Fever: Optimal Responsive View Synchronisation

ANDREW LEWIS-PYE & ITTAI ABRAHAM

View synchronisation is an important component of many modern Byzantine Fault Tolerant State Machine Replication (SMR) systems in the partial synchrony model. Roughly, the efficiency of view synchronisation is measured as the word complexity and latency required for moving from being synchronised in a view of one correct leader to being synchronised in the view of the next correct leader. The efficiency of view synchronisation has emerged as a major bottleneck in the efficiency of SMR systems as a whole. A key question remained open: Do there exist view synchronisation protocols with asymptotically optimal quadratic worst-case word complexity that also obtain linear message complexity and responsiveness when moving between consecutive correct leaders?

We answer this question affirmatively with a new view synchronisation protocol for partial synchrony assuming minimal clock synchronisation, called *Fever*. If n is the number of processors and t is the largest integer $< n/3$, then Fever has resilience t, and in all executions with at most $0 \leq f \leq t$ Byzantine parties and network delays of at most $\delta \leq \Delta$ after GST (where f and δ are unknown), Fever has worst-case word complexity $O(fn + n)$ and worst-case latency $O(\Delta f + \delta)$.

1 INTRODUCTION

Recent years have seen interest in developing protocols for Byzantine Agreement and State Machine Replication (SMR) that work efficiently at scale [10]. In concrete terms, this means looking to minimise the latency and the word complexity per consensus decision as a function of the number of participants n. Most commonly, this analysis takes place in the partial synchrony communication model, first suggested by Dwork, Lynch, and Stockmeyer [12]. The partial synchrony model forces the adversary to choose a point in time called the Global Stabilisation Time (GST) such that any message sent at time $\mathtt{t}$ must arrive by time $\max\{GST, \mathtt{t}\} + \Delta$. While Δ is known, the value of GST is unknown to the protocol. This model forms a practical compromise between the synchronous model (where all message delays are bounded by Δ), which is too optimistic, and the asynchronous model (where message delays are finite but unbounded), which is too pessimistic.

In a recent line of works [1, 9, 14] it has been shown that Byzantine Agreement and SMR can be solved with optimal resilience and with worst-case word complexity $O(n^2)$ after GST. Here, optimal resilience means being able to handle up to t many Byzantine faults [12], where t is the greatest integer less than $n/3$. Given the lower bound of $\Omega(n^2)$ by Dolev and Reischuk [11], this bound on word complexity is tight.

The optimistic case. In practical settings, however, one typically cares not only about the worst-case, but also about the complexity and latency in the optimistic case when the actual (and unknown) number of failures f is less than the given bound t. Indeed, this is one of the principal motivations for considering the partial synchrony model. In the asynchronous model, where randomness is required after the initial cryptographic setup [13], one can already achieve word complexity which is *expected* $O(n^2)$ per consensus decision [2]. In the partially synchrony model, the hope is that one may be able to define protocols which have worst-case complexity (providing cryptographic assumptions hold) which is $O(fn+n)$. Ideally, such protocols should also be *optimistically responsive*. Roughly, this means that the protocol should function at network speed: The protocol should be live during periods when message delay is less than the given bound Δ, but latency should be a function of the actual (unknown) message delay δ. This is important because the actual message delay δ may be much smaller than Δ when the latter value is conservatively set so as ensure liveness under a wide range of network conditions. More formally, we can say that a protocol is optimistically

Author's address: Andrew Lewis-Pye & Ittai Abraham.

responsive if the latency after *GST* is $O(\Delta f + \delta)$ – a precise definition will be given in Section 2. Existing protocols for the partial synchrony model that give optimal resilience and worst-case complexity $O(n^2)$ do not satisfy these requirements. For such protocols [9, 14], the worst-case complexity is $O(n^2)$ but not $O(fn + n)$, while latency is $O(n\Delta)$.

The bottleneck is view synchronisation. Protocols for Byzantine Agreement and SMR typically divide the instructions into *views*, each with a dedicated leader that coordinates the protocol execution during that view. Since Hotstuff [19] shows how to achieve linear complexity within views, the remaining task is to define an efficient protocol that coordinates processors to execute instructions for the same view at the same time as each other. Accordingly, the task of defining efficient protocols for view synchronisation has become a principal focus [4, 9, 14–16], e.g. the protocols mentioned above, that achieve worst-case complexity $O(n^2)$ for Byzantine Agreement in the partial synchrony model, achieve this by defining an appropriate method of view synchronisation.

Clock assumptions. There are actually two scenarios in which view synchronisation becomes a non-trivial task. The first is that processors do not begin the protocol with synchronised clocks or that, even if they do, clocks may experience arbitrary drift prior to *GST*. Even if clocks are initially synchronised and have identical speeds, however, view synchronisation is equally problematic in the case that one requires optimistic responsiveness – the need to produce consensus decisions at 'network speed' means that, during asynchrony prior to *GST*, some correct processors may progress through the protocol instructions arbitrarily faster than others.

The standard approach when defining view synchronisation protocols has been to assume no conditions on initial clock synchronisation and no bound on clock drift prior to *GST* whatsoever, beyond the fact that all correct processors begin the protocol prior to *GST*. In this paper, we take a different approach. We suppose that there may be scenarios in which the participants can form some sort of initial clock synchronisation (e.g. by physically meeting), and where clocks are sufficiently accurate that they will experience bounded drift in any period of asynchrony of realistic length. Atomic clocks can presently be purchased for a few thousand US dollars, for example, and have a typical error of only one second in 100 million years. In fact, we do not assume clocks are perfectly synchronised, but require that clock drift is bounded during periods of asynchrony and define a notion of *minimal initial clock synchronisation* (see Section 2) which can realistically be easily achieved. In this scenario it is then the requirement for optimistic responsiveness which presents a barrier to view synchronisation. Using the assumption of minimal initial clock synchronisation, we are able to define an innovative view synchronisation protocol in which the correct processors send at most $2n$ messages (combined) per view, and which is efficient in both the worst and optimistic cases.

The result. All terms in Theorem 1 will be formally defined in Section 2. Roughly, the worst-case word complexity of a view synchronisation protocol is the maximum number of words (each of maximum length determined by a security parameter) that need to be sent by correct processors during synchrony to synchronise all correct processors on a view with a correct leader. Similarly, the worst-case latency is the maximum time one has to wait during synchrony before all correct processors synchronise on a view with correct leader.

Theorem 1. *Consider the partial synchrony model with maximum delay Δ after GST and with minimal initial clock synchronisation. If t is the largest integer less than $n/3$, there exists a view synchronisation protocol with resilience t, such that for all executions with at most $0 \leq f \leq t$ Byzantine parties and network delays of at most $\delta \leq \Delta$ after GST (where f and δ are unknown):*

(1) *The worst-case word complexity is $O(fn + n)$;*
(2) *The worst-case latency is $O(\Delta f + \delta)$.*

In particular, for $f = 0$ this means $O(n)$ complexity and $O(\delta)$ latency and for $f = t$ this is $O(n^2)$ complexity and $O(\Delta n)$ latency.

Theorem 1 obtains worst-case quadratic communication, constant latency per malicious processor, and responsiveness between consecutive honest leaders. This resolves the main open question raised in Cogsworth[15].

Combined with Hotstuff, Theorem 1 gives an optimally resilient SMR protocol for the partial synchrony model that:

(i) In the worst-case, requires $O(fn + n)$ words to be sent by correct processors after *GST* before confirmation of the first block of transactions after *GST*, and;
(ii) Produces a first confirmed block of transactions after *GST* within time $O(\Delta f + \delta)$ of *GST*.

Since *GST* is unknown to the protocol, note that similar bounds then hold for the word complexity and latency between honestly produced confirmed blocks after *GST*.

The key conceptual contribution of this work is to show that careful use of *local clocks* can improve view synchronisation. Specifically, under minimal initial clock synchronisation, it is possible to maintain a *weak form of clock synchronisation* even under complete network asynchrony (before *GST*). On the one hand, this weak form of synchronisation guarantees that, despite network asynchrony, the correct processor whose local clock is most advanced is advanced by a bounded amount relative to at least t other correct processors. On the other hand, this weak form of clock synchronisation enables processors to *algorithmically move local clocks forward* to obtain responsiveness, which is captured by obtaining $O(\delta)$ latency when $f = 0$.

1.1 Related work

Tendermint [7] showed how to use constant size messages for view-change. Casper FFG [8] extended this approach to allow pipelining. Hotstuff [19] extended these to define an SMR protocol achieving responsivenes and word complexity $O(n)$ within views, but did not rigorously establish an efficient technique for view synchronisation. In response to this, a number of papers have described view synchronisation protocols with different trade-offs.

Cogsworth [15] and Naor-Keidar [16] consider a setup in which leaders are chosen according to successive random permutations of the set of processors. They consider a static and *oblivious* adversary, who must choose *GST* without knowledge of the sequence of randomly chosen leaders, and which must also choose processors to corrupt at the start of the protocol execution without this knowledge. While we do not need to make use of any randomness (beyond that required for the initial cryptographic setup) to establish Theorem 1, we also consider such a setup for the purpose of apples-to-apples comparisons in Table 1 (in the 'Expected Latency' and 'Expected Complexity' columns). Both Cogsworth and Naor-Keidar achieve expected latency $O(\Delta)$ for such a static adversary, but this bound increases to $O(f^2\Delta + \delta)$ in the case that the adversary is adaptive, i.e. if the adversary can choose which processors to corrupt as the execution progresses (and with knowledge as to their choice of *GST*). The principal improvement of Naor-Keidar over Cogsworth is to decrease the expected complexity from $O(n^2)$ in the case of a static and oblivious adversary to $O(n)$. The expected complexity for Cogsworth becomes $O(fn^2 + n)$ in the case of an adaptive adversary, and the worst-case complexity is also $O(fn^2 + n)$. The expected complexity for Naor-Keidar becomes $O(f^2n + n)$ in the case of an adaptive adversary, and the worst-case complexity is also $O(f^2n + n)$. For a more detailed discussion of Cogsworth and Naor-Keidar, see the Appendix.

While the published version of Hotstuff [19] did not describe any efficient method for view synchronisation, the original version (posted on the arXiv [1]) did roughly outline an approach to meeting the $O(n^2)$ worst-case complexity bound of Dolev-Reischuk. This approach was made precise and rigorously proved in [14] and [9]. These papers described view synchronisation protocols

Protocol	Expected Latency	Worst-case Latency	Expected Complexity	Worst-case Complexity	Minimal Initial Clock Sync
Cogsworth	static adv: $O(\Delta)$ adaptive adv: $O(f^2\Delta+\delta)$	$O(f^2\Delta+\delta)$	static adv: $O(n^2)$ adaptive adv: $O(fn^2+n)$	$O(fn^2+n)$	Not needed
Naor-Keidar	static adv: $O(\Delta)$ adaptive adv: $O(f^2\Delta+\delta)$	$O(f^2\Delta+\delta)$	static adv: $O(n)$ adaptive adv: $O(f^2n+n)$	$O(f^2n+n)$	Not needed
Lewis-Pye	$O(n\Delta)$	$O(n\Delta)$	$O(n^2)$	$O(n^2)$	Not needed
Raresync	$O(n\Delta)$	$O(n\Delta)$	$O(n^2)$	$O(n^2)$	Not needed
Fever (this paper)	static adv: $O(\Delta)$ adaptive adv: $O(f\Delta+\delta)$	$O(f\Delta+\delta)$	static adv: $O(n)$ adaptive adv: $O(fn+n)$	$O(fn+n)$	Needed

Table 1. View Synchroniser Comparisons

In Table 1, we assume the *bound* t on the number of Byzantine processors is the largest integer less than $n/3$, so that $t = \Theta(n)$, while $0 \leq f \leq t$ is the *actual* number of Byzantine processors. 'Complexity' means 'word complexity'. Both latency and word complexity are defined in Section 2, as is the 'minimal initial clock synchronisation' condition. We only distinguish explicitly between a static and adaptive adversary when this changes the corresponding bound.

which we will refer to as 'Lewis-Pye' and 'Raresync' respectively. A disadvantage of these protocols over Fever (which we describe in this paper) is that they both have worst-case latency $O(n\Delta)$, as opposed to $O(f\Delta+\delta)$ for Fever, and worst-case complexity $O(n^2)$, as opposed to $O(fn+n)$ for Fever.

Thus far, we have focused on view synchronisation protocols for the partial synchrony model. It should be emphasized that the stronger efficiency bounds for Fever are achieved via a novel view synchronisation protocol combined with stricter assumptions on initial clock synchronisation (as made precise in Section 2). The results are therefore only of interest in the case that one considers these stricter assumptions reasonable under some scenarios of interest.

It is well-known that protocols in the asynchronous model can achieve *expected* complexity $O(n^2)$ per consensus decision (e.g. see [2]). The trade-off when compared with the protocol we present here (when combined with Hotstuff) is that such asynchronous protocols do not require synchronous intervals to be live, but still have complexity $O(n^2)$ in the case that $f = 0$.

In [18], Spiegelman describes a protocol for Byzantine Agreement which is designed to operate efficiently under both synchronous and asynchronous conditions. The protocol achieves $O(n^2)$ complexity in asynchrony and $O(fn+n)$ in synchrony. The principal trade-off here is that this protocol only achieves $O(n^2)$ complexity in the partial synchrony model, even when $f = 0$. Again, the obvious advantage of Spiegelman's protocol is that it is live in the asynchronous model.

A recent sequence of papers by Bravo, Chockler, and Gotsman [4–6] describe a modular framework for the analysis of view-based SMR protocols. The aim of those papers is complementary to and different than our aim here. While those authors describe a general framework and are less concerned with establishing optimal results in terms of complexity and latency (such as those described here), our aim is to describe a specific view synchronisation protocol achieving state-of-the-art efficiency. An advantage of the approach described by those authors is that it allows for a PBFT-style approach to view change, whereby a single leader may persist until correct processors request a change in leader. By contrast, the view synchronisation protocol we describe here has processors automatically pass through views with different leaders. So, this is at least one sense in which the approach described by Bravo et. al. is more general than what we describe here.

2 THE SETUP

We consider a set $\Pi = \{p_0, \ldots, p_{n-1}\}$ of n processors, and let t be the largest integer less than $n/3$. Each processor p_i is told i as part of its input. For the proof of Theorem 1, we assume an adaptive adversary that is able to choose at most t processors to corrupt as the execution progresses. A processor that is corrupted by the adversary at any point in the execution is referred to as *Byzantine*, and may behave arbitrarily once corrupted. Processors that are not Byzantine are *correct*. We let f denote the actual number of Byzantine processors.

Cryptographic assumptions. Our cryptographic assumptions are standard for papers on this topic. Processors communicate by point-to-point authenticated channels. We use a cryptographic signature scheme, a public key infrastructure (PKI) to validate signatures, and a threshold signature scheme [3, 17]. The threshold signature scheme is used to create a compact signature of m-of-n processors, as in other consensus protocols [19]. In this paper, either $m = t + 1$ or $m = n - t$. The size of a threshold signature is $O(\kappa)$, where κ is a security parameter, and does not depend on m or n. We assume a computationally bounded adversary. Following a common standard in distributed computing and for simplicity of presentation (to avoid the analysis of negligible error probabilities), we assume these cryptographic schemes are perfect, i.e. we restrict attention to executions in which the adversary is unable to break these cryptographic schemes.

Communication. As noted above, processors communicate using point-to-point authenticated channels. We consider the standard partial synchrony model, whereby a message sent at time $\mathtt{t}$ must arrive by time $\max\{GST, \mathtt{t}\} + \Delta$. While Δ is known, the value of GST is unknown to the protocol. The adversary chooses GST and also message delivery times, subject to the constraints already defined.

According to the definition above, messages sent prior to GST may be significantly delayed, but are not lost. We only use the assumption that messages are not lost, however, when analysing the word complexity of reaching the *first* synchronisation after GST. Our view synchronisation protocol works without this assumption, and the assumption could be dropped if one was to consider complexity measures which are less strict than that we consider here, such as that in [15].

Minimal Initial Clock Synchronisation. There are two assumptions on clock synchronisation that are standard: (i) All clocks are synchronised at the start of the protocol execution and have zero drift (i.e. identical speeds), even prior to GST, or; (ii) Clocks may have arbitrary drift prior to GST but have zero (or, at least, bounded) drift after GST. We consider a middle ground. To define the condition, let $c(p)$ denote the value of processor p's clock. At any point $\mathtt{t}$ in an execution, let $T(\mathtt{t}) := \{c(p) : p \text{ is correct}\}$. In particular, this means that $T(0)$ is the set of all clock values for correct processors at the start of the protocol execution. Our required condition regarding initial clock synchronisation is that, for some known bound Γ:

($\dagger_{\Gamma,0}$) For any $\mathtt{c} \in T(0)$:

$$|\{\mathtt{c}' \in T(0) : \mathtt{c}' \geq \mathtt{c} - \Gamma\}| \geq t + 1.$$

Recall that t is the bound on the number of Byzantine processors. So, the condition above says that, for each correct processor p, there are at least t other correct processors whose clocks (may be arbitrarily ahead of p's clock but) are at most Γ behind p's clock. Specifically, this is really a condition on the most advanced clock of an correct processor. If p's clock is the most advanced amongst correct processors, then we require that there are at least t correct processors whose clocks are at most Γ behind p's clock. Another way of looking at this is that all correct processors begin the protocol execution with their local clock set to 0, and that if p is the first correct processor to

begin the protocol execution (while other clocks may still be negative, so that those processors are still waiting to start), then at least t other correct processors begin the protocol execution within time Γ. Note that this condition does not place any bound on the maximum difference between the clocks of correct processors.

For the sake of simplicity, we will also initially assume that all correct processors have identical clock speeds. Then, in Section 5, we will consider realistic relaxations of this condition that suffice to give our results.

The underlying protocol. We suppose view synchronisation is required for some underlying protocol (such as Hotstuff) with the following properties:

- **Views**. Instructions are divided into views. Each view v has a designated *leader*, denoted $\mathtt{lead}(v)$. For some parameter $k \geq 3$ (which can be chosen to suit the protocol designer's needs), we suppose views are grouped into sets of k, so that the leader[1] for view v is processor p_i where $i := \lfloor v/k \rfloor \bmod n$. If $v \bmod k = 0$, then v is called 'initial'.
- **Quorum certificates**. The successful completion of a view is marked by all processors receiving a *Quorum Certificate* (QC) for view v. The QC is a threshold signature of length $O(\kappa)$ (for the security parameter κ that determines the length of signatures and hash values) combining $n - t$ signatures from different processors testifying that they have completed the instructions for the view. In a chained implementation of Hotstuff, for example, the leader will propose a block, processors will send votes for the block to the leader, who will then combine those votes into a QC and send this to all processors. Alternatively, one could consider a (non-chained) implementation of Hotstuff, in which the relevant QC corresponds to a successful third round of voting. Note that the production of QCs is not a restrictive assumption, since if it is not satisfied one can easily amend the instructions of the protocol so that it is.
- **Sufficient time for view completion**. We suppose there exists some known $x \geq 2$ such that if $\mathtt{lead}(v)$ is correct, if (the global time) $\mathtt{t} \geq$ GST, and if at least $n - t$ correct processors are in view v from time $\mathtt{t}$ until either they receives a QC for view v or until $\mathtt{t} + x\delta$, then all correct processors will receive a QC for view v by time $\mathtt{t} + x\delta$, so long as all messages sent by correct processors while in view v are received within time $\delta \leq \Delta$. For the sake of simplicity, we assume Γ from the definition of 'minimal initial clock synchronisation' is equal to $x\Delta-$ if these values differ then one can just take the maximum of the two values.

The view synchronisation task. For Γ as above, we must ensure:

(1) If a correct processor is in view v at time $\mathtt{t}$ and in view v' at $\mathtt{t}' \geq \mathtt{t}$, then $v' \geq v$.
(2) There exists some correct $\mathtt{lead}(v)$ and $\mathtt{t} \geq GST$ such that each correct processor is in view v from time $\mathtt{t}$ until either it receives a QC for view v or until $\mathtt{t} + \Gamma$.

Condition (1) above is required by standard view-based SMR protocols to ensure consistency. Since GST is unknown to the protocol, condition (2) suffices to ensure the successful completion of infinitely many views with correct leaders. By a *view synchronisation protocol*, we mean a protocol which determines when processors enters views and which satisfies conditions (1) and (2) above.

Complexity measures. Our proofs are quite robust to the precise notions of latency and word complexity considered, and will hold for any of the definitions used in previous papers on the topic such as [4, 15, 16]. For the sake of concreteness, we fix complexity measures which are as strict as

[1]These assumptions are made for the purpose of proving Theorem 1. In verifying the bounds given in Table 1, we will also consider the possibility of random leader selection.

possible, and note that if we were to adopt the more relaxed measures used in [15], for example, then we could weaken the requirement that messages sent before *GST* are not lost.

By a 'word', we mean a message of length $O(\kappa)$, where κ is the security parameter determining the length of signatures and hash values. We make the following definitions. Let $\mathtt{t}^*$ be the least time $> GST$ at which the underlying protocol has some correct $\mathtt{lead}(v)$ produce a QC for view v. The worst-case word complexity is the maximum number of words sent by correct processors between time $GST + \Delta$ and $\mathtt{t}^*$. The worst-case latency is the maximum possible value of $\mathtt{t}^* - GST$.

Defining optimistic responsiveness. We do not need to define optimistic responsiveness to establish Theorem 1. For the sake of concreteness, however, we can define our view synchronisation protocol to be optimistically responsive if the worst-case latency is $O(f\Delta + \delta)$, where f is the (unknown) number of Byzantine processors and $\delta \leq \Delta$ is the actual (unknown) bound on message delay after *GST*.

3 THE PROTOCOL

Recall that views are grouped into sets of k, so that the leader for view v is processor $\lfloor v/k \rfloor \bmod n$. If $v \bmod k = 0$, then v is called 'initial'. To synchronise processors, we have a predetermined 'clock-time' corresponding to each view: The clock-time corresponding to view v is $\mathtt{c}_v := \Gamma v$.

The rough idea is that, at certain points in the the execution (and to satisfy optimistic responsiveness), we have processors instantaneously forward their clock to some clock-time $\mathtt{c}_v$ and enter view v. We do this in such a way to ensure that, if p is the correct processor whose local clock is most advanced, then there are always at least t other correct processors whose local clocks are at most Γ behind p's clock. This will suffice to ensure correct leaders are able to synchronise all correct processors after *GST*.

The instructions are defined simply as follows:

When processors enter views. Recall that, at any point in the execution, $c(p)$ is the value of processor p's clock. If v is initial, then p enters view v when $c(p) = \mathtt{c}_v$. If v is not initial, then p enters view v if it is presently in a view $< v$ and it receives a QC (formed by the underlying protocol) for view $v - 1$.

View Certificates. When a correct processor p enters a view v which is initial, it sends a $\mathtt{view}\ v$ message to $\mathtt{lead}(v)$. This message is just the value v signed by p. Once $\mathtt{lead}(v)$ receives $t+1$ $\mathtt{view}\ v$ messages from distinct processors, it combines these into a single threshold signature, which is a view certificate (VC) for view v, and sends this VC to all processors.[2]

When processors forward clocks. At any point in the execution, if a correct processor p receives a QC for view $v - 1$ (formed by the underlying protocol) or a VC for view v, and if $c(p) < \mathtt{c}_v$, then p instantaneously forwards their clock to $\mathtt{c}_v$.

Pseudocode for the protocol is given in Algorithm 1.

The informal intuition behind the protocol: The condition for minimal initial clock synchronisation requires that, at the start of the protocol execution, and if p is the correct processor whose local clock is most advanced, there are at least t other correct processors whose local clocks are at most Γ behind p's clock. The protocol above is specified to ensure this condition remains true throughout the execution – see Section 4 for a simple proof. This will be easily seen by checking that the condition can never be violated by the forwarding of clocks. Now suppose that $\mathtt{lead}(v)$ is correct and that a correct processor, p say, is the first to enter view v after *GST* at time $\mathtt{t}$. Our

[2]It is convenient throughout to assume that when a leader sends a message to all processors, this includes itself.

condition on local clocks, described above, means that t other correct processors will also enter view v within a short time. Since $\mathtt{lead}(v)$ only requires $t+1$ signatures to form a VC for view v, all correct processors will then receive a VC for view v within a short time. The underlying protocol will then have $\mathtt{lead}(v)$ put together a QC for view v.

Algorithm 1 The instructions for processor p.

Local variables
$c(p)$, intially 0 ▷ This is the value of p's clock
v, initially 0 ▷ This is the present view of p.

Global parameters
n ▷ Number of processors
t ▷ Largest integer $< n/3$
$k := 3$ ▷ Can take larger values.
$\mathtt{c}_{v'} := v'\Gamma$, $v' \in \mathbb{N}_{\geq 0}$ ▷ Defines clock times
$\mathtt{lead}(v') := p_i$ for $i = \lfloor v'/k \rfloor \bmod n$ and $v' \in \mathbb{N}_{\geq 0}$ ▷ Specifies leaders

Upon $c(p) == \mathtt{c}_{v'}$ for v' initial
    Set $v := v'$
    Send a `view` v message to $\mathtt{lead}(v)$

Upon first seeing a QC for view $v' \geq v$
    Set $v := v' + 1$
    If $c(p) < \mathtt{c}_{v'+1}$ set $c(p) := \mathtt{c}_{v'+1}$

Upon first seeing a VC for initial view $v' > v$
    Set $v := v'$
    If $c(p) < \mathtt{c}_{v'}$ set $c(p) := \mathtt{c}_{v'}$

If $p == \mathtt{lead}(v')$ for $v' \geq v$ **then**
    Upon first seeing $\mathtt{view}(v')$ messages from $t+1$ distinct processors
        Form a VC for view v' and send to all processors

4 THE PROOFS

It is immediate from the instructions that if a correct processor enters a view v then it cannot subsequently enter any lower view.

Recall that, at any point $\mathtt{t}$ in an execution, $T(\mathtt{t}) := \{c(p) : p \text{ is correct}\}$. Our condition for 'minimal initial clock synchronisation' required that a certain condition ($\dagger_{\Gamma,0}$) holds at the start of the protocol execution. This condition requires that if p is the correct processor whose local clock is most advanced, then at least t other correct processors have clocks that are at most Γ behind p's clock. The key to the proof is to show that an analogous condition then holds at all times.

Lemma 4.1. *For all* $\mathtt{t}$ *the following condition holds:*

($\dagger_{\Gamma,\mathtt{t}}$) *For any* $\mathtt{c} \in T(\mathtt{t})$*:*

$$|\{\mathtt{c}' \in T(\mathtt{t}) : \mathtt{c}' \geq \mathtt{c} - \Gamma\}| \geq t + 1.$$

Before proving Lemma 4.1, we note that the lemma does *not* place any bound on the maximum difference between the local clocks of correct processors. In fact, even if all clocks are initially perfectly synchronised, the local clocks of two correct processors can move arbitrarily far apart prior to *GST*. Nevertheless, the fact that $(\dagger_{\Gamma,\mathtt{t}})$ holds for all $\mathtt{t}$ will suffice to establish Theorem 1.

Proof. (Lemma 4.1) Since the local clocks of correct processors only ever move forward, it follows that at any point in an execution, if a correct processor p has already contributed to a QC or a VC for view v, then $c(p) \geq \mathtt{c}_v$. To prove that $(\dagger_{\Gamma,\mathtt{t}})$ holds for all $\mathtt{t}$, suppose towards a contradiction that there is a first point of the execution, $\mathtt{t}$ say, for which there exists some correct processor p such that $|\{\mathtt{c} \in T(\mathtt{t}) : \mathtt{c} \geq c(p) - \Gamma\}| < t + 1$. Then p must forward its clock at $\mathtt{t}$. There are two possibilities:

(1) p forwards its clock because it receives a VC for some view v with $\mathtt{c}_v > c(p)$. In this case, there must exist at least one correct processor $p' \neq p$ which contributed to the VC for view v. By the choice of $\mathtt{t}$, when p' contributed to the VC at $\mathtt{t}' \leq \mathtt{t}$ we had $|\{\mathtt{c} \in T(\mathtt{t}') : \mathtt{c} \geq c(p') - \Gamma\}| \geq t + 1$. Since $c(p') \geq \mathtt{c}_v$ when it contributed to the VC, and since $c(p) = \mathtt{c}_v$ at $\mathtt{t}$, at $\mathtt{t}$ we have that $|\{\mathtt{c} \in T(\mathtt{t}) : \mathtt{c} \geq c(p) - \Gamma\}| \geq t + 1$ also, which gives the required contradiction.
(2) p forwards its clock because it sees a QC. In this case, at least $t + 1$ correct processors must have contributed to the QC, which directly gives the required contradiction.

□

Lemma 4.2. *If v is initial and $\mathtt{t}$ is the first time any correct processor enters a view $\geq v$:*

(i) *A correct processor enters view v at $\mathtt{t}$;*
(ii) *No correct processor enters any view $v' > v$ at $\mathtt{t}$, and;*
(iii) *$c(p) \leq \mathtt{c}_v$ for all correct p at $\mathtt{t}$.*

Proof. Consider the first time any correct processor p enters a view $v' \geq v$. It cannot be because p sees a VC for view v', because some correct processor must then have contributed to that VC and already have been in view v'. It cannot be because p sees a QC for view $v' - 1 > v - 1$, because $t + 1$ correct processors must have already contributed to that QC. It follows that the first view $v' \geq v$ entered by any correct processor is v. When the first correct processor p enters view v we have $c(p) = \mathtt{c}_v$ (either simply because it reaches this value, or else because p sees a QC for view $v - 1$), and that $c(p') \leq \mathtt{c}_v$ for all correct p' at this point. □

Definition 4.3. Let $\mathtt{t}(v)$ be the first time at which a correct processor enters view v.

Since correct processors enter an unbounded number of views, it follows from Lemma 4.2 that if v is initial then $\mathtt{t}(v) \downarrow$ and $\mathtt{t}(v') > \mathtt{t}(v)$ whenever $v' > v$ and $\mathtt{t}(v') \downarrow$.

Note also that if v is initial then, for $j \in (0, k)$, a QC for view $v + j$ cannot be formed prior to the formation of a QC for view $v + j - 1$. This follows because (since v is initial, and for j in the given range) correct processors do not enter view $v + j$ without seeing a QC for view $v + j - 1$. The next lemma will be used to show that correct processors spend a sufficiently long time in each view that a correct leader after *GST* will be able to produce QCs.

Lemma 4.4. *Suppose v is initial. For each $j \in [0, k)$, let $\mathtt{s}_j$ be the first time (if there exists such) at which a correct processor sees a QC for view $v + j$. The first time at which any correct processor enters view $v + k$ is the minimum amongst the values $\{\mathtt{t}(v) + k\Gamma\} \cup \{\mathtt{s}_j + (k - 1 - j)\Gamma : \mathtt{s}_j \downarrow\}$.*

Proof. By Lemma 4.2, some correct processor p enters v at $\mathtt{t}(v)$, and all correct processors p' have $c(p') \leq \mathtt{c}_v$ at this point. As we reasoned in the proof of Lemma 4.2, it cannot be the case that the first time any correct processor enters a view $v' \geq v + k$ it is because it sees a VC for view v' or

a QC for $v' - 1 \geq v + k$. It follows that the first time a correct processor p enters view $v + k$ it is because its local clock has reached c_{v+k}. This happens either because p saw a QC for view $v + j$ ($j \in [0, k)$) and then time $(k - 1 - j)\Gamma$ passed (meaning zero time if $j = k - 1$), or else because p was the first correct processor to enter view v and time $k\Gamma$ passed since that point. □

With Lemmas 4.1, 4.2 and 4.4 in place, the basic intuition behind the idea that a correct leader will produce a QC after *GST* is clear. Let $\mathtt{lead}(v)$ be correct and such that no correct processor enters view v prior to *GST*. From Lemma 4.2, it follows that no correct processor enters any view $v' > v$ prior to $\mathsf{t}(v)$. By Lemma 4.1, at least $t + 1$ correct processors will have entered view v within time Γ – by Lemma 4.4, no correct processor will be in any view $> v$ prior to the first of $\mathsf{t}(v) + k\Gamma$ or else the formation of a QC for view v. The correct processor $\mathtt{lead}(v)$ will then form a VC for view v, and all correct processors will be in view v by time $\mathsf{t} + \Gamma + 2\Delta$ unless a QC for view v has already been formed by this point. This means that all processors will receive a QC for view v by time $\mathsf{t} + 2\Gamma + 2\Delta$. Since $\Gamma \geq 2\Delta$ and $k \geq 3$, this suffices.

Now let us see the details. In the below, we prove more than the fact that a correct $\mathtt{lead}(v)$ will produce a QC for one of the views in $[v, v + k)$. We show that $\mathtt{lead}(v)$ will produce QCs for multiple successive views if k is large enough, since this will be useful in some implementations (such as chained implementations of Hotstuff etc).

Lemma 4.5. *Suppose v is initial, $\mathtt{lead}(v)$ is correct, and that $\mathsf{t}(v) \geq GST$. Then correct processors will see QCs for all views in $[v, v + k - 2)$ before entering view $v + k$.*

Proof. By Lemma 4.2, no correct processor has entered any view $v' > v$ at $\mathsf{t}(v)$. By Lemma 4.1, $(\dagger_{\Gamma,\mathsf{t}(v)})$ is satisfied, which means at least $t + 1$ $\mathtt{view}\ v$ messages will have been sent to $\mathtt{lead}(v)$ by $\mathsf{t}(v) + \Gamma$ – by Lemma 4.4, no correct processor will be in any view $> v$ prior to the point at which these $t + 1$ $\mathtt{view}\ v$ messages have been sent to $\mathtt{lead}(v)$. Then $\mathtt{lead}(v)$ will have sent out a VC for view v by $\mathsf{t}(v) + \Gamma + \Delta$, which will be received by all correct processors by time $\mathsf{t}(v) + \Gamma + 2\Delta$. It then follows from Lemma 4.4, and since $\Gamma \geq 2\Delta$, that a QC for each view $j \in [0, v + k - 2)$ will be seen by all correct processors by time $\mathsf{t}(v) + \Gamma + 2\Delta + (j + 1)\Gamma$, prior to any point at which a correct processor enters view $v + k$. □

Lemma 4.6. *The worst-case word complexity is $O(fn + n)$ and the worst-case latency is $O(\Delta f + \delta)$.*

Proof. We deal with the word complexity first. Let p be the correct processor whose clock is most advanced at *GST* (breaking ties arbitrarily). Suppose p is in view v at *GST*. Let v_0 be the greatest initial view $< v$ such that $\mathtt{lead}(v_0) \neq \mathtt{lead}(v)$ and $\mathtt{lead}(v_0)$ is correct. Let v_1 be the least initial view $> v$ such that $\mathtt{lead}(v_1)$ is correct. Since no correct processor will enter any view $> v_0$ prior to the least of $\mathsf{t}(v_0) + k\Gamma$ or the first time at which a correct processor sees a QC for view v, and since $(\dagger_{\Gamma,\mathsf{t}(v_0)})$ is satisfied, $\mathtt{lead}(v_0)$ must have sent a VC for view v_0 to all processors prior to *GST*. All correct processors will therefore be in at least view v_0 by $GST + \Delta$. Lemma 4.5 shows that all correct processors will see a QC for view v_1 before entering view $v_1 + k$. Let f^* be the number of Byzantine leaders for initial views in the interval (v_0, v_1). Correct processors will send a maximum of $2(f^* + 3)n$ many $\mathtt{view}$ messages (combined) between $GST + \Delta$ and the time at which $\mathtt{lead}(v_1)$ produces a QC for view v_1. If the underlying protocol has correct processors send $O(n)$ messages per view (e.g. Hotstuff), then the underlying protocol will also have correct processors send $O((f^* + 3)n)$ messages during this interval. So the worst-case word complexity is $O(fn + n)$, as required.

Next, we consider the worst-case latency. If $f > 0$, then it suffices to observe that $\mathtt{lead}(v_1)$ will produce a QC for view v_1 by time $GST + k(f^* + 3)\Gamma$. So suppose $f = 0$ and consider the number d of correct processors in view v at *GST*. If $d \geq t + 1$, then $\mathtt{lead}(v)$ will produce a QC for view v

within time $O(\delta)$ according to our assumptions on the underlying protocol, unless at least $t + 1$ processors enter view $v + k$ before this occurs. In the latter case, $\texttt{lead}(v + k)$ will produce a QC for view $v + k$ within time $O(\delta)$. If $d < t + 1$, then (the previous leader) $\texttt{lead}(v_0)$ will produce a QC for view v within time $O(\delta)$, unless at least $t + 1$ processors enter view v before this occurs. In the latter case, $\texttt{lead}(v)$ will produce a QC for view v within time $O(\delta)$. □

Lemma 4.6 completes the proof of Theorem 1. We finish this section by justifying the entries of Table 1, which state that the expected latency is $O(\Delta)$ and the expected word complexity is $O(n)$ with a static adversary according to the model of [15]. According to this model, leaders are given by successive random permutations of the set of all processors. The adversary is static and *oblivious*, which means that they must choose which processors to corrupt at the start of the protocol execution without knowledge as to the random sequence of leaders, and must also choose GST without this knowledge. In this case, the expected value f^* from the proof of Lemma 4.6 is $O(1)$, which means we get expected latency $O(\Delta)$ and expected word complexity $O(n)$, as required.

5 TYING UP LOOSE ENDS

5.1 A note on optimistic responsiveness and Byzantine leaders

In the proof of Lemma 4.6, it was only actually the number of Byzantine *leaders* before the first correct leader after GST that mattered (rather than the total number of Byzantine parties) in establishing that the worst-case word complexity is $O(fn + n)$. The proof that the worst-case latency is $O(f\Delta + \delta)$ was somewhat more subtle, and considered the total number of Byzantine processors. Roughly, the difficulty occurs when none of the relevant leaders are Byzantine, but a correct processor has just entered initial view v at GST, while all other correct processors are still in previous views. In this scenario, the previous leader cannot produce a QC (if t parties are Byzantine while not in their role as leader). Meanwhile, the leader for view v has to wait time $\Gamma + \delta$ before producing a VC. As we argued in the proof of Lemma 4.6, this is not an issue if we let f count the total number of Byzantine parties.

Once a first correct leader produces a QC after GST, however, this subtlety is no longer relevant. Let $\texttt{lead}(v)$ be correct and such that $\texttt{t}(v) \geq GST$. Let v' be the least view with correct leader $> v$ and suppose that the number of initial views with Byzantine leader in the interval (v, v') is f^*. Then the proof of Lemma 4.6 is easily modified to show that correct processors send $O(f^*n + n)$ many words between the times at which $\texttt{lead}(v)$ and $\texttt{lead}(v')$ produce QCs, and that the time between these events is $O(f^*\Delta + \delta)$.

5.2 Revisiting the assumptions regarding clock synchronisation

In Section 2 we assumed that all processors have identical clock speeds. We now consider to what extent we can relax this condition. As we do so, we also consider how realistic are the required assumptions in the context of reasonable bounds on network delays, the length of periods of asynchrony etc., and in a context where atomic clocks are available for use by processors. Recall that atomic clocks can reasonably be assumed to have error less than 1 second every 100 million years.[3]

In the partial synchrony model it is only for the sake of technical convenience that we consider a single period of asynchrony and then a single period of synchrony after GST. In reality, we are interested in contexts where network conditions oscillate between synchrony and asynchrony. We require our protocols to maintain consistency during periods of asynchrony, and to be live during periods of synchrony. To ensure that our analysis extends to such a scenario, let us therefore

[3]See, for example, https://en.wikipedia.org/wiki/Atomic_clock

consider our requirements as the network oscillates between periods of sychrony and asynchrony in this fashion.

Fix $k := 3$. A similar analysis will also apply for larger values of k. Let us say an open interval $(\mathtt{t}, \mathtt{t}')$ is synchronous if every message sent in this interval arrives within time Δ. Let $\ell := 3(t+3)\Gamma$, where t is the bound on the number of Byzantine processors. If $(\dagger_{\Gamma,\mathtt{t}'})$ holds for all $\mathtt{t}' \in I := (\mathtt{t}, \mathtt{t}+\ell)$ and if I is synchronous, the proofs of Section 4 established that some correct leader will produce a QC during interval I and send this to all processors. With this in mind, we inductively define a sequence of times $(\mathtt{t}_i)_{i\geq 0}$ as follows:

- Let $\mathtt{t}_0$ be the least that $(\mathtt{t}_0, \mathtt{t}_0 + \ell)$ is synchronous.
- Given $\mathtt{t}_i$, let $\mathtt{t}_{i+1}$ be the least $\mathtt{t} \geq \mathtt{t}_i + \ell$ such that $(\mathtt{t}, \mathtt{t} + \ell)$ is synchronous.

We suppose that every $\mathtt{t}_i$ is defined. For the sake of making things concrete, it is also useful to stipulate some specific values – a similar argument will hold for comparable values:

- Suppose $\Delta = 1$ second.
- Suppose $\mathtt{t}_0 < 10^5$ years and the maximum value $\mathtt{t}_{i+1} - \mathtt{t}_i$ is less than 10^5 years.
- For view v, suppose $\mathtt{t}$ is the first time at which $t+1$ correct processors are in view v, and that $\mathtt{t}'$ is the first time at which a correct processor sees a QC for view v. Define $\mathtt{u}_v := \mathtt{t}' - \mathtt{t}$. When $\mathtt{u}_v$ is defined, we suppose it always has at least the mininum value $\mathtt{u}$. For the sake of concreteness, we suppose $\mathtt{u} = 10^{-2}$ seconds.
- Suppose that $(\dagger_{\Delta,0})$ holds, and that $\Gamma = 2\Delta$.

Then we claim $(\dagger_{\Gamma,\mathtt{t}})$ holds for all $\mathtt{t}$ – this is the condition required to ensure that every interval $(\mathtt{t}_i, \mathtt{t}_i + \ell)$ has a correct leader produce a QC. Towards a contradiction, suppose there exists a least value i^* such that $(\dagger_{\Gamma,\mathtt{t}})$ fails to hold for some $\mathtt{t}^*$ in the interval $[\mathtt{t}_{i^*}, \mathtt{t}_{i^*+1})$. For each $\mathtt{t}$, let $\Gamma(\mathtt{t})$ be the smallest Γ' such that $(\dagger_{\Gamma',\mathtt{t}})$ holds. Note that:

- Every interval $(\mathtt{t}_i, \mathtt{t}_i + \ell)$ such that $i < i^*$ has at least one correct processor synchronise the clocks of correct processors to within time Δ, i.e. $(\dagger_{\Delta,\mathtt{t}})$ holds for some $\mathtt{t}$ in this interval.
- When a correct processor sees a QC and forwards its clock at $\mathtt{t}$, this may cause $\Gamma(\mathtt{t})$ to increase, e.g. if the leader is Byzantine and only sends the QC to certain processors, or if the QC is not sent during a synchronous interval. In this case, however, the maximum value of $\Gamma(\mathtt{t})$ is still at most $\Gamma - \mathtt{u}$.
- If a processor forwards its clock because it sees a VC at $\mathtt{t}$, this does not increase $\Gamma(\mathtt{t})$.

Define $\mathtt{t} := \mathtt{t}_{i^*-1} + \ell$ if $i^* \neq 0$, and define $\mathtt{t} := 0$ if $i^* = 0$. Let $\mathtt{t}^*$ be defined as above. We conclude that $\Gamma(\mathtt{t})$ is at most $\max\{\Delta, \Gamma - \mathtt{u}\}$, to within a small error term which is the maximum drift of clocks within an interval of length ℓ. Since we suppose $\mathtt{u} = 10^{-2}$ seconds, since our clocks have drift at most 1 second every 100 million years, and since some clocks may drift slow while others drift fast, this means that $\mathtt{t}^* - \mathtt{t} > 5 \times 10^5$ years. This gives the required contradiction, since we assumed above that $\mathtt{t}_0 < 10^5$ years and $\mathtt{t}_{i+1} - \mathtt{t}_i$ is less than 10^5 years for all i.

6 CONCLUDING COMMENTS

We have defined Fever, which is a novel view synchronisation protocol. If n is the number of processors and t is the largest integer $< n/3$, then Fever has resilience t, and in all executions with at most $0 \leq f \leq t$ Byzantine parties and network delays of at most $\delta \leq \Delta$ after *GST* (where f and δ are unknown), Fever has worst-case word complexity $O(fn + n)$ and worst-case latency $O(\Delta f + \delta)$. This improves significantly on the state-of-the-art.

The trade-off is that Fever requires greater assumptions than previous view synchronisation protocols regarding the drift of clocks prior to *GST*. We have argued in Section 5 that there are scenarios in which our required assumptions are reasonable. Atomic clocks can now be purchased

for a few thousand US dollars, and we showed that under reasonable assumptions regarding network latency etc., a system implementing Fever will be able to handle periods of asynchrony of the order of 10^5 years. Of course, this is more than is reasonably required, and so even the use of less accurate clocks may suffice in many scenarios.

Since there will certainly also be scenarios in which these additional assumptions are undesirable, it is a natural question as to whether they are necessary:

Question 1. *Does there exists a view synchronisation protocol for the partial synchrony model that achieves the same efficiency bounds as Fever, but which can accommodate unbounded clock drift prior to GST?*

REFERENCES

[1] Ittai Abraham, Guy Gueta, and Dahlia Malkhi. Hot-stuff the linear, optimal-resilience, one-message BFT devil. *CoRR*, abs/1803.05069, 2018. URL: http://arxiv.org/abs/1803.05069, arXiv:1803.05069.

[2] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 337–346, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293611.3331612.

[3] 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.

[4] Manuel Bravo, Gregory Chockler, and Alexey Gotsman. Liveness and Latency of Byzantine State-Machine Replication. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing (DISC 2022)*, volume 246 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2022/17203, doi:10.4230/LIPIcs.DISC.2022.12.

[5] Manuel Bravo, Gregory Chockler, and Alexey Gotsman. Making byzantine consensus live. *Distrib. Comput.*, 35(6):503–532, sep 2022. doi:10.1007/s00446-022-00432-y.

[6] Manuel Bravo, Gregory V. Chockler, and Alexey Gotsman. Making byzantine consensus live. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 23:1–23:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.23.

[7] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains, 2016. URL: http://hdl.handle.net/10214/9769.

[8] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017. URL: http://arxiv.org/abs/1710.09437, arXiv:1710.09437.

[9] Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. Byzantine consensus is $\Theta(n^2)$: The dolev-reischuk bound is tight even in partial synchrony! In Christian Scheideler, editor, *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*, volume 246 of *LIPIcs*, pages 14:1–14:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.DISC.2022.14.

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

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

[12] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, apr 1988. doi:10.1145/42282.42283.

[13] 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.

[14] Andrew Lewis-Pye. Quadratic worst-case message complexity for state machine replication in the partial synchrony model. *CoRR*, abs/2201.01107, 2022. URL: https://arxiv.org/abs/2201.01107, arXiv:2201.01107.

[15] Oded Naor, Mathieu Baudet, Dahlia Malkhi, and Alexander Spiegelman. Cogsworth: Byzantine View Synchronization. *Cryptoeconomic Systems*, 1(2), oct 22 2021. https://cryptoeconomicsystems.pubpub.org/pub/naor-cogsworth-synchronization.

[16] Oded Naor and Idit Keidar. Expected linear round synchronization: The missing link for linear byzantine SMR. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 26:1–26:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.26.

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

[18] Alexander Spiegelman. In search for an optimal authenticated byzantine agreement. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 38:1–38:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.DISC.2021.38.

[19] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

7 APPENDIX

The following fairly nuanced observation regarding Cogsworth and Naor-Keidar came out of a private conversation with some of the authors of those papers. For the purpose of this discussion, we assume some familiarity with both of those papers [15, 16].

In both Cogsworth and Naor-Keidar, there is a certain known bound, which we will call δ^* here, and which determines the delay before a correct processor gives up on one relay and tries the next. In those papers, δ^* is set to equal 2Δ, which gives the results described in Table 1. By playing with different values for δ^*, however, it is possible to achieve a trade-off between worst-case latency and expected complexity. While the corresponding results do not significantly impact the narrative of this paper, they may be of interest to the dedicated reader.

In the following discussion, we will consider only the case of a static adversary. We assume that δ^* may be much smaller than Δ, but is at most 2Δ. For Cogsworth, the possibility that δ^* may be much smaller than Δ now means that the worst-case latency is $O(f^2\delta^* + f\Delta + \delta)$, while the worst-case complexity remains $O(fn^2 + n)$. Expected latency is $O(\Delta)$ (and $O(\delta)$ if $f = 0$).

The issue with making δ^* very small is that, while this decreases the worst-case latency, it increases the *expected* complexity in the case that $\delta > \delta^*$. In this case, the expected complexity becomes $O(n^2)$ even with benevolent faults (the standard version of Cogsworth has expected complexity $O(n)$ in the case of benevolent faults).

For Naor-Keidar, if actual latency is at most δ^*, then worst-case latency is $O(f^2\delta^* + f\Delta + \delta)$ and worst-case communication is $O(f^2n + n)$. Expected latency is $O(\Delta)$ (and $O(\delta)$ if $f = 0$). The expected communication cost is $O(n)$. If actual latency is $> \delta^*$, then the worst-case latency is $O(f^2\delta^* + f\Delta + \delta)$ and worst-case communication is $O(n^2f + n)$. Expected latency in this case is $O(\Delta)$ and expected communication is $O(n^2)$. So, again, there is a trade-off. Setting a small value of δ^* decreases the worst-case latency, but increases the expected communication in the case that $\delta > \delta^*$.