Optimal Clock Synchronization with Signatures*

Christoph Lenzen
lenzen@cispa.de
CISPA Helmholtz Center for Information Security
Germany

Julian Loss
loss@cispa.de
CISPA Helmholtz Center for Information Security
Germany

ABSTRACT

Cryptographic signatures can be used to increase the resilience of distributed systems against adversarial attacks, by increasing the number of faulty parties that can be tolerated. While this is well-studied for consensus, it has been underexplored in the context of fault-tolerant clock synchronization, even in fully connected systems. Here, the honest parties of an n-node system are required to compute output clocks of small skew (i.e., phase offset) despite local clock rates varying between 1 and $\vartheta > 1$, end-to-end communication delays varying between $d - u$ and d, and the interference from malicious parties. Known algorithms with (trivially optimal) resilience of $\lceil n/2 \rceil - 1$ improve over the tight bound of $\lceil n/3 \rceil - 1$ holding without signatures for *any* skew bound [6, 18], but incur skew d [1] or $\Omega(n(u + (\vartheta - 1)d))$ [14]. Since typically $d \gg u$ and $\vartheta - 1 \ll 1$, this is far from the lower bound of $u + (\vartheta - 1)d$ that applies even in the fault-free case [3].

We prove tight bounds of $\Theta(u + (\vartheta - 1)d)$ on the skew for the resilience range from $\lceil n/3 \rceil$ to $\lceil n/2 \rceil - 1$. Our algorithm is, granted that the adversary cannot forge signatures, deterministic. Our lower bound holds even if clocks are initially perfectly synchronized, message delays between honest nodes are known, ϑ is arbitrarily close to one, and the synchronization algorithm is randomized. This has crucial implications for network designers that seek to leverage signatures for providing more robust time. In contrast to the setting without signatures, they must ensure that message delay is at least $d - u$, even on links with one faulty endpoint.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**.

KEYWORDS

Clock Synchronization, Digital Signatures, Fault Tolerance

ACM Reference Format:
Christoph Lenzen and Julian Loss. 2018. Optimal Clock Synchronization with Signatures. In *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC '22).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.48550/arXiv.2203.02553

*This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 716562).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
PODC '22, July 25–29, 2022, Salerno, Italy
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-9262-4/22/07…$15.00
https://doi.org/10.48550/arXiv.2203.02553

1 INTRODUCTION

Synchronizing clocks is a key task in distributed systems, which has received extensive attention over the years. Given that distributed systems are prone to faults and attacks, a sizeable fraction of this literature is dedicated to studying *fault-tolerant* clock synchronization. Under faults, the question of whether clocks can be synchronized has a fundamental impact: A message passing system prone to crash faults can simulate synchronous execution if and only if communication delay satisfies a known bound and time can be locally approximately measured. If either assumption fails to hold, the FLP proof of impossibility of deterministic consensus applies [10]. Conversely, if both assumptions hold, a network synchronizer [2] can be implemented by detecting crash faults via timeout. At the same time, these conditions are necessary and sufficient to compute logical clocks of bounded *skew*, i.e., bounded maximum difference of concurrent clock readings, on each connected component of the network.

Naturally, assuming crash faults is too optimistic in practice. However, for any less benign fault model, the situation is similar. On the one hand, from the above we know that it is necessary to have bounded delay and a local sense of the progress of time to be able to simulate synchronous execution. On the other hand, these assumptions imply that if we are given a synchronizer, (i) it is guaranteed to complete simulation of a round within bounded time and (ii) we can enforce an arbitrarily large minimum duration of each simulated round (by letting nodes sleep for some time). Intuitively, this allows us to solve the clock synchronization task by using the current round number as "target" logical clock value and from this compute logical clocks of bounded skew and rates by interpolation (this intuition is formalized in [7, Ch. 9, Sec. 3.3.4]).

In light of the above, the task of clock synchronization can be seen as a more general and precise version of running a synchronizer [2]:

- Logical clocks with bounded skew and rates of progress can be readily used to implement a synchronizer. The maximum duration of a simulated round then is $r(d + \mathcal{S})$, where r is the ratio between maximum and minimum clock rate, d is the (maximum) communication delay, and $\mathcal{S}$ is (the bound on) the skew.
- In contrast to a synchronizer, clocks can also be used to coordinate actions in terms of real time.

Note that both of these application scenarios share the property that controlling the rates of the computed logical clocks as well as their skew as precisely as possible matters.

To understand how well clocks can be synchronized under realistic faults or even malicious interference, the research community studied the clock synchronization task in the presence of Byzantine (i.e., worst-case) faults. In striking similarity to consensus [15], it is

possible to synchronize clocks in a fully connected message passing system with authenticated channels if and only if the number of Byzantine faults is strictly less than one third [6, 18]. This is good news: one can run synchronous consensus on top of clock synchronization, without any negative impact on resilience! Even better, this is also true with respect to performance: the same asymptotic bounds on skew can be achieved as in the fault-free case, without any loss in resilience [3, 17].

Concretely, denote by $d-u$ the *minimum* communication delay, i.e., the time between a message being sent and the receiving node completing to process it is between $d-u$ and d, and let $\vartheta > 1$ be the maximum rate of the local reference clocks, whose minimum rate we normalize to 1. The main result of [17] can then be read as saying that $\mathcal{S} \in O(u+(\vartheta-1)d)$ and $r \in 1+O(\vartheta-1)$ can be achieved, so long as $\vartheta < \vartheta_0$ for a constant ϑ_0 and strictly less than one third of the nodes are faulty, cf. [13]. This means that using the computed clocks to simulate synchronous execution, each simulated round takes $r(d+\mathcal{S}) \in d + O(u + (\vartheta-1)d)$ time. Given that in practice $u \ll d$ and $\vartheta - 1 \ll 1$, we have that $d + O(u + (\vartheta-1)d) \approx d$. In other words, synchrony can be simulated with negligible overhead in time!

For these theoretical results to be of practical value, it is paramount to minimize the overheads incurred by achieving fault-tolerance. While the resilience bound is tight under the assumption of authenticated channels, it is a well-known result that when *messages* are authenticated, consensus can be achieved in a fully connected system when up to $\lceil n/2 \rceil - 1$ nodes are faulty [9]. Indeed, also when synchronizing clocks the resilience can be boosted to $\lceil n/2 \rceil - 1$ by authenticating broadcasts [12, 18]. However, these algorithms have skew $\Theta(d) \gg u$. Using signature-based consensus, in [14] optimal resilience is achieved with skew $O(n(u+(\vartheta-1)d))$, where n is the number of nodes; replacing the consensus routine with a faster one could reduce, but not completely eliminate the dependence on n. This begs the question

> *"Which skew can be obtained with signatures at optimal resilience $\lceil n/2 \rceil - 1$?"*

Our Contribution

In this work, we show that the answer to this question is nuanced. To obtain an asymptotically optimal upper bound on the achievable skew, we set out on the track that readers familiar with the literature might expect. The algorithm from [17] is based on simulating iterations of synchronous *approximate agreement.*

Definition 1 (Approximate Agreement)**.** Let Π be protocol executed among n nodes where each node v holds an input $r_v \in \mathbb{R}$ and nodes terminate upon generating an output $o_v \in \mathbb{R}$. Denoting by $\mathcal{H}$ the set of honest nodes, we say that Π is a (ℓ, ϵ, f)*-secure protocol for approximate agreement* if the following properties hold whenever at most f nodes are corrupted and $\max_{v,w\in\mathcal{H}}\{r_v - r_w\} \leq \ell$:

- ϵ-Consistency: $\max_{v,w\in\mathcal{H}}\{o_v - o_w\} \leq \epsilon$.
- Validity: For all $v \in \mathcal{H}$, $\min_{w\in\mathcal{H}}\{r_w\} \leq o_v \leq \max_{w\in\mathcal{H}}\{r_w\}$.

Without signatures, this task can be solved if and only if fewer than one third of the nodes are faulty [8, 11]. The reason is that faulty nodes might claim different inputs to different nodes. Signatures can overcome this by being able to prove to others which input a sender claimed. While achieving consensus on inputs might take a non-constant number of rounds even with randomization, for approximate agreement it is sufficient that at most one value from each sender is accepted by correct nodes. This is easily achieved by outputting "$\perp$" – no value – if another node proved to receive a conflicting value. The resulting relaxed variant of reliable broadcast is referred to as *crusader broadcast* in the literature [4].

We show that when communication is by crusader broadcast, approximate agreement can be solved in a logarithmic number of rounds with resilience $\lceil n/2 \rceil - 1$.

Corollary 2. There is an $(\ell, \epsilon, \lceil n/2 \rceil - 1)$-secure protocol for approximate agreement running in $2\lceil \log(\ell/\epsilon) \rceil$ rounds.

In contrast to standard approximate agreement, for synchronization purposes the crucial "content" of the messages is their *timing*. With message delays between $d-u$ and d, we can simulate crusader broadcast in a timed fashion, which allows each node to estimate the offset between its own and other correct nodes in the system with an error of $O(u + (\vartheta-1)d)$. The property that faulty nodes communicate no different values then is relaxed, too, in that if two correct nodes accept a broadcast from the same sender, then their respective reception times agree up to $O(u + (\vartheta-1)d)$.

With this subroutine in place, we adapt the algorithm from [17] by replacing plain broadcasts by (simulated) crusader broadcasts and adjusting clocks in accordance with the more resilient approximate agreement algorithm with signatures. The algorithm solves so-called *pulse synchronization*, which (up to minor order terms) is equivalent to computing logical clocks of small skew and bounded rates at all times, cf. [7, Ch. 9, Sec. 3.3.3 and 3.3.4].

Definition 3 (Pulse Synchronization)**.** Let Π be a protocol executed among n nodes, where the set of honest nodes is $\mathcal{H}$. We say that Π is an f*-secure protocol for pulse synchronization with skew* $\mathcal{S}$*, minimum period* $P_{\min} > 0$*, and maximum period* $P_{\max}$ if the following properties hold whenever at most f nodes are corrupted:

- Liveness: for all $i \in \mathbb{N}_{>0}$ and each node $v \in \mathcal{H}$, v outputs pulse i exactly once. We denote $p_{v,i}$ as the time where v outputs its ith pulse.
- $\mathcal{S}$-bounded skew: $\sup_{i\in\mathbb{N}_{>0},v,w\in\mathcal{H}}\{|p_{v,i} - p_{w,i}|\} \leq \mathcal{S}$
- $P_{\min}$-minimum period:
$$\inf_{i\in\mathbb{N}}\{\min_{v\in\mathcal{H}}\{p_{v,i+1}\} - \max_{v\in\mathcal{H}}\{p_{v,i}\}\} \geq P_{\min}.$$
- $P_{\max}$-maximum period:
$$\sup_{i\in\mathbb{N}_{>0}}\{\max_{v\in\mathcal{H}}\{p_{v,i+1}\} - \min_{v\in\mathcal{H}}\{p_{v,i}\}\} \leq P_{\max}.$$

In case of a randomized algorithm, the skew bound is allowed to depend on the randomness of the algorithm; $\mathcal{S}$ is then defined as the expected worst-case value, where the expectation is taken over the randomness of the algorithm.

Corollary 4. If $\vartheta \leq 1.11$, there are choices $T \in \Theta(d)$ and $S \in \Theta(u+(\vartheta-1)d)$ such that Algorithm 3 is a $(\lceil n/2 \rceil - 1)$-secure pulse synchronization protocol with skew S, minimum period $P_{\min} \in \Theta(d)$ and maximum period $P_{\max} \in P_{\min} + \Theta(S)$.

Since this skew bound is asymptotically optimal even without faults [3], at first glance it might appear that this settles our above

question. However, there is a crucial difference to the signature-free setting. For the above result, it is necessary that also *faulty* nodes must obey the minimum message delay of $d - u$, both when receiving and sending messages. Otherwise, they could obtain and send a signature used by a correct sender in a crusader broadcast so early that correct nodes reject the sender's broadcast. This is in stark contrast to the algorithm from [17], for which faulty nodes can have full information of the system state at all times, including the future!

Given that it might be very challenging or even impossible for system designers to guarantee that an attacker must obey a minimum communication delay of $d-u$ for $u \ll d$, we need to determine whether this limitation is inherent. Perhaps surprisingly, we prove that this is indeed the case, by providing a matching lower bound. If either messages to or from faulty nodes have delays from $[d - \tilde{u}, d]$ for some $\tilde{u} \in [u, d]$, then we can prove a lower bound of $\Omega(\tilde{u})$ on the skew, regardless of u.

Theorem 5. Let $n \geq 3$ and Π be an $\lceil n/3 \rceil$-secure protocol for pulse synchronization with skew S. Then $\mathbb{E}[S] \geq 2\tilde{u}/3$.

Since a lower bound of $(\vartheta - 1)d$ follows from a simple indistinguishability argument even in absence of faults, we hence establish an asymptotically tight bound of $\Theta(\tilde{u} + (\vartheta - 1)d)$ on the skew that can be achieved when the number of faults is at least $n/3$. We stress that Theorem 5 imposes no restrictions on Π, which might be randomized, holds under perfect initial synchronization, for arbitrarily small $\vartheta - 1$ and $u = 0$, and our adversary is static, i.e., chooses which nodes to corrupt upfront. In other words, all the typical loopholes one might try to exploit to improve on our upper bound result are unavailable, implying asymptotic optimality of its skew in a strong sense.

Organization of this article. In Section 2, we specify our model and cover some preliminaries, including synchronous approximate agreement with resilience $\lceil n/2 \rceil - 1$. In Section 3, we provide and analyze our Crusader Pulse Synchronization algorithm. Section 4 proves the lower bound. Due to space constraints, discussion of further related work, some proofs, and a synchronous Crusader Broadcast algorithm are deferred to the full version [16].

2 PRELIMINARIES AND MODEL

We consider a network of n nodes connected by pairwise, authenticated channels. An unknown subset of the nodes is faulty or even malicious; we denote by $\mathcal{H}$ the set of the remaining honest nodes. We also assume that nodes have established a *public key infrastructure (PKI)*. This means that every node v has a *public key* pk_v that all other nodes agree on. Any honest node is also assumed to hold a matching *secret key* sk_v with which it can create a *signature* $\langle m \rangle_v$ on a message m via $\langle m \rangle_v \leftarrow \mathsf{Sign}(\mathsf{sk}_v, m)$. A signature can be verified via $\mathsf{Verify}(\mathsf{pk}_v, \langle m \rangle_v, m)$, which returns a bit 0 *(invalid)* or 1 *(valid)*. Therefore, we assume that a signature with respect to pk_v on any message m is impossible to create without knowledge of sk_v. We also assume *perfect correctness*: for any message $m \in \{0, 1\}$, $\mathsf{Verify}(\mathsf{pk}_v, \mathsf{Sign}(\mathsf{sk}_v, m), m) = 1$.

Assumptions on the Network. We use a continuous notion of time; thus time takes values in $\mathbb{R}_{\geq 0}$. We assume a fully connected network with known minimum and maximum delays of $d - u$ and d, respectively. This means that any message sent to or from an honest node is delivered after at most d time and no faster than $d - u$ time, where d and u are known parameters. We refer to u as the *uncertainty*. When proving our lower bound, we will allow for the possibility that messages to and from faulty nodes might violate the minimum delay bound of $d-u$ and instead satisfy only a weaker bound of $d-\tilde{u}$ for some $\tilde{u} \in [u, d]$. Note that we assume that the network is fault-free, since we are free to map link failures to node failures. Finally, we assume that only finitely many messages are sent in finite time; this must clearly be satisfied in any real-world network and it simplifies our lower bound construction by enabling us to perform induction over the messages sent by the honest parties executing an arbitrary, but fixed algorithm.

Hardware Clocks and Clock Rate. Nodes have no access to the "true" time $t \in \mathbb{R}_{\geq 0}$. Instead, each node v can measure the progress of time approximately via its *hardware clock*. The hardware clock of node v is modelled by a function $H_v \colon \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}_{\geq 0}$ which the node can evaluate at any point in time. That is, H_v maps time $t \in \mathbb{R}_{\geq 0}$ to a *local time* $H_v(t) \in \mathbb{R}_{\geq 0}$. We assume that hardware clocks run at rates between 1 and ϑ for a known constant $\vartheta > 1$, i.e., for all $t' \geq t \in \mathbb{R}_{\geq 0}$, it holds that

$$t' - t \leq H_v(t') - H_v(t) \leq \vartheta(t' - t).$$

Furthermore, we assume some degree of initial synchrony, represented by bounding $\max_{v,w \in \mathcal{H}}\{|H_v(0) - H_w(0)|\}$. For simplicity, we assume this bound to equal S when showing our upper bound; our lower bound result holds under the assumption of perfect initial synchrony, i.e., $H_v(0) = H_w(0)$ for all $v, w \in \mathcal{H}$.

Adversary and Executions. The adversary is in full control of message delays and hardware clocks within the bounds specified by the model. That is, the adversary specifies arbitrary hardware clock functions and message delays subject to the above constraints. Our algorithm is resilient to any adversary that cannot forge signatures, i.e., it is deterministically correct under this assumption. On the other hand, our lower bound holds for a static adversary that decides which nodes to corrupt upfront. The adversary may use corrupted nodes' secrets to generate signatures for them, but needs to obtain signatures of honest nodes affecting a message it intends to send before it can generate the message. Apart from this restriction, the adversary fully controls the behavior of faulty nodes.

Formally, an *execution* is fully specified by determining $\mathcal{H}$, H_v for each $v \in \mathcal{H}$, which messages faulty nodes send and when, and the delays of all messages. An execution is *well-defined*, i.e., conforms with our model, if these parameters specify the above specification and for each message m sent by a faulty node at time t, it receives all messages m' containing signatures from honest nodes that m depends on, some faulty node received a message m'' with the same signature by time t.[1]

Synchronous Execution and Rushing Adversary. For the sake of clarity of presentation, we use a classic synchronous model in our description and analysis of our solution to the approximate agreement problem. That is, computation proceeds in compute-send-receive rounds, and the goal is to complete the task in as few

[1] In our lower bound construction, we simply guarantee that the faulty sender of m receives m' by time t.

communication rounds as possible. While working in the synchronous model, we also assume a rushing adversary that can immediately observe honest nodes' messages in any given round and choose its own messages based on them. However, since our goal is to synchronize clocks in the above model, we will later show how to overcome this assumption in our main synchronization protocol.

Crusader Broadcast. In the synchronous setting, we assume a subroutine implementing Crusader Broadcast to be available; e.g. [5] provides one.

Definition 6 (Crusader Broadcast)**.** Let Π be protocol executed among n nodes where a designated dealer v holds an input $b_v \in \{0,1\}$ and nodes terminate upon generating an output $o_v \in \{0,1,\perp\}$. We say that Π is an *f-secure protocol for crusader agreement* if the following holds whenever at most f nodes are corrupted:

- Validity: If $v \in \mathcal{H}$, then $o_w = b_v$ for all $w \in \mathcal{H}$.
- Crusader Consistency: If $o_u \in \{0,1\}$ for some $u \in \mathcal{H}$, then for all $w \in \mathcal{H}$, $o_w \in \{\perp, o_u\}$.

2.1 Approximate Agreement with Signatures

In this section, we describe a simple synchronous 2-round $(\ell, \ell/2, \lceil n/2 \rceil - 1)$-secure approximate agreement algorithm, where we describe the protocol from the view of node v with input r_v and ℓ denotes the initial range of honest nodes' values. Note that a $(\ell, \epsilon, \lceil n/2 \rceil - 1)$-secure approximate agreement algorithm follows immediately by repeating the algorithm $\lceil \log \ell/\epsilon \rceil$ times, feeding the output of the previous iteration as input into the new instance.

The algorithm is given in Figure 1. Intuitively, its correctness follows from the facts that (i) nodes always discard enough values to ensure that the remaining ones lie within input range, where any received $\perp$ guarantees that the respective sender is faulty and does not contribute to the list of received non-$\perp$ values and (ii) the intervals spanned by the remaining values must intersect, implying that their midpoints can only be $\ell/2$ apart.

The first point is immediate from the properties of crusader broadcast. To show the second point, we argue as follows. First, observe that if there are no $\perp$-values received by honest parties, all of them have the same lists and, trivially, retain the same interval. Second, a node receiving a $\perp$-value can only increase the interval it retains, allowing us to extend this statement by induction on the number of $\perp$ values received by honest parties to all executions.

We now formalize this intuition. To this end, we denote as $I_v := [a_v, b_v]$ the interval spanned by the remaining values received by node $v \in \mathcal{H}$ in the first step (i.e., the ones that remain after discarding the highest and lowest values for that iteration). For any execution of the protocol in which some $v \in \mathcal{H}$ receives at least one $\perp$ value and any $x \in \mathbb{R}$, we denote by $\tilde{I}_v^x = [\tilde{a}_v, \tilde{b}_v]$ the interval I_v that results from an alternative execution of the protocol in which v receives identical messages except that one of the $\perp$ values is replaced by x.

Lemma 7. For all $x \in \mathbb{R}$ and all $v \in \mathcal{H}$, $\tilde{I}_v^x \subseteq I_v$.

Proof. Fix an arbitrary node $v \in \mathcal{H}$ receiving at least one $\perp$ value. For $0 < b \leq f$, denote by $r_v^1, ..., r_v^{n-k}$ the $n-b$ non-$\perp$ values that v receives in this iteration and assume they are ordered in ascending fashion. Similarly, denote by $\tilde{r}_v^1, ..., \tilde{r}_v^{n-k+1}$ the corresponding values in the alternative execution in which in addition x is received by v. Because the second list is identical to the old one except for exactly one additional value, we have for all $1 \leq j \leq n-b$ that $\tilde{r}_v^j \leq r_v^j \leq \tilde{r}_v^{j+1}$. In particular,

$$\tilde{a}_v = \tilde{r}_v^{f-(b-1)+1} \geq r_v^{f-b+1} = a_v$$

and

$$\tilde{b}_v = \tilde{r}_v^{n-(f-(b-1))} \leq r_v^{n-(f-(b-1))} \leq r_v^{n-(f-b)} = b_v,$$

i.e., $\tilde{I}_v^x = [\tilde{a}_v, \tilde{b}_v] \subseteq [a_v, b_v] = I_v$. □

Lemma 8. There exists $x \in \mathbb{R}$ such that $x \in I_v$ for all $v \in \mathcal{H}$.

Proof. We prove the claim by induction over the total number k of $\perp$ values received by honest nodes.

- **base case:** Suppose that $k = 0$. Thus, for all $v, w \in \mathcal{H}$, $I_v = I_w$ by crusader consistency of CB. Since $f = \lceil n/2 \rceil - 1$, each node retains at least one value, i.e., this interval is non-empty.
- **step from k to $k+1$:** Suppose the claim holds for $k \in \mathbb{N}$. By the induction hypothesis, there is $x \in \mathbb{R}$ such that $x \in I_v$ for all $v \in \mathcal{H}$. Consider an alternative execution in which all nodes receive the same messages except that for some honest node w, one of its values is replaced by $\perp$. Denote by $\tilde{I}_v$, $v \in \mathcal{H}$, the intervals in this new execution. For each honest node $v \neq w$, we have that $x \in I_v = \tilde{I}_v$. For w, we have that $x \in I_w \supseteq \tilde{I}_w$ by Lemma 7.

Since this construction applies for any inputs r_v, $v \in \mathcal{H}$, and we imposed no restrictions on the values received from faulty nodes, the claim of the lemma follows for all executions of the algorithm. □

Theorem 9. Algorithm 1 is an $(\ell, \ell/2, \lceil n/2 \rceil - 1)$-secure 2-round protocol for approximate agreement.

Proof. Without loss of generality, we order the inputs of the honest nodes in ascending order. As there are at most $f = \lceil n/2 \rceil - 1$ corrupted nodes and CB has validity, we have that $r_1 \leq a_v$ and $r_{n-f} \geq b_v$ for all $v \in \mathcal{H}$. Hence, $o_v = (a_v + b_v)/2$ satisfies the validity condition of approximate agreement.

By Lemma 8, there is $x \in \mathbb{R}$ such that $x \in I_v = [a_v, b_v]$ for all $v \in \mathcal{H}$. For all $v, w \in \mathcal{H}$, we can hence infer that

$$\begin{aligned} o_v - o_w &= \frac{a_v + b_v - a_w - b_w}{2} \\ &\leq \frac{x + r_{n-f} - r_1 - x}{2} = \frac{r_{n-f} - r_1}{2} \leq \frac{\ell}{2}. \end{aligned}$$ □

Corollary 2 readily follos by inductive application of Theorem 9, where the output of the ith run is used as input for the $(i+1)$th run.

3 CRUSADER PULSE SYNCHRONIZATION

In this section, we provide and prove correct our pulse synchronization algorithm. In analogy to the classic algorithm by Lynch and Welch [17] that achieves asymptotically optimal skew in the signature-free setting, the algorithm can be viewed as simulating iterations of synchronous approximate agreement. The goal is to agree on the pulse times, i.e., pull them closer together despite interference from faulty nodes.

Algorithm APA

- v sends r_v to all nodes using (an instance of) Algorithm CB
- Denote r_w the value received from w (which might be $\perp$). Determine o_v as follows:
  - Denote b as the number of CB instances that output $\perp$ in the previous step.
  - Sort all the non-$\perp$ values received via the instances of CB in the previous step, then discard the lowest $f-b$ and the highest $f-b$ of those values. Denote I the interval spanned by the remaining values.
  - Output the midpoint o_v of I.

Figure 1: A synchronous 2-round $(\ell, \ell/2, f)$-secure algorithm for approximate agreement, where $f = \lceil n/2 \rceil - 1$.

There are two notable differences from running "plain" approximate agreement. One is that the goal is to agree on (real) times, which cannot directly be accessed by the nodes. This results in the algorithm using communication and the hardware clocks to estimate the *differences* between pulse times. Because these measurements are inexact and clocks drift, the second difference emerges: while approximate agreement works towards decreasing the error in each iteration, (in the worst case) these inaccuracies in the nodes' perception work towards increasing them.

In the following, assume parties are running an algorithm for pulse synchronization, where $\vec{p}_r$ denotes the pulse times of honest parties for the rth pulse.[2] We define $\|\vec{p}_r\| := \max_{v\in\mathcal{H}}\{p_v^r\} - \min_{v\in\mathcal{H}}\{p_v^r\}$. Denote by S the upper bound on $\|\vec{p}_r\|$ for all $r \in \mathbb{N}$ that we are going to show. Note that the algorithm is allowed to make use of S, even though we will be able to determine S only once our analysis is complete.

3.1 Timed Crusader Broadcast

Before presenting the full algorithm, let us discuss how estimates of clock offsets can be obtained that comply with the requirements of (our simulation of) Algorithm APA. Intuitively, instead of communicating the (unknown) pulse times, each node $v \in \mathcal{H}$ will broadcast (up to a small, fixed local time offset) when locally generating its pulse. Knowing that these messages are underway for about d time, each recipient $w \in \mathcal{H}$ can then determine an approximation of $\Delta_{v,w}^r$ of $p_w^r - p_v^r$ of error $O(u + (\vartheta-1)d)$.

In order for these estimates to be used in the simulation of Algorithm APA, we need them to satisfy a timed analogon of Crusader Consistency. In the absence of any error, this would mean that for any two honest parties $v, w \in \mathcal{H}$ and (possibly faulty) node $x \in \mathcal{H}$ such that $\Delta_{v,x}^r, \Delta_{w,x}^r \neq \perp$, it holds that $\Delta_{v,x}^r - \Delta_{w,x}^r = p_w^r - p_v^r$. Under this condition, the estimates $p_v^r + \Delta_{v,x}^r$ and $p_w^r + \Delta_{w,x}^r$ are equal, i.e., it is as if faulty node x had broadcast to a subset of the honest nodes at some specific time p_x^r that each $v \in \mathcal{H}$ with $\Delta_{v,x}^r \neq \perp$ agrees on. Naturally, we cannnot ensure such an exact match, but signatures give us the possibility to prove reception time up to an error of $O(u + (\vartheta-1)d)$. By rejecting the broadcast message from node x if another node proves to have received the same "broadcast" more than u (real) time earlier, we can hence ensure the above consistency condition up to an error of $O(u + (\vartheta-1)d)$. The protocol ensuring these guarantees is given in Figure 2.

We now formalize the above claims. We first establish that honest dealers' messages are always accepted, i.e., recipients will output their local reception time, which corresponds to validity of Crusader Broadcast.

Lemma 10. Let $r \in \mathbb{N}$ and suppose that $\|\vec{p}^r\| \leq S$. Then for all $v, w \in \mathcal{H}$, v outputs $h \neq \perp$ in TCB^r with w as the sender.

Proof. We begin by proving that w's signature is received by v at a local time $h \in (H_v(p_v^r), H_v(p_v^r) + \vartheta(d + (\vartheta+1)S))$. Denote by t_w the (unique) time at which w sends $\langle r \rangle_w$ and let $t_v \in [t_w + d - u, t_w + d]$ be the time at which v receives it. We observe that

$$S = \frac{H_w(t_w) - H_w(p_w^r)}{\vartheta} \leq t_w - p_w^r \leq H_w(t_w) - H_w(p_w^r) = \vartheta S.$$

Hence,

$$\begin{aligned} H_v(t_v) \geq H_v(t_w) &= H_v(t_w) - H_v(p_v^r) + H_v(p_v^r) \\ &\geq t_w - p_v^r + H_v(p_v^r) \\ &\geq p_w^r - p_v^r + S + H_v(p_v^r) \geq H_v(p_v^r), \end{aligned}$$

where the last step uses that $\|\vec{p}^r\| \leq S$ by assumption. For the upper bound on h, we get that

$$\begin{aligned} H_v(t_v) &\leq H_v(t_w + d) \\ &\leq H_v(p_w^r + \vartheta S + d) - H_v(p_v^r) + H_v(p_v^r) \\ &\leq \vartheta(p_w^r - p_v^r + \vartheta \cdot S + d) + H_v(p_v^r) \\ &\leq \vartheta((\vartheta+1)S + d) + H_v(p_v^r). \end{aligned}$$

It remains to prove that v receives no correct signature of the form $\langle r \rangle'_w$ at a local time $h' \in (H_v(p_v^r), h + d - 2u)$ from $x \neq w$. Since w sends no other signature than $\langle r \rangle_w$ and signatures cannot be forged, any such message must be sent after x learned $\langle r \rangle_w$, at time $t_x \geq t_w + d - u$. Hence, v receives any such message at a time $t'_v \geq t_x + d - u \geq t_w + 2(d-u)$. On the other hand $t_v \leq t_w + d$, implying that $t'_v - t_v \geq d - 2u$. We conclude that $h' - h = H_v(t'_v) - H_v(t_v) \geq t'_v - t_v \geq d - 2u$. □

Having established that honest nodes broadcasts are accepted, let us establish the counterpart of Crusader Consistency, namely that if honest nodes accept a broadcast, they do so within a short time of each other.

Lemma 11. Let $r \in \mathbb{N}$ and $u, v \in \mathcal{H}$. Suppose that u, v participate in an instance of TCB^r with dealer w and output $h_u, h_v \notin \{\perp\}$, respectively. Let t_u, t_v denote the times at which u and v receive the messages from w, respectively. Then $|t_u - t_v| \leq (1 - 1/\vartheta) \cdot d + 2u/\vartheta$.

Proof. Denote as t_u, t_v the times that u and v receive a message from w in TCB^r s.t. $H_u(t_u) \in (H_u(p_u^r), H_u(p_u^r) + \vartheta(d + \vartheta \cdot S))$ (and similarly for v). If for either u or v no such time exists, then that party outputs $\perp$ and the claim is vacuously true.

[2] Since a priori there is no formal guarantee that $v \in \mathcal{H}$ will indeed generate all pulses, one can define $p_v^r := \infty$ if v generates fewer than r pulses. However, our inductive proof does not need to reason about $p_{v,r+1}$ before it is established that it is finite.

Algorithm TCB^r

We describe the protocol from the view of node v, where w is the dealer.

- If v is the dealer, i.e., $v = w$, v sends $\langle r \rangle_v$ to all nodes at local time $H_v(p_v^r) + \vartheta \cdot S$ and terminates.
- If v is not the dealer, i.e., $v \neq w$:
  - If no correctly formed signature $\langle r \rangle_w$ is received from w at a local time $h \in (H_v(p_v^r), H_v(p_v^r) + \vartheta(d + (\vartheta+1)S))$, terminate with output $\perp$.
  - Otherwise, denote by h the first such local time. Forward $\langle r \rangle_w$ to all nodes at time h.
  - If v receives a correctly formed signature $\langle r \rangle'_w$ at a local time $h' \in (H_v(p_v^r), h + d - 2u)$ from a node $x \neq w$, it terminates with output $\perp$.
  - Otherwise, v terminates with output h at local time $h + d - 2u$.

Figure 2: Timed Crusader Broadcast. The routine assumes that $\|\vec{p}_r\| \leq S$, where p_v^r is the time at which v generates the rth pulse. Encoding $r \in \mathbb{N}$ allows to distinguish instances, so that faulty nodes cannot reuse "old" signatures to disrupt an instance.

Without loss of generality, assume that $t_u \geq t_v$. u receives the echo from v at time $t \leq t_v + d$. Since u does not output $\perp$, we have that $\vartheta \cdot (t - t_u) \geq H_u(t) - H_u(t_u) \geq d - 2u$. This implies that $t - t_u \geq (d - 2u)/\vartheta$. It follows $t_u - t_v = t - t_v - (t - t_u) \leq d - (d - 2u)/\vartheta = (1 - 1/\vartheta) \cdot d + 2u/\vartheta$. □

3.2 Pulse Synchronization Algorithm

With Timed Crusader Broadcast in place, we are ready to proceed to the main algorithm, i.e., iterative simulation of synchronous approximate agreement steps on pulse times. The algorithm is given in Figure 3.

Intuitively, in each iteration $r \in \mathbb{N}$ the algorithm lets each node w communicate its pulse time using crusader broadcast. The output of this subroutine at v is then used to compute an estimate $\Delta_{v,w}^r$ of $p_w^r - p_v^r$, which in case of a faulty sender might fail and result in $\perp$. These estimates are then used exactly as in Algorithm APA to determine a value of Δ, which is then used as a correction to the next pulse time relative to the nominal duration of an iteration of T. The fact that Algorithm APA is used on the *differences* of pulse times is compensated for by adding its output to the previous local pulse time of $H_v(p_v^r)$; since substracting p_v^r does not change the order of the received values and the output is a convex combination of two inputs of a specific rank, up to the errors introduced by clock drift and delay uncertainty, this is equivalent to executing Algorithm APA on inputs p_v^r.

We begin our analysis of the algorithm by translating the validity and consistency guarantees of Algorithm TCB^r into corresponding guarantees on the computed estimates $\Delta_{v,w}^r$. In the following, we will bound the error in the estimates by $\delta := 2u + (\vartheta^2 - 1)d + 2(\vartheta^3 - \vartheta^2)S$. First, we show validity, i.e., that honest nodes compute estimates of their difference in pulse times with error smaller than δ. The result is shown analogously to [7, Ch. 10, Lem. 10.8].

Lemma 12. Let $r \in \mathbb{N}$ and suppose that $\|\vec{p}^r\| \leq S$. Let $v, w \in \mathcal{H}$ and suppose that v participates in an instance of TCB^r with dealer w. Consider $\Delta_{v,w}^r$ as defined in algorithm CPS. Then $\Delta_{v,w}^r \in [p_w^r - p_v^r, p_w^r - p_v^r + \delta)$.

Proof Sketch. The computation of $\Delta_{v,w}^r$ accounts for the minimum time (and hence local time) that passes before v receives the message from w, which is accepts by Lemma 10. The upper bound follows by checking the maximum local time that passes, which is proportional to ϑ times the real difference between the pulse times plus a delay uncertainty. □

Next, we show the counterpart of Crusader Consistency, i.e., that non-$\perp$ estimates are consistent up to error δ.

Lemma 13. Let $r \in \mathbb{N}$ and suppose that $\|\vec{p}^r\| \leq S$. Moreover, let $v, w \in \mathcal{H}$, $x \in [n]/\mathcal{H}$, and $h_v, h_w \notin \{\perp\}$ denote the outputs of v and w in TCB^r with sender x. Then $|\Delta_{v,x}^r - \Delta_{w,x}^r - (p_w^r - p_v^r)| < \delta$.

Proof. Let again t_v and t_w denote the times at which v and w receive the messages from the dealer x. By Lemma 11, it holds that $t_v - t_w \leq (1 - 1/\vartheta)d + 2u/\vartheta$. We get that

$$
\begin{aligned}
&\Delta_{v,x}^r - \Delta_{w,x}^r - (p_v^r - p_u^r) \\
&= H_v(t_v) - H_v(p_v^r) - (H_w(t_w) - H_w(p_w^r)) - (p_w^r - p_v^r) \\
&\leq \vartheta(t_v - p_v^r) - (t_w - p_w^r) - (p_v^r - p_w^r) \\
&= (\vartheta - 1)(t_v - p_v^r) + t_v - t_w \\
&\leq (\vartheta - 1)(t_v - p_v^r) + \left(1 - \frac{1}{\vartheta}\right) d + \frac{2u}{\vartheta} \\
&\leq (\vartheta - 1)(H_v(t_v) - H_v(p_v^r)) + \left(1 - \frac{1}{\vartheta}\right) d + \frac{2u}{\vartheta} \\
&\leq (\vartheta - 1)(\vartheta d + (\vartheta^2 + \vartheta)S) + \left(1 - \frac{1}{\vartheta}\right) d + \frac{2u}{\vartheta} < \delta.
\end{aligned}
$$

□

Based on the above bounds, we will prove by induction that for all $r \in \mathbb{N}$, $\|\vec{p}^r\| \leq S$. The base of the induction is given by the assumption that hardware clocks are initialized with skew S. The following lemma is the key argument required for the step, establishing that, essentially, an approximate agreement step on the pulse times with error at most δ is performed, assuming that the estimates satisfy the above validity and consistency conditions.

Lemma 14. Fix $r \in \mathbb{N}$. Suppose that each $v \in \mathcal{H}$ computes for each $w \in [n]$ a value $\Delta_{v,w}^r \in \mathbb{R} \cup \{\perp\}$, such that the following properties hold:

- For $v, w \in \mathcal{H}$, $\Delta_{v,w}^r \in [p_w^r - p_v^r, p_w^r - p_v^r + \delta]$.
- For $v, w \in \mathcal{H}$ and $x \in [n]$ such that $\Delta_{v,x}^r, \Delta_{w,x}^r \notin \{\perp\}$, $|\Delta_{v,x}^r - \Delta_{w,x}^r - (p_w^r - p_v^r)| \leq \delta$.

Then the following statements are true:

(1) For all $v \in \mathcal{H}$: $-\|\vec{p}^r\| \leq \Delta_v^r \leq \|\vec{p}^r\| + \delta$.
(2) $\|\vec{\Delta}^r + \vec{p}^r\| \leq \|\vec{p}^r\|/2 + \delta$.

Proof. We first show the claim for the special case $\delta = 0$. Consider an honest node $v \in \mathcal{H}$ and a faulty node $x \in [n] \setminus \mathcal{H}$ such that $\Delta_{v,x}^r \neq \perp$. We define $p_x^r := \Delta_{v,x}^r + p_v^r$. We observe that

Algorithm CPS

We describe the protocol from the view of node v. Wait until local time S.

Then do for all $r \in \mathbb{N}$:

- Generate rth pulse.
- Simultaneously participate in an execution of TCB^r with sender w, for each node $w \in [n]$. Let $h_{v,w}$ denote the output of the instance corresponding to sender w.
- For each node $w \in [n]$ s.t. $h_{v,w} \neq \perp$, compute $\Delta^r_{v,w} := h_{v,w} - H_v(p^r_v) - d + u - S$. For all remaining $w \in [n]$, set $\Delta^r_{v,w} := \perp$. Denote b the number of $\perp$ values.
- Sort all the non-$\perp$ values computed in the previous step, then discard the lowest $f - b$ and the highest $f - b$ of those values. Denote I the interval spanned by the remaining values.
- Set Δ^r_v to the midpoint of I.
- Wait until local time $H_v(p^r_v) + \Delta^r_v + T$.

Figure 3: Crusader Pulse Synchronization, where $f = \lceil n/2 \rceil - 1$ is the number of faulty nodes that can be sustained. The algorithm assumes that $H_v(0) \in [0, S]$ for all $v \in \mathcal{H}$.

$p^r_v + \Delta^r_{v,x} = p^r_w + \Delta^r_{w,x}$ for any correct node $w \in \mathcal{H}$ from the second condition of the statement and the assumption that $\delta = 0$. Hence $p^r_x = p^r_w + \Delta^r_{w,x}$ for all $w \in \mathcal{H}$ satisfying $\Delta^r_{w,x} \neq \perp$.

We next show that $p^r_v + \Delta^r_v$ for $v \in \mathcal{H}$ equals the output of v in an iteration of an execution of algorithm APA specified as follows:

- Nodes $v \in \mathcal{H}$ have input p^r_v.
- For nodes $x \in [n] \setminus \mathcal{H}$ such that CB with dealer x outputs $\Delta^r_{v,x} \neq \perp$ at $v \in \mathcal{H}$, v receives p^r_x from x.

We consider the vector L of values used to compute the value Δ^r_v in iteration r of CPS. For $w \in [n]$, $\Delta^r_{v,w} = p^r_w - p^r_v$. Hence, L can be obtained by shifting the non-$\perp$ input values used in the above execution of APA by $-p^r_v$. This implies that in both computations, v assigns the input corresponding to node w to the same position i_w in the sorted vector of non-$\perp$ inputs. Hence, the indices of discarded inputs in the list L remains the same in both of these executions as well. Let $\ell = |L|$ denote the length of L and $b := n - \ell$ the number of $\perp$ values. Without loss of generality, we assume L to be sorted in ascending order, and that parties are sorted in ascending order by size of their pulse times $\vec{p}$. Then in the rth iteration of CPS, v computes the midpoint Δ^r_v of interval I_L spanned by the remaining points in L as

$$\begin{aligned}\Delta^r_v &= \frac{L_{\ell-f+b} + L_{f-b+1}}{2} \\ &= \frac{(p^r_{\ell-f+b} - p^r_v) + (p^r_{f-b+1} - p^r_v)}{2} \\ &= \frac{p^r_{\ell-f+b} + p^r_{f-b+1}}{2} - p^r_v.\end{aligned}$$

By comparison, the above execution of APA computes the midpoint as $(p^r_{\ell-f+b} + p^r_{f-b+1})/2 = M_v$ for every node $v \in \mathcal{H}$. Thus, both executions compute the same midpoint, up to a shift of p^r_v. That is, CPS computes the vector of midpoints $\Delta^r_v = M_v - p^r_v$ for every node $v \in \mathcal{H}$. As APA satisfies the 1/2-consistency condition of approximate agreement, we know that the above execution of APA computes midpoints $\vec{M}$ satisfying $\|\vec{M}\| \leq \|p^r\|/2$. It follows that $\|\vec{\Delta}^r + \vec{p}^r\| = \|\vec{M}\| \leq \|\vec{p}^r\|/2$. This proves the second statement of the lemma.

To prove the first statement, we use the validity condition of approximate agreement. We obtain for all $v \in \mathcal{H}$ that $\min_{w\in\mathcal{H}}\{p^r_w\} \leq M_v \leq \max_{w\in\mathcal{H}}\{p^r_w\}$. Since $\vec{\Delta}^r = \vec{M} - \vec{p}^r$, we obtain for all $v \in \mathcal{H}$ that

$$\begin{aligned}-\|p^r\| = \min_{w\in\mathcal{H}}\{p^r_w\} - \max_{w\in\mathcal{H}}\{p^r_w\} \leq M_v - p^r_v = \Delta^r_v \\ \leq \max_{w\in\mathcal{H}}\{p^r_w\} - \min_{w\in\mathcal{H}}\{p^r_w\} = \|p^r\|.\end{aligned}$$

For the general case of $\delta > 0$, we note that the list of values received by $v \in \mathcal{H}$ can be obtained from the one of an execution with $\delta = 0$ by adding to each received value a shift between 0 and δ. Thus, our analysis for the case of $\delta = 0$ implies that

$$\begin{aligned}\frac{p^r_{\ell-f+b} + p^r_{f-b+1}}{2} - p^r_v \leq \Delta^r_v \leq \frac{p^r_{\ell-f+b} + \delta + p^r_{f-b+1} + \delta}{2} - p^r_v \\ = \frac{p^r_{\ell-f+b} + p^r_{f-b+1}}{2} - p^r_v + \delta.\end{aligned}$$

We conclude that $-\|\vec{p}^r\| \leq \Delta^r_v \leq \|\vec{p}^r\| + \delta$, showing the first claim of the lemma. Moreover, $\Delta^r_v + p^r_v \in [M_v, M_v + \delta]$ and hence, $\|\Delta^r + p^r\| \leq \|\vec{M}\| + \delta \leq \|p^r\|/2 + \delta$. □

An immediate consequence of Lemma 14 is that if the hypothesis of our induction holds, i.e., $\|\vec{p}^r\| \leq S$, then the computed shifts $\vec{\Delta}^r$ are feasible, in the sense that $v \in \mathcal{H}$ the waiting statement at the end of the main loop of Algorithm CPS refers to a local time that is larger then the local time when all instances of crusader broadcast for round r have terminated at v, so that v can compute Δ^r_v.

Corollary 15. Fix $r \in \mathbb{N}$. Suppose that $T \geq (\vartheta^2 + \vartheta + 1)S + (\vartheta + 1)d - 2u$ and $\|\vec{p}^r\| \leq S$. Moreover, denote as τ^r_v the time at which $v \in \mathcal{H}$ finalizes the computation of Δ^r_v. Then for all $v \in \mathcal{H}$, $H_v(p^r_v) + \Delta^r_v + T \geq H_v(\tau^r_v)$.

Proof of Corollary 15. First, we note that Algorithm TCB terminates at the latest at local time $H_v(p^r_v) + (\vartheta + 1)d - 2u + (\vartheta^2 + \vartheta)S$. Hence, $H_v(\tau^r_v) \leq H_v(p^r_v) + (\vartheta + 1)d - 2u + (\vartheta^2 + \vartheta)S$. Moreover, due to the condition $\|\vec{p}^r\| \leq S$, we can apply Lemmas 12 and 13, which prove that the preconditions of Lemma 14 hold for iteration r. We apply the precondition $\|\vec{p}^r\| \leq S$ and the first statement of Lemma 14, which together yields that for all $v \in \mathcal{H}$, $-S \leq -\|\vec{p}^r\| \leq \Delta^r_v$. Thus,

$$\begin{aligned}&H_v(p^r_v) + \Delta^r_v + T \\ &\geq H_v(p^r_v) - S + T \\ &\geq H_v(\tau^r_v) - ((\vartheta + 1)d - 2u + (\vartheta^2 + \vartheta + 1)S) + T \geq H_v(\tau^r_v).\end{aligned}$$ □

Similarly to [7, Ch. 10, Lem. 10.7], we can now prove that the induction steps succeeds, i.e., that if $\|\vec{p}^r\| \leq S$, then also $\|\vec{p}^{r+1}\| \leq S$.

Lemma 16. Suppose that $T \geq (\vartheta^2+\vartheta+1)S + (\vartheta+1)d - 2u$, $S \geq (2(2\vartheta-1)\delta + 2(\vartheta-1)T)/(2-\vartheta)$, and $\|\vec{p}^r\| \leq S$. Then

- $(T-S)/\vartheta \leq \min_{w\in\mathcal{H}}\{p_w^{r+1}\} - \min_{w\in\mathcal{H}}\{p_w^r\} \leq T + S + \delta$ and
- $\|\vec{p}^{r+1}\| \leq S$.

Proof Sketch. Lemma 14 allows us to interpret the new pulse times as the result of performing an approximate agreement step with error δ on the previous pulse times (up to adding T and clock drift). The first statement follows from the resulting bound of $-S \leq \|\vec{p}^r\| \leq \Delta_v^r \leq \|\vec{p}^r\| + \delta \leq S + \delta$. The second statement is shown by using that approximate agreement reduces the "old" error carried over from the previous iteration by $\|\vec{p}^r\|/2$, which compensates for the "new" error due to the measurement error of δ and clock drift. □

The main theorem is shown analogously to [7, Ch. 10, Thm. 10.9].

Theorem 17. Suppose that $4 - \vartheta + \vartheta^2 - 3\vartheta^3 > 0$ and

$$T \geq \frac{(\vartheta^2+\vartheta+1)2(2\vartheta-1)(2u+(\vartheta^2-1)d)}{4-\vartheta+\vartheta^2-3\vartheta^3} + (\vartheta+1)d - 2u \in O(d).$$

Define S as

$$S := \frac{2(2\vartheta-1)(u+(\vartheta-1)d) + 2(\vartheta-1)T}{4-2\vartheta-\vartheta^2} \in O(u + (1-1/\vartheta)T),$$

and suppose that $\max_{v\in\mathcal{H}}\{H_v(0)\} \leq S$. Then Algorithm CPS is an $(\lceil n/2\rceil - 1)$-secure clock synchronizaton protocol with skew S, minimum period $P_{\min} \geq (T - (\vartheta+1)S)/\vartheta$, and maximum period $P_{\max} \leq T + 3S$.

Proof Sketch. S and T are chosen in accordance with the prerequisites of Lemma 16, from which the claims readily follow. As $T \in \Omega(S)$ and $S \in \Omega((\vartheta-1)T$, this is only feasible if $\vartheta - 1$ is small enough. □

By checking for which values of $\vartheta > 1$ the polynomial in ϑ in the preconditions of the theorem is positive, we arrive at Corollary 4.

Corollary 4. If $\vartheta \leq 1.11$, there are choices $T \in \Theta(d)$ and $S \in \Theta(u + (\vartheta-1)d)$ such that Algorithm 3 is a $(\lceil n/2\rceil - 1)$-secure pulse synchronization protocol with skew S, minimum period $P_{\min} \in \Theta(d)$ and maximum period $P_{\max} \in P_{\min} + \Theta(S)$.

4 LOWER BOUND ON THE SKEW

Algorithm CPS exploits signatures in order to prevent faulty nodes from equivocating about their own perception of time, i.e., when they broadcast. However, echoing signatures comes not only at the expense of another message delay (and thus timing uncertainty u), but also at the expense of adding indirection to communication relevant for the algorithm. As a result, the timing delay uncertainty determining its skew bound is actually $\tilde{u}$, the uncertainty on communication links involving a faulty party, which might be larger than u.

In this section, we prove that this restriction is inherent. Concretely, we show that no algorithm can achieve a skew smaller than $2\tilde{u}/3$, even if $u = 0$. Our lower bound is established by the standard technique of manipulating hardware clocks in a way that is hidden by adjusting message delays, enabling us to construct three executions that are indistinguishable to honest nodes. Two executions are indistinguishable to an honest node if the sequence of received messages as well as the local reception times are identical; in this case, the algorithm must send the same messages at the same local times and generate pulses at the same time.[3] The hardware clock skew of $2\tilde{u}/3$ we build up then translates to a skew of at least $2\tilde{u}/3$ between pulses in at least one of the three executions.

In our setting, this approach faces the technical challenge that it is not sufficient to define the executions such that hardware clocks and message delays conform to the model, followed by proving their indistinguishability. In addition, we must prove that the adversary can obtain the required knowledge to determine its strategy upfront and have faulty nodes send the required messages to maintain indistinguishability in time.

We now set up to formally prove the lower bound. Fix an arbitrary deterministic pulse synchronization algorithm $\mathcal{A}$. For convenience of notation, w.l.o.g. we assume that no two messages are received at the exact same local time; the general case can be covered by breaking ties lexicographically. Also w.l.o.g., we can assume $n = 3$; the general case follows from a simple simulation argument we provide later. In order to distinguish the three executions we construct, we use superscripts $i \in [3]$. For notational convenience, both execution and node indices are always taken modulo 3, which also exposes the symmetry of the construction.

Throughout this section, all the execution triples $(\mathsf{Ex}^i)_{i\in[3]}$ we consider share the following properties P:

- $\mathcal{H}^i = [3] \setminus \{i\}$.
- Messages sent between honest parties have delay d.
- All other messages have delay $d - \tilde{u}$.
- $H_{i+1}^i(t) = t$.
- $H_{i+2}^i(t) = \vartheta t$ for $t \leq 2\tilde{u}/(3(\vartheta-1))$ and $H_{i+2}^i(t) = t + 2\tilde{u}/3$ for $t \geq 2\tilde{u}/(3(\vartheta-1))$.

In particular, hardware clocks of honest nodes never deviate by more than $2\tilde{u}/3$ from real time.

We construct our triple of executions by induction over the number of messages k sent by faulty nodes, where indistinguishability holds before the maximum local reception time of such a message. The base case is trivial; the following lemma performs the step.

Lemma 18. Let $n = 3$, $k \in \mathbb{N}$, and Π be a 1-secure protocol for pulse synchronization with skew S. Suppose there exist executions $\mathsf{Ex}^1, \mathsf{Ex}^2, \mathsf{Ex}^3$ of Π satisfying P and the following properties.

- Faulty nodes send a total of k messages in Ex^1, Ex^2, and Ex^3.
- Let h^* be the maximum local time at which a message from a faulty node is received (or 0 if faulty nodes send no messages). Honest node i cannot distinguish Ex^{i+1} and Ex^{i+2} until local time h^*.

[3] For a randomized algorithm, this holds only after fixing the randomness. However, the strategy of the adversary is independent of the algorithm, which allows us to prove the result for deterministic algorithms and infer the general statement by Yao's principle.

If there exists a node i that can distinguish Ex^{i+1} and Ex^{i+2}, then there exist executions $\widetilde{\mathsf{Ex}}^1$, $\widetilde{\mathsf{Ex}}^2$, and $\widetilde{\mathsf{Ex}}^3$ with the same properties, except that faulty nodes send an additional message which is received at a local time $h > h^*$. Moreover, for $i \in [3]$, nodes $i+1$ and $i+2$ cannot distinguish $\widetilde{\mathsf{Ex}}^i$ from Ex^i before time h.

Proof. Let h denote the minimum local time (over all executions) at which a node i can distinguish Ex^{i+1} and Ex^{i+2}. By the prerequisites and the assumption that no two messages are received at the same time, there exists an honest message m that i receives in exactly one of these executions at local time $h > h^*$. Observe that the requirements on $\widetilde{\mathsf{Ex}}^1, \widetilde{\mathsf{Ex}}^2, \widetilde{\mathsf{Ex}}^3$ fully specify these executions save for the additional message that is sent by some faulty party. We rule that the additional message is m, which is sent by the same (there faulty) node with the same local reception time at i. We will need to establish that $\widetilde{\mathsf{Ex}}^1, \widetilde{\mathsf{Ex}}^2, \widetilde{\mathsf{Ex}}^3$ are well-defined, in that faulty nodes always learn all signatures required for the messages they send. However, let us first establish the indistinguishability statements, as these are needed to show well-definedness.

We begin by showing that for each $j \in [3]$, executions $\widetilde{\mathsf{Ex}}^{j+1}$ and Ex^{j+1} are indistinguishable before time h. An analogous statement holds for $\widetilde{\mathsf{Ex}}^{j+2}$ and Ex^{j+2}. Toward a contradiction, suppose that there exists a local time $\tilde{h} < h$ at which some node j can distinguish $\widetilde{\mathsf{Ex}}^{j+1}$, Ex^{j+1}. Without loss of generality, we let $\tilde{h}$ denote the minimal such time. This implies that j receives a message $\tilde{m}$ at local time $\tilde{h}$ in execution $\widetilde{\mathsf{Ex}}^{j+1}$, which it does not receive in Ex^{j+1} (or vice versa).

We analyze two cases:

- $\tilde{m}$ is sent by an honest node $j' \neq j$: As $d > 2\tilde{u}/3$, honest parties' local clocks are less than d apart at all times. As messages sent between honest parties have delay d, thus j' must have sent $\tilde{m}$ at some local time less than $\tilde{h}$. Because $\widetilde{\mathsf{Ex}}^{j+1}$ and Ex^{j+1} are indistinguishable before time $\tilde{h}$, j' sends $\tilde{m}$ in both Ex^{j+1} and $\widetilde{\mathsf{Ex}}^{j+1}$. Hence, $\tilde{m}$ is received at the same local time by j in both Ex^{j+1} and $\widetilde{\mathsf{Ex}}^{j+1}$. This is a contradiction.
- $\tilde{m}$ is sent by an faulty node $j' \neq j$: As $\tilde{m}$ is received by j before local time h, $\tilde{m} \neq m$. It follows that j' sends $\tilde{m}$ in both executions. Hence it is received at the same local time by j in both Ex^{j+1} and $\widetilde{\mathsf{Ex}}^{j+1}$. This is a contradiction.

Since Ex^{j+1} and Ex^{j+2} are indistinguishable to j before local time h, it follows that $\widetilde{\mathsf{Ex}}^{j+1}$ and Ex^{j+2} are also indistinguishable before local time h to j. As, analogously, Ex^{j+2} and $\widetilde{\mathsf{Ex}}^{j+2}$ are indistinguishable to j before local time h, $\widetilde{\mathsf{Ex}}^{j+1}$ and $\widetilde{\mathsf{Ex}}^{j+2}$ are also indistinguishable to j before local time h. Finally, by construction and the assumption that only one message is received at any given time, $\widetilde{\mathsf{Ex}}^{j+1}$ and $\widetilde{\mathsf{Ex}}^{j+2}$ are also indistinguishable to j at local time h.

It remains to show that the behaviour of faulty nodes in executions $\widetilde{\mathsf{Ex}}^1, \widetilde{\mathsf{Ex}}^2, \widetilde{\mathsf{Ex}}^3$ is also well-defined. Suppose that faulty node $i \in [3]$ sends $\tilde{m}$ in $\widetilde{\mathsf{Ex}}^i$. Let $\bar{h} \leq h$ be the local time when $\tilde{m}$ is received (by an honest node) in $\widetilde{\mathsf{Ex}}^i$. Denote as $t^i \geq \bar{h} - 2\tilde{u}/3$ the time at which $\tilde{m}$ is received in $\widetilde{\mathsf{Ex}}^i$. By definition, i sends $\tilde{m}$ at time $t^i - d + \tilde{u} \geq \bar{h} - d + \tilde{u}/3$ in $\widetilde{\mathsf{Ex}}^i$. It is sufficient to show that i receives every message m' on which $\tilde{m}$ depends by this time.

Consider such a message m' and suppose that $i+1 \neq i$ is its sender; the case that $i+2$ is the sender is treated analogously. We distinguish two cases based on which node receives $\tilde{m}$.

- $i+2$ receives $\tilde{m}$: Since $H_{i+2}^{i+1}(\bar{h}) \geq \bar{h}$, and i sends $\tilde{m}$ no later than time $\bar{h} - d$ in $\widetilde{\mathsf{Ex}}^{i+1}$. Thus, m' is received by i in $\widetilde{\mathsf{Ex}}^{i+1}$ no later than at time $\bar{h} - d$, which corresponds to local time at most $h^{i+1} := H_i^{i+1}(\bar{h} - d) \leq \bar{h} - d + 2\tilde{u}/3 < \bar{h}$. By indistinguishability of executions before local time $h \geq \bar{h}$, i receives m' by the same local time h^{i+1} in $\widetilde{\mathsf{Ex}}^{i+2}$, and accordingly no later than (real) time h^{i+1}. Hence, $i+1$ sends m' in $\widetilde{\mathsf{Ex}}^{i+2}$ by time $h^{i+1} - d$ and local time $h^{i+2} := H_{i+1}^{i+2}(h^{i+1} - d) \leq h^{i+1} - d + 2\tilde{u}/3 < \bar{h}$. By indistinguishability of $\widetilde{\mathsf{Ex}}^i$ and $\widetilde{\mathsf{Ex}}^{i+2}$ before local time h, it follows that $i+1$ sends m' in $\widetilde{\mathsf{Ex}}^i$ by local time (and also time) h^{i+2}. Hence, in $\widetilde{\mathsf{Ex}}^i$, i receives m' by time
$$h^{i+2} + d - \tilde{u} \leq h^{i+1} - \frac{\tilde{u}}{3} \leq \bar{h} - d + \frac{\tilde{u}}{3}.$$
- $i+1$ receives $\tilde{m}$: Since $H_{i+1}^{i+2}(\bar{h}) \geq \bar{h}$, we have that i sends $\tilde{m}$ by time $\bar{h} - d$ in $\widetilde{\mathsf{Ex}}^{i+2}$. Thus, m' is received by i in $\widetilde{\mathsf{Ex}}^{i+2}$ by time $\bar{h} - d$, and hence sent by $i+1$ by time $\bar{h} - 2d$. This corresponds to local time $h^{i+2} := (H_{i+1}^{i+2})(\bar{h} - 2d) \leq \bar{h} - 2d + 2\tilde{u}/3 < \bar{h}$. By indistinguishability of executions before local time $h \geq \bar{h}$, $i+1$ sends m' by the same local time and time h^{i+2} in $\widetilde{\mathsf{Ex}}^i$. Hence, i receives m' at time at most
$$h^{i+2} + d - \tilde{u} \leq \bar{h} - d - \frac{\tilde{u}}{3} < \bar{h} - d + \frac{\tilde{u}}{3}.$$ □

Using the Lemma 18 inductively, we construct executions that are indistinguishable to honest nodes at all times.

Lemma 19. Let $n = 3$ and let Π be a 1-secure protocol for pulse synchronization with skew S. Then there exist executions $\mathsf{Ex}^1, \mathsf{Ex}^2, \mathsf{Ex}^3$ of Π that satisfy P and where node $i \in [3]$ cannot distinguish Ex_{i+1} and Ex_{i+2}.

Proof Sketch. Inductive application of Lemma 18 yields a series of triples of executions with an increasing number of messages sent by faulty nodes, which cannot be distinguished by correctness up to the largest local time when such a message is received. There are two cases: The induction halts, because the indistinguishability holds at all times; in this case we are done. Otherwise, the constructed executions share indistinguishable prefixes with those that are constructed later. As only finitely many messages are sent in finite time, these prefixes must become arbitrarily long. Hence, they define limit executions satisfying the required properties. □

Equipped with these executions, we are in the position to prove the claimed lower bound.

Theorem 5. Let $n \geq 3$ and Π be an $\lceil n/3 \rceil$-secure protocol for pulse synchronization with skew S. Then $\mathbb{E}[S] \geq 2\tilde{u}/3$.

Proof. We show the statement for the special case of $n = 3, t = 1$. For general n, we can reduce the argument to the case of $n = 3, t = 1$ as follows. Assume the existence of an $\lceil n/3 \rceil$-secure protocol Π for pulse synchronization with skew $S < 2\tilde{u}/3$. Now, partition the set of n nodes into three non-empty subsets S_1, S_2, S_3 of size at most

$\lceil n/3 \rceil$. Then, node $i \in [3]$ simulates the protocol behaviour of nodes in S_i in Π and outputs the pulse times of the lexicographically first node in S_i. By assumption, this yields a 1-secure protocol for pulse synchronization for $n = 3$ with skew S.

Thus, let Π be a 1-secure protocol for pulse synchronization, and assume for now that Π is deterministic. By Lemma 19, there exist executions $\mathsf{Ex}^1, \mathsf{Ex}^2, \mathsf{Ex}^3$ with the following properties:

- $\mathcal{H}^i = [3] \setminus \{i\}$.
- $H^i_{i+1}(t) = t$.
- $H^i_{i+2}(t) = \vartheta \cdot t$ for $t \leq 2\tilde{u}/(3(\vartheta - 1))$ and $H^i_{i+2}(t) = t + 2\tilde{u}/3$ for $t \geq 2\tilde{u}/(3(\vartheta - 1))$.
- Ex_{i+1} and Ex_{i+2} are indistinguishable to node i.

Recall that Π must guarantee some minimum period $P_{\min} > 0$. We define $r := \lceil \tilde{u}/P_{\min}(\vartheta - 1) \rceil + 1$, such that for all $i \in [3]$, $\min\{p^{i,r}_{i+1}, p^{i,r}_{i+2}\} \geq 2\tilde{u}/(3(\vartheta - 1))$, where $p^{i,r}_v$ denotes the rth pulse time of honest node v in execution Ex^i. Recall that for all times t, $H^{i+2}_i(t) = t$ and for times $t \geq p^{i+1,r}_i$, $H^{i+1}_i(t) = t + 2\tilde{u}/3$. By indistinguishability of executions Ex^{i+1} and Ex^{i+2} for node i, $H^{i+1}_i(p^{i+1,r}_i) = H^{i+2}_i(p^{i+2,r}_i)$. Hence,

$$\begin{aligned} p^{i+1,r}_i &= (H^{i+1}_i)^{-1}(H^{i+1}_i(p^{i+1,r}_i)) \\ &= (H^{i+1}_i)^{-1}(H^{i+2}_i(p^{i+2,r}_i)) \\ &= (H^{i+1}_i)^{-1}(p^{i+2,r}_i) = p^{i+2,r}_i - \frac{2\tilde{u}}{3}. \end{aligned}$$

We conclude that

$$\begin{aligned} 3S &\geq (p^{i,r}_{i+1} - p^{i,r}_{i+2}) + (p^{i+1,r}_{i+2} - p^{i+1,r}_i) + (p^{i+2,r}_i - p^{i+2,r}_{i+1}) \\ &= (p^{i+2,r}_i - p^{i+1,r}_i) + (p^{i,r}_{i+1} - p^{i+2,r}_{i+1}) + (p^{i+1,r}_{i+2} - p^{i,r}_{i+2}) \\ &= 2\tilde{u}. \end{aligned}$$

This implies that $S \geq 2\tilde{u}/3$.

It remains cover the case that Π is randomized. To this end, we interpret Π as random variable evaluating to a deterministic protocol (as a result of fixing the randomness of the nodes). Independently of Π, the adversary uniformly at random picks node $i \in [3]$ to corrupt. These choices determine executions Ex^i, $i \in [3]$, as above. The adversary then lets i behave such that Ex^i is realized.

We claim that, while the adversary might not be able to learn the randomness of nodes $i + 1$ and $i + 2$, it is not required to do so. The messages node i sends are those it would send as a correct node in Ex^{i+1} and Ex^{i+2}, respectively. It receives the same messages in Ex^i, just at different times. However, from the reception times it can compute the corresponding local reception times at i in Ex^{i+1} and Ex^{i+2}, based on the known message delays and local times of the respective senders in the respective executions. Hence, it can simply simulate two copies of Π at i with the randomness of i, to which it feeds the received messages with the local times at which they are received in Ex^{i+1} and Ex^{i+2}, respectively. As we have shown that all messages required in the adversary's simulation are received early enough to produce all appropriate messages to be sent in time, we conclude that the adversary can indeed realize Ex^i.

Denote by $p^{i,r}_v(\pi)$ the rth pulse of v in the execution Ex^i of deterministic protocol π. Using independence, we get that

$$\begin{aligned} 3 \cdot \mathbb{E}[S] &= 3 \cdot \sum_{i=1}^{3} P[i \text{ is corrupted}] \cdot \mathbb{E}[S \mid i \text{ is corrupted}] \\ &= \sum_{i=1}^{3} \mathbb{E}[S \mid i \text{ is corrupted}] \\ &\geq \sum_{i=1}^{3} \sum_{\pi} P[\Pi = \pi] \cdot (p^{i,r}_{i+1}(\pi) - p^{i,r}_{i+2}(\pi)) \\ &= \sum_{\pi} P[X = \pi] \cdot \sum_{i=1}^{3} p^{i,r}_{i+1}(\pi) - p^{i,r}_{i+2}(\pi)) \\ &\geq \sum_{\pi} P[X = \pi] \cdot 2\tilde{u} = 2\tilde{u}. \end{aligned}$$

□

REFERENCES

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected $o(1)$ rounds, expected $o(n^2)$ communication, and optimal resilience. In *Financial Cryptography*, pages 320–334, 2019.

[2] Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM (JACM)*, 32(4):804—823, 1985.

[3] Saâd Biaz and Jennifer L. Welch. Closed form bounds for clock synchronization under simple uncertainty assumptions. *Information Processing Letters (IPL)*, 80(3):151—157, 2001.

[4] Danny Dolev. The byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.

[5] Danny Dolev. The byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.

[6] Danny Dolev, Joe Halpern, and H. Raymond Strong. On the possibility and impossibility of achieving clock synchronization. In *ACM Symposium on Theory of Computing (STOC)*, pages 504—511, 1984.

[7] Danny Dolev and Christoph Lenzen. Clock synchronization and adversarial fault tolerance. Online lecture notes, 2021. https://www.mpi-inf.mpg.de/departments/algorithms-complexity/teaching/summer21/clock-synchronization-and-adversarial-fault-tolerance.

[8] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM (JACM)*, 33(3):499—516, 1986.

[9] Danny Dolev and Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[10] Michael Fischer, Nanch Lynch, and Robert Patterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[11] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 59—70, 1985.

[12] Joseph Y. Halpern, Barbara Simons, Ray Strong, and Danny Dolev. Fault-tolerant clock synchronization. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 89—102, 1984.

[13] Pankaj Khanchandani and Christoph Lenzen. Self-stabilizing byzantine clock synchronization with optimal precision. *Theory of Computing Systems (TOCS)*, 63:261–305, 2018.

[14] Leslie Lamport and P. M. Melliar-Smith. Byzantine Clock Synchronization. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 68–74, 1984.

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

[16] Christoph Lenzen and Julian Loss. Optimal Clock Synchronization with Signatures. *CoRR*, 2022.

[17] Jennifer Lundelius and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 75—88, 1984.

[18] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. In *Principles of Distributed Computing (PODC)*, pages 71–86, 1985.