect an honest leader/organizer to be elected in each zone of every generation. Honest organizer will send valid proposals for all the proposed blocks to all validators with a valid proof of acceptance. This consequently means that some of the honest validators have either already accepted the proposed blocks or they will accept them since it is safe (as described in the theorem of safety). Hence, all honest validators will vote for the proposed block and as a result they will receive $mf + 1$ votes since there are $mf{+}1$ honest nodes. Therefore, all honest validators will achieve to reach a consensus and complete the voting process for the proposed block header at the end of each round for every generation.

V. Related Work

In the proposed work we aim at building a replicated state machine system, such as in [31], where a client generates and submits transactions and the system tries to fulfill them based on total ordering and append-only transaction log. Those systems are characterized by the atomic broadcast protocol [32]. In Bitcoin [5], this terminology is called the blockchain. Since, our work involves improvements in relation to existing blockchain systems, we explain below the most closely related efforts to ours.

Regarding fault tolerance, while Paxos [33] and Raft [34] and many other similar distributed systems tolerate failures in the network such as crashes, Byzantine fault-tolerant protocols such as PBFT [12] and AdrastusBFT tolerate even arbitrary, corrupted, or malicious nodes. However, even though BFT [12] protocols manage to tolerate byzantine nodes, they still rely on timing assumptions about the distributed network. Our work takes this approach further, offering improved performance by guaranteeing good throughput even in a fully asynchronous network. In general, the AdrastusBFT is evaluated in deployment scenarios where latency and CPU are bottlenecks. More specifically, the AdrastusBFT leverages PBFT [12] for the following reasons: our system consists of fewer communication rounds thus, we lower the communication latency from $\mathrm{O}\big(n^2\big)$ to $\mathrm{O}\big(n\big)$, depends on a unique signature aggregation, and has both linearity and responsiveness.

Bitcoin-NG [35] is a distributed fault-tolerant protocol designed to scale the blockchain architecture, which has inspired our work. Although Bitcoin-NG [35] increases the overall throughput, it is still vulnerableto some type of attacks [36], [37]. The Adrastus was designed and inspired by the idea of Bitcoin-NG [35]. Our architecture goes beyond the state of the art and it can be seen as an enhancement of the existing models, improving the performance and focusing on the achievement of better security, scalability, and robustness.

Regarding scalability, Chainweb [38] is another attempt to scale the Bitcoin consensus by maintaining multiple parallel shards. It requires multiple synchronous growths on all shards to periodically maintain the system consistency. Unlike, the Adrastus involves random-assigning validators to its z zones, and it can easily use as large z as needed to scale better. Section III-A has already discussed this effort. Spectre [39] confirms blocks without guaranteeing a total order, while the inclusive protocol [40] includes as many non-conflicting transactions as possible. These works provide weaker consistency notions than the Adrastus, which guarantees a total order.

Finally, OHIE blockchain system [41] is the first work so far, which claims that it can achieve consistency while maintaining a constant failure ratio of $f < 1/2$ compared with the Adrastus that needs $f < 1/3$. We give some inner thoughts for future work on the work of [42] which concerns routing communications over a sparse graph and mantains less than quadratic computation time.

VI. Concluding Summary

In this paper, we presented the design of the Adrastus system, Adrastus consensus protocol and the integrated consistent hashing mechanism which solves load balancing problems. Then, we explained how we determine bounds with respect to adding limit constraints on the capacity of each zone. Following this analysis, we examined functions to achieve randomness for zones and validators. We then verified our insights by proposing the AdrastusBFT consensus protocol for the construction of a secure scheme that has linear complexity. Additionally, we displayed the way our nodes obey to our protocol to guarantee safety and liveness.

Our discussion and theoretical analysis show that Adrastus can be a useful component in the cryptocurrency industry. We believe that our work demonstrates the promises of building a scalable cryptocurrency system inspired from traditional fault tolerant processing systems.

As a part of future work, we will evaluate the scalability and fault-tolerance of Adrastus by using different blockchain topologies such as different number of zones and a different number of validations per zone to measure the performance of the system in terms of throughput and latency. Furthermore, we plan to analyze some of the most common attacks against blockchain systems and evaluate the resilience of the proposed system concerning these attacks. Moreover, we are going to present the global state of Adrastus and give more emphasis to the components of the system.

References

[1] U. Mukhopadhyay, A. Skjellum, O. Hambolu, J. Oakley, L. Yu, and R. Brooks, "A brief survey of Cryptocurrency systems," in *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, Dec. 2016, pp. 745–752.

[2] N. Jonker, "What drives the adoption of crypto-payments by online retailers?" *Electronic Commerce Research and Applications*, vol. 35, p. 100848, May 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1567422319300250

[3] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," in *2017 IEEE International Congress on Big Data (BigData Congress)*, Jun. 2017, pp. 557–564.

[4] "Amazon.com: The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology (Audible Audio Edition): William Mougayar, Vitalik Buterin, Christopher Grove, Audible Studios: Audible Audiobooks." [Online]. Available: https://www.amazon.com/The-Business-Blockchain-audiobook/dp/B01LZ92RUS

[5] A. Berentsen, "Aleksander Berentsen Recommends "Bitcoin: A Peer-to-Peer Electronic Cash System" by Satoshi Nakamoto," pp. 7–8, 2019.

[6] "Small Business Retail." [Online]. Available: https://usa.visa.com/run-your-business/small-business-tools/retail.html

[7] C. Dannen, *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. Apress, 2017. [Online]. Available: https://www.springer.com/de/book/9781484225349

[8] A. E. Gencer, S. Basu, I. Eyal, R. van Renesse, and E. G. Sirer, "Decentralization in Bitcoin and Ethereum Networks," in *Financial Cryptography and Data Security*, S. Meiklejohn and K. Sako, Eds. Berlin, Heidelberg: Springer, 2018, pp. 439–457.

[9] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing," 2016, pp. 279–296. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kogias

[10] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, "Web caching with consistent hashing," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 31, no. 11-16, pp. 1203–1213, May 1999. [Online]. Available: https://doi.org/10.1016/S1389-1286(99)00055-9

[11] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 205–220, Oct. 2007. [Online]. Available: https://doi.org/10.1145/1323293.1294281

[12] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proceedings of the third symposium on Operating systems design and implementation*. New Orleans, Louisiana, USA: USENIX Association, Feb. 1999, pp. 173–186.

[13] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed. Berlin, Heidelberg: Springer, 2001, pp. 514–532.

[14] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "SybilGuard: defending against sybil attacks via social networks," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: Association for Computing Machinery, Aug. 2006, pp. 267–278. [Online]. Available: https://doi.org/10.1145/1159913.1159945

[15] S. King and S. Nadal, "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake," 2012.

[16] A. Bugday, A. Ozsoy, and H. Sever, "Securing Blockchain Shards By Using Learning Based Reputation and Verifiable Random Functions," in *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, Jun. 2019, pp. 1–4.

[17] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock Puzzles and Timed-release Crypto," 1996.

[18] B. Wesolowski, "Efficient verifiable delay functions," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 379–407.

[19] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007, publisher: ACM New York, NY, USA.

[20] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A Fast and Reliable Software Network Load Balancer," 2016, pp. 523–535. [Online]. Available: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eisenbud

[21] V. Mirrokni, M. Thorup, and M. Zadimoghaddam, "Consistent Hashing with Bounded Loads," Jul. 2017, arXiv: 1608.01350. [Online]. Available: http://arxiv.org/abs/1608.01350

[22] D. Fotakis, R. Pagh, P. Sanders, and P. Spirakis, "Space Efficient Hash Tables with Worst Case Constant Access Time," *Theory of Computing Systems*, vol. 38, no. 2, pp. 229–248, Feb. 2005. [Online]. Available: https://doi.org/10.1007/s00224-004-1195-x

[23] T. Ristenpart and S. Yilek, "The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks," in *Advances in Cryptology - EUROCRYPT 2007*, M. Naor, Ed. Berlin, Heidelberg: Springer, 2007, pp. 228–245.

[24] D. Johnson, A. Menezes, and S. Vanstone, "The Elliptic Curve Digital Signature Algorithm (ECDSA)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, Aug. 2001. [Online]. Available: https://doi.org/10.1007/s102070100002

[25] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing," Aug. 2016, arXiv: 1602.06997. [Online]. Available: http://arxiv.org/abs/1602.06997

[26] C. P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, Jan. 1991. [Online]. Available: https://doi.org/10.1007/BF00196725

[27] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988, publisher: ACM New York, NY, USA.

[28] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "SBFT: a Scalable and Decentralized Trust Infrastructure," *arXiv:1804.01626 [cs]*, Jan. 2019, arXiv: 1804.01626. [Online]. Available: http://arxiv.org/abs/1804.01626

[29] E. Buchman, "Tendermint: Byzantine Fault Tolerance in the Age of Blockchains," Ph.D. dissertation, 2016.

[30] M. Baudet, A. Ching, A. Y. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State Machine Replication in the Libra Blockchain," 2019.

[31] A. Bessani, J. Sousa, and E. E. P. Alchieri, "State Machine Replication for the Masses with BFT-SMART," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Jun. 2014, pp. 355–362, iSSN: 2158-3927.

[32] 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, pp. 158–179, Apr. 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0890540185710607

[33] L. Lamport, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[34] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," 2014, pp. 305–319. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[35] I. Eyal, A. E. Gencer, E. G. Sirer, and R. v. Renesse, "Bitcoin-NG: A Scalable Blockchain Protocol," 2016, pp. 45–59. [Online]. Available: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eyal

[36] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 375–392, iSSN: 2375-1207.

[37] A. Gervais, H. Ritzdorf, G. O. Karame, and S. Capkun, "Tampering with the Delivery of Blocks and Transactions in Bitcoin," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. Denver, Colorado, USA: Association for Computing Machinery, Oct. 2015, pp. 692–705. [Online]. Available: https://doi.org/10.1145/2810103.2813655

[38] M. Quaintance, "4 Scenario 1 : Full-braid replacement," 2018.

[39] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "SPECTRE : Serialization of Proof-of-work Events : Confirming Transactions via Recursive Elections," 2017.

[40] Y. Lewenberg, Y. Sompolinsky, and A. Zohar, "Inclusive Block Chain Protocols," in *Financial Cryptography and Data Security*, R. Böhme and T. Okamoto, Eds. Berlin, Heidelberg: Springer, 2015, pp. 528–547.

[41] H. Yu, I. Nikolić, R. Hou, and P. Saxena, "OHIE: Blockchain Scaling Made Simple," in *2020 IEEE Symposium on Security and Privacy (SP)*, May 2020, pp. 90–105, iSSN: 2375-1207.

[42] V. King and J. Saia, "From Almost Everywhere to Everywhere: Byzantine Agreement with $\tilde{O}(n^{3/2})$Bits," in *Distributed Computing*, I. Keidar, Ed. Berlin, Heidelberg: Springer, 2009, pp. 464–478.