Quadratic worst-case message complexity for State Machine Replication in the partial synchrony model

ANDREW LEWIS-PYE, London School of Economics, UK

We consider the message complexity of State Machine Replication protocols dealing with Byzantine failures in the partial synchrony model. A result of Dolev and Reischuk gives a quadratic lower bound for the message complexity, but it was unknown whether this lower bound is tight, with the most efficient known protocols giving worst-case message complexity $O(n^3)$. We describe a protocol which meets Dolev and Reischuk's quadratic lower bound, while also satisfying other desirable properties. To specify these properties, suppose that we have n replicas, f of which display Byzantine faults (with $n \geq 3f + 1$). Suppose that Δ is an upper bound on message delay, i.e. if a message is sent at time t, then it is received by time $\max\{t, GST\} + \Delta$. We describe a deterministic protocol that simultaneously achieves $O(n^2)$ worst-case message complexity, optimistic responsiveness, $O(f\Delta)$ time to first confirmation after GST and $O(n)$ mean message complexity.

1 INTRODUCTION

Protocols for State Machine Replication (SMR) [Lam78, Sch90] are used to ensure that a distributed network of 'replicas' (e.g. servers) can agree on an order in which to implement client-initiated service requests. Traditionally, such protocols have been studied in the 'permissioned' setting, which means that the protocol is carried out by a fixed set of replicas, all of whom are known to each other from the start of the protocol execution. Together with the related task of reaching Byzantine Agreement [PSL80, LSP82], SMR protocols are a central topic in distributed computing. For an up-to-date account of the latest advances in this area, see [CKN21].

Of direct relevance to the study of SMR protocols is the fact that recent years have seen a surge of interest in 'blockchain' technologies, such as Bitcoin [N+08]. The key factor differentiating Bitcoin from traditional SMR protocols is that it functions in a 'permissionless' setting, which means that the protocol operates over an unknown network of replicas that anybody is free to join. Despite this distinction, the surge of investment in blockchain technology has led to significant renewed interest in permissioned SMR protocols. On the one hand, this is because many of the envisaged blockchain applications actually pertain to fundamentally permissioned settings – a prominent example is the use of SMR protocols to facilitate international transfers between members of banking consortia. On the other hand, there are also now several established techniques (e.g. [CM16]) for implementing permissioned protocols in permissionless settings. The latest implementations of some crytocurrencies, such as Ethereum [But18], use permissioned SMR protocols [BG17] that are tailored in a simple way to instantiate a form of permissionless entry.

These new applications bring with them a focus on the need for permissioned SMR protocols that are both robust and scalable. Robustness means being able to deal with arbitrary (especially malicious) behaviour from a non-trivial proportion of replicas. In the parlance of the distributed computing literature, protocols that can deal with behaviour of this kind are called 'Byzantine Fault Tolerant' (BFT). Practical solutions may also need to tolerate unbounded periods of bad network connectivity and denial-of-service attacks. A standard way to model this is to work in the 'partial synchrony' communication model of [DLS88], which means that protocols are required to guarantee consistency at all times (i.e. that non-faulty replicas never disagree on the ordering of client requests), but are only required to make progress on processing new client requests during time intervals when message delivery is reliable, i.e. during intervals when messages are guaranteed to be delivered within some known time bound Δ.

Author's address: Andrew Lewis-Pye, Department of Mathematics, London School of Economics, London, UK, a.lewis7@lse.ac.uk.

The issue of scalability. PBFT [CL+99] is well-known as the first practical BFT SMR protocol for the partial synchrony model. When PBFT was first conceived, however, typical real-world SMR implementations would involve a number of replicas n in the single digits. Today, implementations are envisaged that use hundreds or even thousands of replicas. It therefore becomes vital to consider protocols that scale well with the number of replicas, and message complexity is an important metric in this regard.[1] A result of Dolev and Reishchuck [DR85] establishes a quadratic lower bound on the message complexity when the given bound f on the number of faulty replicas is $\Theta(n)$ (as we assume here). Many BFT SMR protocols have been developed since PBFT (e.g. Zyzzyva [KAD+10], Prime [ACKL10], SBFT [GAG+19], Upright [CKL+09], 700BFT [GKQV10], BFT- SMaRt [BSA14], Tendermint [Buc16, BKM18], Hotstuff [YMR+18], LibraBFT [BCC+19], Casper [BG17]), but none achieve worst-case message complexity better than $O(n^3)$.

The aim of this paper is to describe a BFT SMR protocol for the partial synchrony model that achieves worst-case message complexity $O(n^2)$, and so which is the first such protocol to meet the quadratic lower bound of Dolev and Reishchuk. The protocol we describe also satisfies other desirable properties: Optimistic responsiveness, $O(f\Delta)$ time to first confirmation after GST and $O(n)$ mean message complexity. These terms will be explained in Sections 1.1 and 1.2. In the context of the partial synchrony model, a protocol is said to have *optimal resiliency* if it can handle any number f of Byzantine faults, so long as $n \geq 3f + 1$ (this being optimal by a result of Dwork, Lynch and Stockmeyer [DLS88]).

Theorem 1.1. *Consider the partial synchrony model. There exists a BFT SMR protocol with optimal resiliency and worst-case message complexity $O(n^2)$. This protocol can simultaneously be made to satisfy optimistic responsiveness, to give $O(f\Delta)$ time to first confirmation after GST and $O(n)$ mean message complexity.*

Theorem 1.1 above is stated in terms of *message* complexity, i.e. the number of messages sent. A more fine-grained analysis is achieved by thinking in terms of *communication* complexity, i.e. the number of bits sent to confirm requests. The variable κ in the theorems below is a security parameter specifying the length of signatures and hashes. Theorem 1.2 gives an analogue of Theorem 1.1 in terms of communication complexity, but drops the requirement for $O(n)$ mean communication complexity. Theorem 1.3 can be seen as a corollary of Theorem 1.2. We will also give an independent proof, which is simpler than a direct proof of Theorem 1.2.

Theorem 1.2. *Consider the partial synchrony model. There exists a BFT SMR protocol with optimal resiliency and worst-case communication complexity $O(\kappa n^2)$. This protocol can simultaneously be made to satisfy optimistic responsiveness and to give $O(f\Delta)$ time to first confirmation after GST.*

Theorem 1.3. *Consider the partial synchrony model. There exists a protocol for Byzantine Agreement with optimal resiliency and worst-case communication complexity $O(\kappa n^2)$. This protocol can be made to ensure that all correct replicas terminate within time $O(f\Delta)$ after GST.*

1.1 Defining the model and metrics

The replicas. We consider a set of $n \geq 3f + 1$ replicas, indexed by $i \in \{0, \ldots, n-1\}$. Certain replicas may be *corrupted* by an *adversary*, and may then display Byzantine failures. We allow that the adversary can choose which replicas to corrupt in a dynamic fashion over the course of the protocol execution, so long as they corrupt a total of at most f many replicas. We will refer to the (at least $n - f$) replicas that are never corrupted as *correct*. Every pair of replicas has a bidirectional, reliable, and authenticated channel between them, i.e., each replica is able to send messages individually

[1] Message complexity will be defined formally in Section 1.1. Roughly, it is the worst-case number of messages that need to be sent to confirm requests once message delivery is reliable.

to each of the other replicas, these messages will eventually arrive, and the recipient can verify the sender's identity. When we say that a replica *broadcasts* a message, this means that it simultaneously sends the message to all replicas.[2]

Cryptographic assumptions. We assume the existence of a cryptographic signature scheme, a collision-resistant hash function h, a public key infrastructure (PKI) to validate signatures, and a threshold signature scheme [BLS01, CKS05, Sho00]. As is standard, we assume that replicas are polynomial-time bounded, which means that replicas are given a security parameter κ specifying the length of signatures and hashes, and may perform a number of basic operations during the protocol execution which is at most polynomial in κ. In particular, this means that the length of the protocol execution is bounded by a polynomial in κ. As in other SMR protocols [YMR+18], the threshold signature scheme is used to create a signature of size κ combining signatures from k of n many replicas. In our case, k will always take the value $n - f$. We suppose that hashes are unique during any protocol execution, i.e. for any two inputs m and m' given to the hash function, $h(m) = h(m')$ implies $m = m'$. We also restrict attention to protocol executions in which the adversary is unable to break the cryptographic schemes described above.

Communication models. Three standard communication models are:

(1) *The synchronous model.* All replicas begin the protocol execution simultaneously with access to a global clock. There is a known bound Δ on the time for message delivery.
(2) *The partial synchrony model.* The known bound Δ on message delivery only applies after some unknown *Global Synchronisation Time*[3] (GST), i.e. if a message is sent at time t then it arrives by time $\max\{\text{GST}, t\} + \Delta$. Replicas do not have access to a global clock, but can accurately measure local time, i.e. each replica has an accurate stopwatch/timer.[4] There is no requirement that replicas all start the protocol execution simultaneously, but all correct replicas are assumed to join prior to GST.
(3) *The asynchronous model.* Each message can take any finite amount of time to be delivered. Replicas do not have access to timers.

An important assumption in the description of the partial synchrony model above is that replicas do not have globally synchronised clocks. If replicas have access to globally synchronised clocks, then a simple approach to achieving the quadratic lower bound of Dolev and Reischuk in terms of message complexity (which works if one is willing to drop the requirement for optimistic responsiveness – see Section 1.2 for a definition of the latter term) is to implement a version of Hotstuff in which each view is carried out at a previously determined time according to the global clock. More generally, the problem of achieving the quadratic lower bound is interesting so long as *any* of the following three conditions hold: (1) Replicas do not have globally synchronised clocks; (2) There is a requirement for optimistic responsiveness; (3) We are concerned with communication (rather than just message) complexity. We will consider dropping all of these assumptions in this paper.

A seminal result of result of Fisher, Lynch and Paterson [FLP85] shows that, for the asynchronous model, deterministic SMR protocols are not possible and that no SMR protocol (deterministic or otherwise) is able to achieve unbounded values for the worst-case complexity measures we consider here. We work in the partial synchrony model and it will be convenient to think of the adversary as controlling message delivery subject to the constraints outlined above.

[2]It will be technically convenient to suppose that, when a replica broadcasts a message, it also 'sends' the message to itself. This just means that, for the purpose of carrying out its instructions, the replica immediately regards the message as having been received.
[3]The model is really intended to model a scenario in which consistency must be maintained at all times, and new requests must be confirmed during synchronous intervals of *sufficient length*. The use of the unknown Global Synchronisation Time is a standard technical convenience, which can be argued to be equivalent to these requirements. See [DLS88] for further details.
[4]This assumption is made for technical convenience, but it will be clear that the protocol we describe can easily be adjusted (without communication complexity cost) to handle scenarios in which there is a known upper bound on the ratio between the speed of the timers held by different replicas.

The blockchain. Rather than being concerned with the rate at which clients produce requests, we consider a protocol that produces chains of *blocks*, each of which is thought of as containing a constant bounded number of requests. Replicas begin the protocol execution with agreement on a unique *genesis* block. All blocks other than the genesis block must have a unique *parent* block, whose hash is specified in the block. If B' is the parent of B, then B' is also called a *predecessor* of B, and all predecessors of B' are predecessors of B. All blocks must have the genesis block as a predecessor – to avoid constant mention of replicas having to check for block 'validity', it will be convenient in what follows to assume replicas simply ignore blocks that cannot be verified to satisfy these conditions. Two distinct blocks are called *incompatible* if neither is a predecessor of the other.

Consistency and liveness. We consider a protocol that *confirms* blocks. The requirement for *consistency* is that no pair of incompatible blocks should ever be confirmed. The requirement for *liveness* is that arbitrarily many blocks will be confirmed if the protocol is run for sufficiently long after GST – for the protocols we describe here, we will be able to guarantee that each interval of length $O(f\Delta)$ after GST produces a new confirmed block.

The complexity measures. In the protocols we describe here, messages will contain *epoch/view* numbers. Since we assume that replicas are polynomial-time bounded, the written forms of epoch/view numbers are of length $O(\kappa)$.

It follows from the result of Fisher, Lynch and Paterson, mentioned above, that SMR protocols in the partial synchrony model cannot guarantee the confirmation of requests prior to GST+Δ. For this reason, the worst-case complexity measure described in the following definition counts the number of messages sent by correct replicas after GST+Δ. The maximum value in the definition is taken over all possible actions of the adversary and choices for GST (which we can suppose chosen by the adversary).

Definition 1.4. The worst-case message complexity is the maximum number of messages sent by correct replicas after GST + Δ and prior to the first block confirmation.

We also consider a version of Definition 1.4 which is modified in the obvious way to give the corresponding notion for communication complexity. Let B_i^* be the unique block (if it exists) which is the ith to be confirmed after GST + Δ. In the following definition, x_i denotes the number of messages sent by correct replicas after the confirmation of B_i^* and before the confirmation of B_{i+1}^*. The slightly circuitous wording of the definition stems from the fact that the adversary may prevent $(1/k)\sum_{i=1}^{k} x_i$ from moving towards a limit value as k increases, although it always remains bounded.

Definition 1.5. For $i \in \mathbb{N}_{>0}$, let x_i be defined as above. We say the mean message complexity is $O(n)$ if there exists some constant C and a function $k(n)$ such that, for all n and all $k > k(n)$, $\frac{1}{k}\sum_{i=1}^{k} x_i < Cn$ for all protocol executions in which at least k blocks are confirmed after GST+Δ.

1.2 Related work

In this paper, we are concerned with the partial synchrony model. PBFT was the first practical BFT SMR protocol for the partial synchrony model. As is now standard for BFT SMR protocols in the partial synchrony model, the protocol implements a sequence of *views*, with each view having a designated leader who is responsible for driving consensus on the next set of requests to be confirmed. Subsequent to PBFT a great number of BFT SMR protocols have been developed (e.g. Zyzzyva [KAD+10], Prime [ACKL10], SBFT [GAG+19], Upright [CKL+09], 700BFT [GKQV10], BFT- SMaRt [BSA14]) which work by using the same paradigm of leaders, views and view changes, but none achieve worst-case message complexity better than $O(n^3)$.

For PBFT-like protocols, such as those listed above, the complexity bottleneck tends to be the part of the protocol dedicated to *view changes* (which are required because leaders may be faulty). Recently, a number of elegant protocols, such as Tendermint [Buc16, BKM18] and Casper [BG17] have been described, which considerably simplify the leader replacement protocol. In the case of Tendermint and Casper, however, this simplicity is achieved at the expense of *optimistic responsiveness*. Informally, optimistic responsiveness [YMR+18] requires that, for a fixed input Δ, an unbounded number of blocks/requests can be confirmed in any time interval of fixed length after GST, so long as leaders are correct, and so long as messages are delivered and instructions carried out fast enough. So, when leaders are correct, the rate at which blocks are confirmed after GST depends on the actual speed of the system, and is not limited by high estimates for Δ (although liveness with Byzantine failures still depends on the bound Δ holding after GST).

The significant achievement of Hotstuff [YMR+18], was to simultaneously achieve optimistic responsiveness, while keeping a linear bound on communication complexity *within* each view. Liveness for Hotstuff, however, still requires implementing a protocol to ensure *view synchronisation*, i.e. to ensure that all replicas eventually spend long enough within the same view with correct leader. In Hotstuff, this task of view synchronisation is delegated to a separate *Pacemaker* module. The implementation of the Pacemaker module is left unspecified, although it is pointed out that a practically infeasible method of using expontially increasing 'timeouts' would suffice.

The task of achieving a general and efficient implementation of the Pacemaker module is addressed in a couple of recent papers by Naor et. al. [NBMS19] and Naor and Keidar [NK20]. The first of these two papers describes a version of the Pacemaker module named Cogsworth, which, when faced with benign failures, achieves $O(n)$ mean message complexity. In the face of Byzantine failures, however, Cogsworth requires $\Omega(n^2)$ messages per view change, which means that a sequence of $\Omega(n)$ many view changes before finding a correct leader requires sending $\Omega(n^3)$ many messages. The module described in [NK20] then improves on the ability to deal with Byzantine failures, achieving $O(n)$ mean message complexity in the face of Byzantine failures, and $O(\Delta)$ expected time to first block confirmation after GST (when implemented with a protocol such as Hotstuff). Since the protocol utilises a random process of leader selection it has unbounded worst-case message complexity, but if modified in the obvious way (using a deterministic sequence of rotating leaders) gives a deterministic protocol with worst-case message complexity which is $\Omega(n^3)$, with time to first confirmation after GST which is $O(f^2\Delta)$.

The study of protocols for Byzantine Agreement was introduced in [PSL80, LSP82], where it was shown that the problem can be solved in the synchronous model iff $f < n/2$ when a PKI is provided. A recent paper by Momose and Ren [MR20] describes a protocol with optimal resiliency and quadratic communication complexity for the synchronous model, thereby meeting the lower bound of Dolev and Reischuk [DR85] for that model. For the partial synchrony model, Dwork, Lynch and Stockmeyer [DLS88] showed that Byzantine Agreement can be solved iff $f < n/3$. The state-of-the-art in terms of message/communication complexity for the partial synchrony model is achieved by using SMR protocols to solve Byzantine Agreement. The most efficient previously known protocols have worst case message complexity $O(n^3)$ – this can be achieved, for example, by implementing Hotstuff with any Pacemaker module that has message complexity $O(n^2)$ per view-change. $O(n)$ *expected* message complexity can be achieved by combining Hotstuff with the Pacemaker module of Naor and Keidar [NK20].

We refer the reader to [CKN21] for a survey which also goes into detail re the state-of-the-art for probabilistic protocols in the asynchronous setting, and well as further results on protocols for achieving Byzantine Agreement.

2 THE PROTOCOL FOR THEOREM 1.1

As is standard, we describe a protocol for which the instructions are divided into a sequence of *views*. Within each view, the instructions will be essentially the same as for Hotstuff – where the protocol differs from Hotstuff is in the mechanisms used to achieve view synchronisation. We assume the reader is familiar with Hotstuff, but the role of Hotstuff in our protocol can really be entirely blackboxed. All one needs to understand about Hotstuff to follow the protocol we describe here, is that it achieves the following:

- Consistency: Incompatible blocks will never be confirmed so long as each individual replica never executes view instructions out of order, i.e. so long as no correct replica executes instructions for a view v and then instructions for a view $v' < v$.
- Liveness: Block confirmation will occur, so long as all replicas are eventually in the same view for sufficiently long and that view has a correct leader.

Since Theorem 1.1 refers to message rather than communication complexity, we don't really need to use a threshold signature scheme for the protocol of this section. We do so to facilitate our discussion in Section 4, which describes how to modify the protocol to deal with communication complexity.

2.1 The high level idea.

The basic plan is that we will spend $O(n^2)$ messages to synchronise, but will do so only every $f + 1$ views. Since at least one leader amongst those $f + 1$ views will be correct and will produce a new confirmed block (if the view is initiated after GST), and since the complexity of each view is $O(n)$, this results in $O(n^2)$ messages being required before a correct leader produces a new confirmed block.

2.2 Epochs.

As well as being divided into views, the instructions are divided into *epochs*. Each epoch consists of $f + 1$ views that are specific to that epoch. Each epoch therefore has $f + 1$ different leaders (one for each view within the epoch), at least one of which must be correct. Leaders are rotated, so that the leaders for epoch i are replicas $i \bmod n, \ldots, i + f \bmod n$. We consider three different kinds of *certificate*, each of which combines a set of $n - f$ signatures into a single threshold signature:

- QCs (quorum certificates) correspond to a specific round of voting within a view.
- VCs (view certificates) combine votes to begin a new view.
- ECs (epoch certificates) combine votes to begin a new epoch.

Just as for Hotstuff, each view has three stages of voting. We order QCs by epoch number, then by view number within the epoch, and then by stage number within the view. Each QC will *correspond* to a specific block. All votes and blocks are assumed to be signed by the sender, and to contain the corresponding epoch, view and stage number.

Before describing how to coordinate the views within an epoch, we'll first describe how to implement epoch changes. We start that way because understanding epoch changes does not require understanding the rest of the protocol, and because the instructions for epoch changes are very simple. This simplicity stems from the fact that we can afford to spend $O(n^2)$ messages for each epoch change. Roughly, when any replica *wishes to enter epoch* e, it sends an "epoch e" message to each of the $f + 1$ leaders of epoch e. When a correct leader receives $n - f$ epoch e messages, it combines those messages into a message with a single threshold signature, which we call an EC for epoch e. The leader then

broadcasts that EC, signalling that replicas should begin the epoch. When any correct replica first sees an EC for epoch e, they broadcast that EC and begin the epoch. The precise instructions are below.

The instructions for epoch changes.

- When any replica *wishes to enter epoch* e, it sends an "epoch e" message to each of the $f + 1$ leaders of epoch e (potentially including itself). The conditions under which a replica *wishes to enter* an epoch will be described subsequently in the instructions for view changes.
- If a replica is one of the $f + 1$ leaders for epoch e, they enter epoch e if they are presently in a lower epoch[a] and either:
  (a) They receive $n - f$ epoch e messages (from different replicas), or;
  (b) They receive an EC for epoch e.
  In the case that (a) holds, the leader combines the $n - f$ epoch e messages into a message with a single threshold signature, which we call an EC for epoch e, and the leader then broadcasts that EC. In the case that (b) holds, the leader simply broadcasts the EC.
- Each replica which is not a leader for epoch e enters the epoch if they are presently in a lower epoch and if they see an EC for epoch e. Upon entering epoch e, they broadcast that EC for epoch e to all replicas.
- A replica leaves an epoch when it enters a higher epoch.

[a]If at any point any replica finds that the requirements to enter two different views or epochs are simultaneously fulfilled, then the replica enters the higher of the two.

The instructions described above ensure:

($\dagger_0$) If a correct replica i enters epoch e at time t, then all correct replicas will have entered an epoch $e' \geq e$ by time $\max\{t, GST\} + \Delta$, since they will see the EC sent by i by that time.

($\dagger_2$) Each epoch change has message complexity $O(n^2)$, since each replica sends $O(f)$ messages upon wishing to enter epoch e and $O(n)$ messages upon entering epoch e.

2.3 Proceeding through the views within an epoch.

A simple approach to coordinating the views within an epoch would be to require correct replicas to restart their timer upon entering the epoch, and then to begin each view at a designated time according to their timer (so that timers are reset once per epoch). So long as all correct replicas begin the epoch within time Δ of each other, this approach would also (sufficiently) synchronise correct replicas for each view within the epoch. The problem with this simple approach is that it would negate the possibility for optimistic responsiveness – for the protocol to be optimistically responsive, we need correct leaders to be able to produce confirmed blocks at a rate that depends on how fast messages are delivered, rather than at a maximum speed which is determined by Δ. To achieve optimistic responsiveness, we therefore also allow replicas to begin a view $v + 1$ earlier than the designated time, if a confirmed block corresponding to view v is produced.

Roughly, the instructions are as follows. When each replica starts epoch e, they reset their personal timer to 0 and begin by "wishing to enter view 0" (of epoch e). Each replica will wish to enter a view $v > 0$ when either their timer reaches the designated time for that view, or else they see a block corresponding to view $v - 1$ confirmed. When they wish to enter a certain view v, they send a "view v" message to the leader of view v. When the leader sees $n - f$ view v

messages, they form them into a VC for that view, and broadcast that VC to all replicas. Replicas will begin view v when they see a VC for the view (unless they are already in a higher view). The precise instructions, which are described below, refer to the 'message state' of a replica: At any point in the protocol execution, the message state of a replica is the set of all messages received by that replica so far (including messages they have 'sent' to themselves).

The instructions for view changes.

- When each replica starts epoch e, they reset their personal timer to 0 and begin by *wishing to enter view* 0.
- A replica *wishes to enter view* $v > 0$ of epoch e at the first point at which it is in a lower view (for this epoch) and either:
  (a) It sees a block corresponding to view $v-1$ confirmed, or;
  (b) Its personal timer reaches $12v\Delta$ (the idea to have in mind here is that each view will last for at most 12Δ after *GST* when leaders are correct). We'll refer to $12v\Delta$ as the *trigger time* for view v.
- When any replica *wishes to enter view* v, it sends a `view` v message to the leader of view v. It attaches to that message its present message state (which includes the highest QC it has seen).
- The leader of view v of epoch e enters the view if they are in epoch e and both:
  (a) $i = 0$ or they are presently in a lower view, and;
  (b) They receive $n-f$ `view` v messages (from different replicas).
  When this occurs, the leader combines the $n-f$ `view` v messages into a message with a single threshold signature, which we call a VC for view v. The leader considers the highest QC it has seen, which corresponds to a block B' (say). The leader then broadcasts a new proposed block B with parent B', together with the newly produced VC, and attaches to that message its present message state.
- Each replica which is not the leader of view v of epoch e enters the view if they are in epoch e and both:
  (a) $v = 0$ or they are presently in a lower view, and;
  (b) they see a VC for view v.
- Each replica in epoch e wishes to enter epoch $e+1$ the first time that either it sees a confirmed block for view f of epoch e, or else its personal timer reaches $12(f+1)\Delta$. Later, we'll also refer to $12(f+1)\Delta$ as a *trigger time*.

The instructions above suffice to describe how views and epochs are coordinated. It only remains to describe the instructions within each view, which are similar to Hotstuff.

2.4 The instructions within each view.

In addition to the instructions for view and epoch changes above, the remaining instructions are as follows. These instructions are similar to Hotstuff and, in particular, make use of the same 'lock' functionality (we refer the reader to [Buc16, YMR+18] for an explanation of this functionality). All replicas begin in epoch 1 with their 'lock' set to be the genesis block, which corresponds to view 0 of epoch 0. The 'highest QC' *below* a block B is the highest QC that corresponds to any block (including B) in the initial segment of that chain. The instructions for each view are then as follows.

The instructions while in view $v \geq 0$. At any time while in view v, until such a point as it wishes to enter a greater view or epoch (or actually enters a greater view or epoch), each replica carries out the following instructions.

- **Stage 1 Voting.** Each replica considers the first block B they receive for view v produced by the leader (this arrives with the VC for the view). If it extends their lock, or else if the highest QC below the block is as high or higher than they have previously seen (if it's higher they release their lock), they send a stage 1 vote for B to the leader. The stage 1 vote for B is a signed message including the stage, view and epoch number and a hash of the block.
- If the leader receives $n - f$ stage 1 votes for the block they proposed, then they combine these votes into a stage 1 QC for the block, and broadcast this QC.
- **Stage 2 Voting.** The first time a replica sees a stage 1 QC for the block B (specified above) they send a stage 2 vote for B to the leader.
- If the leader receives $n - f$ stage 2 votes for the block they proposed, then they combine these votes into a stage 2 QC for the block, and broadcast this QC.
- **Stage 3 Voting.** The first time a replica sees a stage 2 QC for the block B (as specified above) they set B as their lock and send a stage 3 vote for B to the leader.
- If the leader receives $n - f$ stage 3 votes for the block they proposed, then they combine these votes into a stage 3 QC for the block (which marks the block as confirmed), and broadcast this QC.

3 CONSISTENCY, LIVENESS AND COMPLEXITY

First of all, we deal with consistency, which works exactly as for Hotstuff. For the sake of making our paper as self-contained as possible, we include a proof here.

Lemma 3.1. *The protocol described in Section 2 satisfies consistency.*

Proof. Suppose, towards a contradiction, that two incompatible blocks B and B' are both confirmed. During the description of the protocol in Section 2, we numbered views within each epoch, so that each epoch has views numbered from 0 to f. Just for the duration of this proof, however, it is convenient to suppose that views all have distinct numbers, so that epoch 1 contains views $0, \ldots, f$, then epoch 2 contains views $f + 1, \ldots, 2f + 1$, and so on. Note also, that blocks and votes all contain their epoch, view and stage number. So, we can talk of blocks and votes as corresponding to specific views (and stages of voting). Since any two sets of $n - f$ replicas must have a correct replica in the intersection, and since each correct replica only sends at most one stage 3 vote in each view, it immediately follows that B and B' must correspond to different views. Without loss of generality, suppose that B corresponds to view v and that B' corresponds to a greater view $v' > v$. Let v'' be the least view $> v$ such that a block B'' corresponding to v'' is incompatible with B and receives a stage 1 QC. The fact that B is confirmed means that at least $n - 2f$ correct replicas must set B as their lock while in view v. None of those $n - 2f$ correct replicas which set B as their lock during view v can produce a stage 1 vote for B'' during view v''. Since any two sets of $n - 2f$ correct replicas contain a correct replica in the intersection, this means that B'' cannot receive a stage 1 QC, and gives the required contradiction. □

Next, we prove liveness. As we do so, however, it is convenient to consider also the message complexity:

Lemma 3.2. *The protocol described in Section 2 satisfies liveness, $O(n^2)$ worst-case message complexity and $O(f\Delta)$ time to first confirmation after GST.*

Proof. Let e_0 be the greatest epoch that any correct replica is in at any time < GST, let t_1 be the first time (if there exists such) at which any correct replica enters epoch $e_0 + 1$, and let v_1 be the least view of epoch $e_0 + 1$ which has a correct leader. Our basic aim is to prove that view v_1 of epoch $e_0 + 1$ will produce a confirmed block. To do so, we first want to produce a bound on t_1.

Bounding t_1. Note first that, by $(\dagger_0)$ (of Section 2), all correct replicas will see an EC for epoch e_0 by $GST + \Delta$, and will be in an epoch $\geq e_0$ by this time. Generally, for any correct replica to enter an epoch $e > 1$, at least $n - 2f$ correct replicas have to be in epoch $e - 1$ and wish to enter epoch e. This means that the first epoch $e > e_0$ that any correct replica will enter at any time $\geq GST$ will be $e_0 + 1$. In fact, we can argue that $t_1 \leq GST + 2\Delta + 12(f + 1)\Delta$. To see this, suppose towards a contradiction that no correct replica enters epoch $e_0 + 1$ (and therefore no higher epoch either) by time $GST + 2\Delta + 12(f + 1)\Delta$. Then all correct replicas will wish to enter epoch $e_0 + 1$ by time $GST + \Delta + 12(f + 1)\Delta$, and will send an epoch $e_0 + 1$ message to all leaders of epoch $e_0 + 1$. A correct leader for epoch $e_0 + 1$ will then receive $n - f$ epoch $e_0 + 1$ messages by time $GST + 2\Delta + 12(f + 1)\Delta$, and will enter epoch $e_0 + 1$ at this time.

Proving that view v_1 **produces a confirmed block**. Let t^* be the first time (if it exists) at which a block corresponding to view v_1 of $e_0 + 1$ is confirmed, and let $t_2 = \min\{t^*, t_1 + 12(v_1 + 1)\Delta\}$. Note that, for a correct replica to enter a view or epoch (> 1), at least $n - 2f$ correct replicas must first wish to enter the respective view or epoch. Correct replicas will only wish to enter a given view or epoch when either they see a block corresponding to the previous view[5] confirmed, or else their timer reaches the relevant 'trigger time' (see the instructions for view changes in Section 2). This means that:

- $(\diamond_0)$ No correct replica can enter an epoch $> e_0 + 1$ prior to t_2.
- $(\diamond_1)$ No correct replica can enter any view $v > v_1$ of epoch $e_0 + 1$ prior to t_2.

We claim that:

- $(\diamond_2)$ The number of messages sent by correct replicas (combined) in the time interval $(\text{GST} + \Delta, t_2]$ is $O(n^2)$.
- $(\diamond_3)$ A block corresponding to view v_1 will be confirmed by time t_2.

From $(\diamond_2)$ and $(\diamond_3)$ it follows that $O(n^2)$ messages are sent by all correct replicas combined after $GST + \Delta$ and before the first block proposed by a correct leader is confirmed. It also follows that the time to first confirmation after GST is $O(f\Delta)$, since $t_1 \leq GST + 2\Delta + 12(f + 1)\Delta$ and $t_2 \leq t_1 + 12(v_1 + 1)\Delta$.

Statement $(\diamond_2)$ follows straight from the definition of the protocol – correct replicas only carry out at most one change of epoch (from e_0 to $e_0 + 1$) in the time interval $(\text{GST} + \Delta, t_2]$, and each of the $O(n)$ many views within an epoch has the correct replicas combined sending $O(n)$ many messages.

Next we show $(\diamond_3)$. This now follows essentially just as in Hotstuff. Suppose, towards a contradiction, that no block corresponding to view v_1 is confirmed by time $t_1 + 12(v_1 + 1)\Delta$. By $(\diamond_0)$, it follows that all correct replicas will be in epoch $e_0 + 1$ by time $t_1 + \Delta$. By $(\diamond_0)$ and $(\diamond_1)$, it follows that all correct replicas will be in view v_1 of epoch $e_0 + 1$ by time $t_1 + 3\Delta + 12v_1\Delta$ (and within time Δ of each other). Before broadcasting the VC for view v_1, the leader receives `view` v_1 messages from at least $n - 2f$ correct replicas, and each of these messages includes the highest QC that the replica has seen at the point that the message is sent. When correct replicas wish to enter view v_1 and send a `view` v_1 message, they cease executing instructions for previous views. Now consider the highest lock of any correct replica i before receiving the block B from the leader of view v_1. This lock was set because i saw a stage 2 QC for a block B'. This means that at least $n - 2f$ correct replicas must have produced stage 2 votes for B' (prior to any point at which

[5]Here, it is to be understood that the view previous to view 0 of epoch e is view f of epoch $e - 1$.

they wished to enter view v_1 and stopped executing instructions for views $< v_1$), and so must have seen a stage 1 QC for B' by that time. Since any two sets of $n - 2f$ correct replicas have a correct replica in the intersection, it follows that one of the `view` v_1 messages seen by the leader of v_1 must be from a correct replica who had already seen the stage 1 QC for B'. This means that all correct replicas will then produce stage 1,2 and 3 votes for the block proposed by the leader of view v_1, and this block will be confirmed before time $t_1 + 12(i+1)\Delta$, giving the required contradiction. □

The fact that the protocol is optimistically responsive is clear from the protocol description, so we are left to consider mean message complexity.

Lemma 3.3. *The protocol described in Section 2 has $O(n)$ mean message complexity.*

Proof. Let e_0 be as defined in the proof of Lemma 3.2, i.e. let e_0 be the greatest epoch that any correct replica is in at any time < GST. The proof of Lemma 3.2 actually suffices to establish that all correct leaders for views in epochs $> e_0$ will produce confirmed blocks (just redefine t_1 in the proof to be the first time at which any correct replica enters the relevant epoch and omit the parts of the proof that are now not relevant). The lemma therefore follows, since epochs $> e_0$ will confirm $\Omega(n)$ many blocks on average and will each require correct replicas to send $O(n^2)$ many signatures. □

4 COMMUNICATION COMPLEXITY

4.1 The issue in dealing with communication complexity.

All messages sent by replicas in the protocol of Section 2 would be of length $O(\kappa)$, were it not for the fact that the instructions required some messages to have attached the message state of the replica.[6] When replicas send a `view` v message, for example, they are required to attach their message state. In fact, the instructions differ from the original Hotstuff protocol in that regard – in the Hotstuff protocol, replicas are only required to attach the highest QC they have seen to the message (which is called a NEW-VIEW message in that paper). The question then becomes, if one follows the instructions for Hotstuff, is the information received by the leader actually enough to produce the next block? If the leader is to produce a block which does not repeat requests from earlier in the blockchain, then it isn't enough just to see the highest QC, they also need to know all blocks which are predecessors of the block to which that QC corresponds. One could allow that requests be repeated in the blockchain, and dictate that only the first instance of a request should count (with clients numbering different instances of requests to avoid confusion), but this does not negate the fact that any replica actually needs to know all blocks in the chain to carry out the requests in order. So, there is a subtle difference between the information that is required to build the blockchain and the information that is actually required to execute the requests, in that case. Generally, one has the issue that (even after GST), faulty leaders may confirm f consecutive blocks, with up to f many of the correct replicas not being aware of those blocks. If the protocol is to remain live, then those correct replicas certainly have to be made aware of the confirmed blocks *at some point*. If one has leaders routinely broadcasting f many predecessors each time they broadcast a new block, then each view will have communication complexity $\Theta(\kappa n^2)$. If one has previous blocks only sent upon demand by the relevant replica, then it seems difficult to avoid faulty replicas making demands that cause communication complexity $\Theta(\kappa n^2)$ per view.

[6] Recall from Section 1.1 that the written versions of epoch numbers are $O(\kappa)$.

4.2 Byzantine Agreement.

To deal with this issue, we first observe that such considerations become unproblematic in the case that we only wish to satisfy Byzantine Agreement [LSP82]. Recall that, in the task of reaching Byzantine Agreement, each replica is given an input (which we assume to have length $O(\kappa)$), that all correct replicas are required to terminate with the same output, and that if all correct replicas are given the same input u, then u must be their common output.

To prove Theorem 1.3, we make the following changes to the protocol of Section 2:

- The protocol begins with an extra instruction that has each correct replica broadcast a signed message with its input.
- Blocks no longer contains requests. Any block that has the genesis block as parent must contain signed inputs from $n - f$ processors. The *decision value* corresponding to this set of signed inputs is the most common signed value, with ties broken by least hash. If a correct replica is unable to propose a new block when instructed to do so, it waits for time Δ and then omits the instruction if still unable to proceed.
- Blocks which do not have the genesis block as parent contain only their epoch and view number and a hash of their parent block, together with a decision value, which must be the same as the decision value for their parent block (otherwise the block is not valid and will be ignored by correct replicas).
- When a replica sends a `view` v message, it now attaches only the highest QC it has seen, together with the block corresponding to that QC.
- When a leader proposes a new block B with parent block B' other than the genesis block, it now broadcasts only B, B' and the QC for B', together with the relevant VC. Correct replicas will ignore the new block unless it arrives with B', a QC for B', and unless the decision and hash values match. If B has the genesis block as parent, then the leader broadcasts only the signed block B, together with the relevant VC.
- When a correct replica sees a confirmed block with decision value u, it broadcasts the block together with the stage 3 QC, before terminating with output u.

Verification. Since correct replicas will ignore a new proposed block unless either it has parent the genesis block or else they are able to verify that it has the same decision value as its parent, it follows that no block can receive a QC unless it records the same decision value as its parent or has the genesis block as parent. For any block B with a QC, it follows inductively that all predecessors (other than the genesis block) also receive QCs, and so all record the same decision value. The proof of Lemma 3.1, which was oblivious to the content of blocks beyond the fact that they should satisfy certain structural properties to be valid, suffices to show that incompatible blocks cannot be confirmed, which means that all confirmed blocks must have the same decision value. Since all signed inputs from correct replicas will be delivered by GST+Δ, the proof of Lemma 3.2 suffices to show that a confirmed block will be produced and all correct replicas will terminate within time $O(f\Delta)$ after GST. Since $O(n^2)$ messages will be sent after GST+Δ and prior to the first point at which a correct replica sees a confirmed block, and since all messages are of length $O(\kappa)$, the worst-case communication complexity is $O(\kappa n^2)$. If all correct replicas receive the same input u, then the decision value of any (valid) block with the genesis block as parent must be u. It follows that any block receiving a QC must have decision value u.

4.3 Proving Theorem 1.2.

In the following proof, we assume either that all requests are of length $O(\kappa)$, or else that the length of requests is discounted when counting communication complexity. As described in Section 1.1, we also suppose that blocks contain

a constant bounded number of requests. The basic idea is that we carry out repeated instances of Byzantine Agreement, using a protocol like that described in the proof of Theorem 1.3. We make the following changes to the protocol of Section 2.

(1) We consider *super-epochs*:

- As before, each epoch contains $f + 1$ views. Now, though, we consider also *super-epochs*, which may contain any number of epochs. Each super-epoch will guarantee the confirmation of one new block.
- The genesis block corresponds to view 0 of epoch 0 of super-epoch 0.
- We consider QCs ordered by super-epoch, then by epoch, then by view, and then by stage number within the view.
- Replicas begin in super-epoch 1. Upon entering any super-epoch, replicas immediately wish to enter epoch 1 of that super-epoch.
- Within each super-epoch we rotate leaders as before, and also rotate so that the leaders of epoch 1 of super-epoch s are replicas $s \bmod n, \ldots, s + f \bmod n$.
- Upon first seeing any confirmed block B corresponding to super-epoch s, correct replicas broadcast B, together with a stage 3 QC for B, and then immediately begin super-epoch $s + 1$.

(2) We stipulate how requests should be included in blocks as follows:

- A block whose parent corresponds to a previous super-epoch cannot contain requests included in any predecessor block (otherwise it fails to be valid).
- Blocks whose parent belongs to the same super-epoch must contain the same ordered set of requests as their parent block to be valid.

(3) We change the information attached to messages as follows:

- When a replica sends a `view` v message, it now attaches only the highest QC it has seen, together with the block corresponding to that QC.
- When a leader proposes a new block B with parent block B' other than the genesis block, it now broadcasts only B, B' and the QC for B', together with the relevant VC. Correct replicas will ignore the new block unless it arrives with B', a QC for B', and until they can verify that the conditions on requests included in B (described above) are satisfied. If B' is the genesis block, then the leader just broadcasts B, together with the relevant VC.

(4) The rest of the instructions while within any given view or epoch are exactly as before, except that (just as replicas will not enter any view unless already in the corresponding epoch) replicas will not enter any epoch unless already in the corresponding super-epoch. Replicas will also cease carrying out the instructions for any super-epoch as soon as they leave that super-epoch.

Verification. First of all, we consider the structure of blocks and the relationship with super-epochs.

The requests inside blocks in a chain. Just as in the proof of Theorem 1.3, it follows inductively that for any block B that receives a QC, all predecessors (other than the genesis block) receive QCs, and all predecessors that belong to the same super-epoch record the same ordered set of requests. Again, the proof of Lemma 3.1 suffices to show that incompatible blocks cannot be confirmed. All confirmed blocks corresponding to the same super-epoch must therefore record the same ordered set of requests. When a correct replica first sees a block corresponding to super-epoch s confirmed, it follows inductively that, for each previous super-epoch, at least one correct replica must have seen a block

corresponding to that super-epoch confirmed, and will have broadcast that block together with a stage 3 QC for the block. This means that, when any correct replica first sees a block corresponding to a given super-epoch confirmed, within time Δ it will know at least one confirmed block from each previous super-epoch, and will therefore have the information it needs to execute the requests in order.

Liveness. To establish liveness, we follow almost exactly the same proof as for Lemma 3.2. Let s_0 and e_0 be the greatest super-epoch and epoch that any correct replica is in at any time $<$ GST (so that e_0 is an epoch within s_0). Let t_1 be the first time (if there exists such) at which any correct replica enters either super-epoch s_0+1 (meaning that they have seen a block corresponding to s_0 confirmed) or epoch e_0+1 of s_0. Whenever we refer to epoch e_0+1 (or epoch e_0) in the following, it is always to be assumed that this means epoch e_0+1 (or e_0) of super-epoch s_0. Let v_1 be the least view of epoch e_0+1 which has a correct leader.

We produce a bound on t_1, by much the same argument as in the proof of Lemma 3.2. All correct replicas will see confirmed blocks corresponding to all super-epochs $< s_0$ and an EC for epoch e_0 by $GST+\Delta$, and so will either be in a super-epoch $> s_0$ or else in an epoch $\geq e_0$ of s_0 by this time. The same reasoning as in the proof of Lemma 3.2 then suffices to show that $t_1 \leq GST+2\Delta+12(f+1)\Delta$. Let t^* be the first time (if it exists) at which a correct replica sees a block corresponding to super-epoch s_0 confirmed, and let $t_2=\min\{t^*, t_1+12(v_1+1)\Delta\}$. Note that:

($\diamond_0$) No correct replica can enter an epoch $> e_0+1$ prior to t_2.
($\diamond_1$) No correct replica can enter any view $v > v_1$ of epoch e_0+1 prior to t_2.

We claim that:

($\diamond_2$) The number of bits sent by correct replicas (combined) in the time interval $(\text{GST}+\Delta, t_2]$ is $O(\kappa n^2)$.
($\diamond_3$) A block corresponding to super-epoch s_0 will be confirmed and seen by a correct replica by time t_2.

From ($\diamond_2$) and ($\diamond_3$) it follows that $O(\kappa n^2)$ bits are sent by all correct replicas combined after $GST+\Delta$ and before the first block confirmation. It also follows that the time to first confirmation after GST is $O(f\Delta)$, since $t_1 \leq GST+2\Delta+12(f+1)\Delta$ and $t_2 \leq t_1+12(v_1+1)\Delta$.

As before, statement ($\diamond_2$) follows straight from the definition of the protocol. We are left to establish ($\diamond_3$), which again follows essentially just as in Hotstuff. Suppose, towards a contradiction, that no block corresponding to super-epoch s_0 is confirmed and seen by a correct replica by time $t_1+12(v_1+1)\Delta$. By ($\diamond_0$), it follows that all correct replicas will be in epoch e_0+1 by time $t_1+\Delta$. By ($\diamond_0$) and ($\diamond_1$), it follows that all correct replicas will be in view v_1 of epoch e_0+1 by time $t_1+3\Delta+12v_1\Delta$ (and within time Δ of each other). Before broadcasting the VC for view v_1, the leader receives `view` v_1 messages from at least $n-2f$ correct replicas, and each of these messages includes the highest QC that the replica has seen at the point that the message is sent, together with the corresponding block. When correct replicas wish to enter view v_1 and send a `view` v_1 message, they cease executing instructions for previous views. Now consider the highest lock of any correct replica i before receiving the block B from the leader of view v_1. This lock was set because i saw a stage 2 QC for a block B'. This means that at least $n-2f$ correct replicas must have produced stage 2 votes for B' (prior to any point at which they wished to enter view v_1 and stopped executing instructions for views $< v_1$), and so must have seen a stage 1 QC for B' by that time. Since any two sets of $n-2f$ correct replicas have a correct replica in the intersection, it follows that one of the `view` v_1 messages seen by the leader of v_1 must be from a correct replica who had already seen the stage 1 QC for B'. This means that all correct replicas will then produce stage 1,2 and 3 votes for the block proposed by the leader of view v_1, and this block will be confirmed before time $t_1+12(i+1)\Delta$, giving the required contradiction.

5 DIRECTIONS FOR FUTURE WORK

We close by pointing out some weaknesses in the proofs we have presented here, and asking to what extent those weaknesses are necessary. First of all, consider the protocol described in Section 2, for the proof of Theorem 1.1. A weakness of this protocol is that, after GST, a *single* faulty replica can produce a delay of length $\Omega(f\Delta)$ in the production of confirmed blocks. For example, if the first f leaders of an epoch are all correct and produce f confirmed blocks in close to zero time, then a faulty leader for view f of the epoch can cause a delay of length $\Omega(f\Delta)$ simply by failing to propose any block. The following (slightly imprecise) question naturally arises:

Question 1. *Can one improve the protocol of Section 2 so that it still establishes the claims of Theorem 1.1, while also ensuring that any set of k consecutive faulty leaders can only cause a delay of length $O(k\Delta)$ in the production of confirmed blocks?*

Next, we consider two weakness in the proof of Theorem 1.2. The protocol of Section 2 actually produced something stronger than explicitly claimed in the statement of Theorem 1.1, in that the claims of Theorem 1.1 are all shown to hold if we restrict attention to confirmed blocks *produced by correct leaders*, e.g. at most $O(n^2)$ many messages are sent after GST+Δ and prior to the first confirmation of a block *produced by a correct leader*. In the proof of Theorem 1.2, on the other hand, a block containing a set of requests that are proposed by a correct leader will be produced whenever the first leader of a super-epoch is correct, but super-epochs for which the first leader is faulty may confirm a set of requests that are proposed by a faulty leader (even if it is a correct leader that eventually produces a confirmed block). This means that $O(\kappa n^2)$ many bits need to be sent for each block confirmation after GST, but if f consecutive super-epochs produce blocks confirming a set of requests proposed by a faulty leader, then $O(\kappa n^3)$ many bits are required to be sent before the first confirmation of a block containing requests proposed by a correct leader.

Question 2. *Can one modify the proof of Theorem 1.2 so that the result still holds when we restrict attention to blocks that confirm sets of requests proposed by correct leaders?*

Another obvious weakness of Theorem 1.2, is that we dropped the requirement for $O(\kappa n)$ mean communication complexity:

Question 3. *Consider the partial synchrony model. Does there exist a BFT SMR protocol with optimal resiliency, and worst-case communication complexity $O(\kappa n^2)$, which is also optimistically responsive, with $O(f\Delta)$ time to first confirmation after GST and $O(\kappa n)$ mean communication complexity?*

ACKNOWLEDGEMENTS

The author would like to thank Ittai Abraham, Oded Naor and Kartik Nayak for a number of helpful conversations.

REFERENCES

[ACKL10] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. Prime: Byzantine replication under attack. *IEEE transactions on dependable and secure computing*, 8(4):564–577, 2010.

[BCC+19] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. State machine replication in the libra blockchain. *The Libra Assn., Tech. Rep*, 2019.

[BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.

[BSA14] Alysson Bessani, Joao Sousa, and Eduardo EP Alchieri. State machine replication for the masses with bft-smart. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362. IEEE, 2014.

[Buc16] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, 2016.

[But18] Vitalik Buterin. What is ethereum? *Ethereum Official webpage. Available: http://www. ethdocs. org/en/latest/introduction/what-is-ethereum. html. Accessed*, 14, 2018.

[CKL+09] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. Upright cluster services. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 277–290, 2009.

[CKN21] Shir Cohen, Idit Keidar, and Oded Naor. Byzantine agreement with less communication: Recent advances. *ACM SIGACT News*, 52(1):71–80, 2021.

[CKS05] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

[CL+99] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[CM16] Jing Chen and Silvio Micali. Algorand. *arXiv preprint arXiv:1607.01341*, 2016.

[DLS88] Cynthia Dwork, Nancy A. Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[DR85] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[GAG+19] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: a scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 568–580. IEEE, 2019.

[GKQV10] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. In *Proceedings of the 5th European conference on Computer systems*, pages 363–376, 2010.

[KAD+10] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Transactions on Computer Systems (TOCS)*, 27(4):1–39, 2010.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications*, 1978.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[MR20] Atsuki Momose and Ling Ren. Optimal communication complexity of authenticated byzantine agreement. *arXiv preprint arXiv:2007.13175*, 2020.

[N+08] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system.(2008), 2008.

[NBMS19] Oded Naor, Mathieu Baudet, Dahlia Malkhi, and Alexander Spiegelman. Cogsworth: Byzantine view synchronization. *arXiv preprint arXiv:1909.05204*, 2019.

[NK20] Oded Naor and Idit Keidar. Expected linear round synchronization: The missing link for linear byzantine smr. *arXiv preprint arXiv:2002.07539*, 2020.

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[Sch90] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[Sho00] Victor Shoup. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 207–220. Springer, 2000.

[YMR+18] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus in the lens of blockchain. *arXiv preprint arXiv:1803.05069*, 2018.