Switching On: How to Boot Clock Synchronization in Partially Synchronous Systems

JOSEF WIDDER*
Technische Universität Wien
Department of Automation
Treitlstrasse 1, A-1040 Vienna (Austria)
widder@auto.tuwien.ac.at

Abstract

We address the problem of network booting: Distributed processes boot at unpredictable times and require to start some distributed algorithm; we consider clock synchronization algorithms in systems of $n \geq 3f+1$ processes where at most f exhibit Byzantine behavior. Obviously, assumptions like "there are always at most one third of the running processes Byzantine faulty" do not hold during system start-up when processes boot one after the other. Another peculiarity of network booting is message loss, even if perfect communication is assumed: A message that is sent by a correct process could be lost because the receiver has not completed booting when the message arrives.

Using a partially synchronous model where upper and lower bounds upon transmission and computation time exist but are unknown, we show that a suitable modification of Srikanth & Toueg's non-authenticated clock synchronization algorithm can handle network booting and guarantees bounded precision both during normal operation and start-up. Accuracy (in the sense of clocks being within a linear envelope of real-time) is only guaranteed, however, if sufficiently many correct processes are eventually up and running.

Keywords: *Fault-tolerant distributed algorithms, initialization, clock synchronization, partially synchronous systems, Byzantine faults*

*This research is part of our W2F-project, which targets a wireline/wireless fieldbus based upon spread-spectrum (CDMA) communications, see *http://www.auto.tuwien.ac.at/Projects/W2F/* for details. W2F is supported by the Austrian START programme Y41-MAT.

1 Introduction

Synchronized clocks are vital to many applications (see [16] for an overview) and should therefore be provided as early as possible during system operation. Algorithms [13, 17, 28] have been proposed that establish initial synchronization, but their assumptions cannot always be guaranteed during system start-up: Often it is assumed that all correct processes are up and listening to the network when the algorithm is started [17, 28]. In systems where processes boot at unpredictable times, this assumption is too strong. We will see that it can be dropped.

The problem of booting with initially down processes has been solved for the special MAFT architecture [13]. There a priori assumptions on message transmission delay and local timers are used to construct a sufficiently large listen window. Termination is achieved by Byzantine Agreement, which, however, requires $2f + 1$ correct running processes that cannot always be guaranteed during start-up. Our goal is minimizing the number of a priori assumptions.

Before we turn our attention to initial synchronization during system start-up, we shortly revisit the problem of clock synchronization that we need to achieve after booting: Every correct process p, which has completed booting and initial synchronization—we call such a process *active*—, maintains a discrete clock $C_p(t)$ that can be read at arbitrary real-times t. It must satisfy the following properties:

(P) **Precision Requirement.** There is some constant *precision* $D_{max} > 0$ such that $|C_p(t) - C_q(t)| \leq D_{max}$ for any two active correct processes p and q and any real-time t.

(A) **Accuracy Requirement.** There are some constants $a, b, c, d > 0$ such that $a(t_2 - t_1) - b \leq C_p(t_2) - C_p(t_1) \leq c(t_2 - t_1) + d$ for any active correct process p and any two real-times $t_2 \geq t_1$.

The *precision requirement* (P) states that the difference of any two correct clocks in the system must always be bounded, whereas the *accuracy requirement* (A) guarantees some relation between the progress of clock-time and progress of real-time; in literature (A) is also called *linear envelope requirement*. It is well known [9] that without authentication[1] no more than one third of the processes may be Byzantine faulty to ensure (P) and (A). Our goal is to handle system initialization without increasing the number of required processes.

Obviously, the problem of startup vanishes if an a priori bound on the period of time required for completing the startup of all correct processes can be guaranteed: An initializing process in a (semi-) synchronous system can simply setup a suitable local timeout before it starts sending its first message. However, many real networks cannot be modeled properly as synchronous systems, and even a single correct process that violates the booting time assumption could cause the initialization to fail. Consequently, a time(r)-free initialization, if available, is preferable.

Let us now take a closer look at the problems evolving in systems of $n \geq 3f + 1$ processes where all correct processes are initially down. Let us assume that there is an initialization algorithm $\mathcal{I}$ that handles initially down correct processes and Byzantine faults. For liveness, $\mathcal{I}$ must eventually initialize all correct processes and bring them into a state where they can update their clocks regularly such that each correct clock always satisfies (P) and (A). $\mathcal{I}$ must of course also handle the case where Byzantine faulty processes are just down and never send messages, hence it must reach initial synchrony when at least $n - f$ correct processes are up. Now consider the case where Byzantine faulty processes behave exactly as correct ones during initialization. It is obvious that $\mathcal{I}$ again initializes the system, but in this case we could have a system of $n_{up} \geq 2f + 1$ running processes with up to f faulty ones among them (more than one third of the processes would be faulty).

A straightforward approach could be based upon determining at runtime when sufficiently many (at least $2f + 1$) correct processes are eventually up. A process would have to wait until it has received messages from $3f + 1$ processes. To guarantee liveness in the case of Byzantine processes not sending any messages, the number of required processes would increase to $4f + 1$. The question is: Could this penalty somehow be avoided?

One possible solution is that the clock synchronization algorithm has *graceful degradation* [19]. This prevents Byzantine processes from corrupting the sys-

[1] We do not consider authenticated algorithms. Besides the disadvantages of computational and communication overhead, it is never guaranteed that malicious processes cannot break the authentication scheme. Using the algorithm of Srikanth and Toueg [28], our correctness proofs do not rely on the probability of the security of authentication services .

tem state if more than one third of the processes are faulty. But even with graceful degradation the implementation is not trivial since it must guarantee that late starters will be synchronized with the earlier started processes such that eventually properties (P) and (A) are achieved.

If, on the other hand, the algorithm has no graceful degradation, Byzantine processes could force the system into an arbitrary state and the solution of initialization must be *self-stabilizing* [7]. Most self-stabilizing clock synchronization algorithms [2,10,21] do not stabilize if some processes remain faulty during the whole execution. Exceptions are the algorithms by Dolev and Welch [11], which stabilize even in the presence of Byzantine faults, but they require synchronous systems (enforced by a common pulse) or semi-synchronous systems (processes are equipped with physical clocks). This does not match with our partially synchronous system model. Stabilizing algorithms, however, cannot guarantee bounded precision during whole system life-time since the transition from illegitimate system states to normal operation cannot always be detected.

Accomplishments: In our paper, we show that the number of required processes need not be increased for clock synchronization[2] in partially synchronous (and hence also synchronous) systems: By modifying the well-known algorithm of Srikanth and Toueg [28], we provide an algorithm that is complete time- and timer-free and requires only $n \geq 3f + 1$ processes even during system startup. It guarantees precision D_{max} during whole system operation, whereas progress of the clocks (linear envelope requirement) can only be guaranteed when sufficiently many correct processes are up and running. Using our clock synchronization service in conjunction with any higher-level distributed algorithm that requires just ε-synchronized clocks hence guarantees safety also during system startup.

Related Work: Clock synchronization in distributed systems is a very well-researched field (see [9, 20,23,24,26,27] for an overview). Still, there are only a few papers [13, 17, 20, 28, 29] known to us that deal with initial synchronization, mostly in the context of integrating a new process in an already running system. Exceptions are solutions for very specific architectures: For the TTP system, the startup—as change from asynchronous to synchronous operation—has been investigated [29]. As we have seen above, initial clock synchronization for the MAFT architecture [13] has been solved, but under stronger system assumptions.

Still we do not know of any approach that could be compared to ours with respect to partial synchrony in conjunction with initially down correct processes.

Much research has been conducted on partially synchronous systems [4, 8, 12, 18]. Clock synchronization is an important issue here as well (see [12, 22]). Our modifications of Srikanth and Toueg's algorithm [28] are in fact based upon ideas from work conducted on consensus algorithms for partially synchronous systems [3, 12]. Still, we do not know of any work that considers system startup. Another system model that is neither completely synchronous nor asynchronous is the Timed Asynchronous Model [6] where processes are equipped with physical clocks, which is not the case in our system model. Further semi-synchronous models (see e.g. [22]) assume that processes know a priori about the timing bounds of the system, which is not the case in partially synchronous systems.

The results of this paper are related to the crash recovery model [1], where processes may crash and recover arbitrarily during the execution of a consensus algorithm. We consider Byzantine faults, though. Similar work was conducted in the context of clock synchronization [5]. We, however, consider more than $N/2$ "crashed" (actually, "initially dead") processes during startup. This exceeds the bounds of the previous work [1, 5]. As mentioned above, there is also some relation of our algorithm to the synchronous approximate agreement algorithms by Mahaney and Schneider [19], which provide graceful degradation when between $1/3$ and $2/3$ of the processes are faulty. We also reach this bound.

Organization of the paper: Section 2 contains our partially synchronous system model as well as some additional notation related to the initialization phase. Our novel algorithm is described in Section 3 and analyzed for degraded mode in Section 4. Advancing from degraded to normal mode is discussed in Section 5. In Section 6 we shortly discuss our algorithms accuracy properties during normal operation.

[2] Although this problem is traditionally studied in systems with known timing behavior where the processes are equipped with local hardware clocks, it can also be solved in partially synchronous systems with software clocks (counters) .

2 System Model

We consider a system of n distributed processes denoted as $p, q, \ldots$, which communicate through a reliable, error-free and fully connected[3] point-to-point network. We assume that a non-faulty receiver of a message knows the sender. The communication channels between processes need not provide FIFO transmission, and there is no authentication service.

Among the n processes there is a maximum of f faulty ones. Since we examine the starting of a network, correct processes that just have not booted yet are not counted as faulty. No assumption is made on the behavior of faulty processes; they may exhibit Byzantine faults [14].

Our system model is partially synchronous [12]. Rather than the global stabilization time model, where it is assumed that the system is synchronous from some unknown point in time on, we use (a slight variant of) the model where bounds upon transmission and computation delays exist but are unknown. Therefore processes have no timing information and can only make decisions based on received messages. In our analysis we will denote by δ the end-to-end computational + transmission delay of a message sent between two correct processes; δ can be different for each message.

We consider a timing model that differs from previous models [12], since we do not only give an upper bound τ^+ for the transmission delay but also a lower bound τ^- such that $0 < \tau^- \leq \delta \leq \tau^+ < \infty$, where τ^- and τ^+ are not known in advance. Since $\tau^+ < \infty$, every message sent from a correct process to another one is eventually received.

The two values ε and Θ describe the timing uncertainty: We will use the *transmission delay uncertainty* $\varepsilon = \tau^+ - \tau^-$ frequently in our analysis. A second, particularly important measure of uncertainty is the *transmission delay ratio* $\Theta = \tau^+/\tau^-$. All our results will solely depend on Θ[4], which has a number of interesting consequences [15], e.g. improved coverage: The bound on Θ may hold even in the case when τ^+ is violated during periods of high network load, when the current lower bound also increases.

[3]Note that even under our perfect communication assumption, messages that reach a process that is not booted are lost. The transfer of our algorithm into a system model with hybrid node and link faults is subject to ongoing work .

[4]In real systems Θ can be adjusted by introducing a constant local timeout z as part of the computational delay. This increases both τ^+ and τ^- and thus decreases Θ. Since z is only used to slow down the algorithm, the system remains partially synchronous.

2.1 Model of the Initialization Phase

At the beginning all correct processes are down, i.e. they do not send or receive messages. Every message that arrives at a correct process while it is down is lost. A correct process decides independently when it wishes to participate in the system (or is just switched on). Faulty processes may be Byzantine, we can hence safely assume that faulty processes are always up or at least booted before the first correct one.

During initialization, correct processes go through the following modes:

1. *down*: A process is down when it has not been started yet or has not completed booting.
2. *up*: A process is up if it has completed booting. To get a clean distinction of up and down we assume that a process flushes the input queues of its network interface as first action after booting is completed. Hence it receives messages only if they have arrived when it was up.
   (a) *passive*: Running processes initially perform an initialization algorithm that does not provide the required service to the application. During this phase they are said to be passive.
   (b) *active*: Processes which have completed their initialization in passive mode and provide the required service (in our case clock synchronization) to the application are active.

Let n_{up} be the number of processes which are up at a given time; n_{up} includes at most f faulty processes.

2.2 Messages

There are only two types of messages sent by our algorithm: *(init, k)* and *(echo, k)*, where k is the senders clock value. We require messages with history: *(echo, k)* implies *(echo, $k-1$)* and *(echo, $k-2$)* messages (the information is transferred implicitly and therefore requires no additional data). This is necessary since processes that start late could have missed previous messages. Going back two rounds with history size is—as we will see—sufficient for our algorithm.

3 The Algorithm

The algorithm given in Figure 1 is an extension of the well known non-authenticated clock synchronization algorithm by Srikanth and Toueg [28]. In fact the first three **if** clauses (`line 4` to `line 16`) are identical to their algorithm.

```
For each correct process

1  VAR k : integer := 0;
2  VAR mode : {passive, active} := passive;
3
4  if received (init, k) from at least f + 1 distinct processes
5     → send (echo, k) to all [once];
6  fi
7
8  if received (echo, k) from at least f + 1 distinct processes
9     → send (echo, k) to all [once];
10 fi
11
12 if received (echo, k) from at least n − f distinct processes
13    → if mode = active → C := k + 1; fi /* update clock */
14      k := k + 1;
15      send (init, k) to all [once]; /* start next round */
16 fi
17
18 /* catch-up rule */
19 if received (echo, l) from at least f + 1 distinct processes
20    with l > k + 1
21    → if mode = active → C := l − 1; fi /* update clock */
22      k := l − 1; /* jump to new round */
23      send (echo, k) to all [once];
24 fi

Additional Code for each passive correct process

25 if received (init, x) from at least f + 1 distinct processes
26    → C := max(x − 1, k);
27      k := max(x − 1, k);
28      mode := active;
29      send (echo, k) to all [once];
30 fi
```

Figure 1. Clock Synchronization Algorithm with Start-up

The algorithm basically implements a nearly simultaneous global event in a system of $n \geq 3f + 1$ processes, which is used to simultaneously increment the clocks at all processes: When a local clock has made its k^{th} tick, the process sends an *(init, k)* message to all. If any correct process receives $f + 1$ *(init, k)* messages it can be sure that at least one was sent by a correct process and it therefore sends *(echo, k)*. When $f+1$ *(echo, k)* messages are received at a process it also sends *(echo, k)*.[5] If a process receives $n - f \geq 2f + 1$ *(echo, k)* messages it can be sure that among those are at least $f + 1$ messages sent by correct processes. These will be echoed by every other correct process so that within bounded time every correct process also receives $n - f$ *(echo, k)* messages—this property is called *relay*. This only works because $n - f$ is the minimum number of correct running processes in the system. Therefore, every process that has received $n - f$ *(echo, k)* messages may safely increment its local clock value to $k + 1$ and send *(init,* $k + 1$*)*.

Srikanth and Toueg have shown [28] that the algorithm achieves (P) and (A) for systems with $n \geq 3f+1$ if they are initially synchronized. They give an algorithm for initialization as well, which, however, does not work in our setting: If $n_{up} \geq n - f$ it could be that with the "help" of faulty processes, correct ones make some progress. Still there are not sufficiently many correct processes up to guarantee relay and hence progress at every correct process; (P) and (A) could be violated. In [28] a solution for integration of late starters is given as well, which relies on progress. Since there is no progress guarantee for our reduced n_{up} there also is no guarantee for integration.

We reach our goal of bounded precision (P) during whole system operation based on the following observation: Progress of clock value at a correct process requires always at least $f + 1$ messages from distinct correct processes. Since correct processes always sent messages to all, every correct process must receive those messages. If a process p receives $f + 1$ *(echo)* messages for a future tick, it can conclude that at least one correct process has such a clock value and p could update its clock. Therefore we extend Srikanth and Toueg's algorithm by (1) the current clock value k in `line 1` (which is not available in the original algorithm [28] since all ticks are observed concurrently) and (2) the *catch-up rule*[6] in `line 19-26` that triggers if $f + 1$ messages for a future tick are received such that p can update its clock value.

To overcome the problem of lost messages due to

[5] Since correct processes send *(init,k)* and *(echo, k)* only if at least one correct clock has made its k^{th} tick (set $C := k$), we summarize them as messages for the k^{th} tick frequently in our discussion.

[6] A similar construct is used in a clock synchronization algorithm for partially synchronous systems in [12]. In [3] it is used in a phase protocol which is part of a consensus algorithm. Booting is not addressed there.

initially down processes we add the following protocol (which is not shown in Figure 1): As first action after getting up, a correct process sends *(echo,* 0*)* to all. If a correct process p receives *(echo,* 0*)* by a process q it resends the last *(echo,* k*)* message it has sent to q.

It is easy to see that the first $f+1$ correct processes are initially synchronized at clock value 0[7], since no correct process can make any progress before at least $f+1$ correct processes are up. Before discussing the behavior of correct late starters we give some useful definitions and lemmas.

Definition 3.1 (Local Clock Value). *$C_p(t)$ denotes the local clock value of a correct process p at real-time t; σ_p^k, where $k \geq 0$, is the sequence of real-times when process p has set its local clock value to $k+1$.*

Definition 3.2 (Maximum Local Clock Value). *$C_{max}(t)$ denotes the maximum of all local clock values of correct processes that are up at real-time t. Further, let $\sigma_{first}^k = \sigma_p^k \leq t$ be the real-time the first correct process p has set its local clock value to $k+1 = C_{max}(t)$.*

The following Lemma 3.3 shows that progress of $C_{max}(t)$ is only possible via the third **if** (`line 12`), which needs at least $f+1$ messages by distinct correct processes. This fact will be heavily used in our proofs.

Lemma 3.3 (3rd if). *In a system of $n \geq 3f+1$ processes, every correct process—executing the algorithm given in Figure 1—that sets its clock to $C_{max}(t)$ by time t must do so by the third* **if** *clause in* `line 12`.

Proof. By contradiction. Let a correct process p set its clock to $k = C_{max}(t)$ at instant t by a catch-up rule (`line 19` or `line 27`). At least one correct process must have sent a message for a tick $l > k$ before t. Since correct processes never send messages for ticks greater their local clock value, at least one had a clock value $l > k$ at instant t. Thus $C_{max}(t) > k$ which provides the contradiction. □

Lemma 3.4 (Minimal Number of Init Messages). *Given an arbitrary point in time t with $l = C_{max}(t)$, let $t' \geq t$ be the instant when C_{max} further increases. For $n \geq 3f+1$, at least $f+1$ correct processes set their clocks to $C_{max}(t)$ by the third* **if** *and therefore send (init, l) before t'.*

Proof. In order for the first correct process to set its clocks to $l+1$, it must have received at least $n - f \geq 2f+1$ *(echo, l)* messages sent by distinct processes. At least $f+1$ of those must originate from correct processes, which must have set their local clock to l by the third **if** (Lemma 3.3) and sent *(init, l)* earlier. □

As we have seen in Lemma 3.4, $f+1$ *(init, k)* messages are sent for every tick k. Correct processes never send *(init, k)* messages for arbitrarily small ticks compared to C_{max}[8]. These two facts are used for changing from passive to active mode. As discussed in the system model in Section 2.1 processes start in passive mode. We have already mentioned that each correct process starts with sending *(echo,* 0*)* to all. Then it just executes the algorithm from Figure 1. When a correct passive process p eventually receives $f+1$ *(init, x)* messages for a tick x (see `line 25`) it can be sure that at least one correct process has sent one. Because k cannot be too far apart from C_{max}, process p can conclude that its clock value is within precision and can hence switch to active. We discuss initialization of late starters in Section 5. But first we give a precision D_{MCB} in the following section.

4 Degraded Mode

In this section we will see that the clocks of all correct early starters are always within D_{MCB} of each other. Note that late starters are guaranteed to eventually reach this precision as well. For now we assume that there is a fixed number $n_{up} \leq n$ of processes which are initially up and the correct ones (at least $n_{up} - f$) have clock value 0. This models exactly our early phase, starting when enough correct processes are up such that progress is possible but not guaranteed. In this section's analysis we assume that there are no late starters (late starters are incorporated in Section 5). In the following Theorem 4.1, we will see that even a reduced number of processes achieves some weak properties that are sufficient—as we will see in Theorem 4.5—to guarantee (P) with some precision D_{MCB}[9].

[7] In real systems where physical clocks (with different values) must be synchronized initially, starting with clock value 0 could be regarded as unsatisfactory. We do not think, however, that this is a major drawback since we consider initialization where the processes are just booted. In general the initial physical clock values cannot be trusted directly after booting, especially if only internal synchronization is considered.

[8] See Appendix A for details

[9] Note that we have no progress guarantee in this phase and hence cannot guarantee (A).

Theorem 4.1 (Weak Synchronization Properties). *For $n \geq 3f+1$ with any n_{up}, where $0 \leq n_{up} \leq n$, the algorithm from Figure 1 achieves:*

P1W Weak Correctness. If at least $f+1$ correct processes set their clocks to k by time t, then every correct process sets its clock at least to $k-1$ by time $t+2\tau^+$.

P2 Unforgeability. If no correct process sets its clock to k by time t, then no correct process sets its clock to $k+1$ by time $t+2\tau^-$ or earlier.

P3W Weak Relay. If a correct process sets its clock to k at time t, then every correct process sets its clock at least to $k-2$ by time $t+\varepsilon$.

Proof. Weak Correctness. If $k = C_{max}(t)$ all $f+1$ correct processes must have set their clocks to k using the third **if** (`line 12`), and therefore have sent *(init, k)* by time t (see Lemma 3.3). These correct processes receive the *(init, k)* messages by time $t+\tau^+$ and therefore send *(echo, k)* to all. All correct processes must receive those $f+1$ *(echo, k)* messages by time $t+2\tau^+$ and therefore set their clocks to $k-1$ by the fourth **if** (`line 19`), if they have not already done so.
If $k < C_{max}(t)$ at least $f+1$ correct processes have set their clocks to k by time $t' < t$ using the third **if** from `line 12` (otherwise no correct process may have set its clock to a value greater than k—see Lemma 3.3) and therefore all correct processes set their clocks by time $t' + 2\tau^+ < t + 2\tau^+$ (see previous paragraph).

Unforgeability. Setting the clock value can be done at correct processes by three rules. The proof for all is by contradiction.
Assume there is a process p that sets its clock to $k+1$ before instant $t+2\tau^-$ by the third **if** (`line 12`). It does so because it has received $n-f \geq 2f+1$ *(echo, k)* messages by distinct processes. There are at least $f+1$ *(echo, k)* messages sent by correct processes among those, which must have been sent before $t+\tau^-$. Correct processes only send *(echo, k)* when they have received $f+1$ *(init, k)* or *(echo, l)* messages for a $l \geq k$ from distinct processes. At least one correct process must have sent a message for tick $l \geq k$ before t. By assumption no tick k messages are sent before t. Since all tick $l > k$ messages are sent after tick k messages by a correct process no such message has been sent by time t, which provides the required contradiction.
Assume that there is a process p that sets its clock to $k+1$ before instant $t+2\tau^-$ using the fourth **if** (`line 19`). Process p does so because it has received $f+1$ *(echo, l)* messages by distinct processes for some $l > k+1$. That is, at least one *(echo, l)* message must have been sent by a correct process q before $t+\tau^-$. Process q has sent it, because it has received at least $f+1$ messages for tick $x \geq l$. At least one of these messages must have been sent by a correct process before time t, since messages for tick $x > k$ are sent by a correct process only after tick k messages. But by P2s assumption no tick k message was sent before t, which again provides the contradiction.
Assume finally that there is a correct process p that sets its clock to $k+1$ before instant $t+2\tau^-$ using the fifth **if** (`line 27`). Process p does so because it has received at least $f+1$ *(init, $k+2$)* messages by time $t+2\tau^-$. At least one of these *(init, $k+2$)* message must have been sent by an correct process q before $t+\tau^-$. Process q has sent it, because it has received at least $n-f \geq 2f+1$ *(echo, $k+1$)* messages by time $t+\tau^-$, such that at least one (in fact $f+1$) correct process must have sent *(echo, $k+1$)* before time t. A correct process never sends any $k+1$ messages before it has send a message for tick k. By assumption no tick k message was sent by time t which again provides the contradiction.

Weak Relay. Assume $k = C_{max}(t)$. A correct process must set its clock to k using the third **if** from `line 12` (recall Lemma 3.3), when it has received at least $n-f \geq 2f+1$ *(echo, $k-1$)* messages. Among those are at least $f+1$ messages sent by distinct correct processes. These messages must be received by all correct processes by time $t+\varepsilon$ and therefore they set their clocks to $k-2$ (using the fourth **if**).
If $k < C_{max}(t)$ then at least one correct process has already set its clock to $k' > k$ at time $t' \leq t$ using the third **if** (`line 12`). We have shown in the previous paragraph that all correct processes must set their clocks to $k'-2$ by time $t'+\varepsilon$ so all correct processes must also set their clocks to $k-2$ by time $t+\varepsilon$. □

Note that P1W and P3W are directed to past ticks: Progress is not guaranteed because the properties only ensure that processes reach prior clock values. Still there are time bounds in P1W, P2 and P3W which are sufficient to satisfy the precision requirement (P), as we will show in Theorem 4.5. We require some preparative lemmas for this purpose. The following simple Lemma 4.2 follows immediately from P2 and is hence true for any algorithm that respects unforgeability. It will be used frequently in our proofs.

Lemma 4.2 (Fastest Progress). *Let p be the first correct process that sets its clock to k at time t. Then no correct process can reach a larger clock value $k' > k$ before $t + 2\tau^-(k' - k)$.*

Proof. By induction on $l = k' - k$. For $l = 1$ Lemma 4.2 is identical to unforgeability and therefore true. Assume that no correct process has set its clock to $k + l$ before $t + 2\tau^- l$ for some l. Thus no processes may set their clocks to $k + l + 1$ before $t + 2\tau^- l + 2\tau^- = t + 2\tau^-(l + 1)$ following unforgeability. Hence Lemma 4.2 is true for $l + 1$ as well. □

In our analysis we will frequently require to bound the increase of C_{max} during a given real-time interval $[t_1, t_2]$. Lemma 4.2 can be applied for this purpose if $t_1 = \sigma^k_{first}$ but not for arbitrary times t_1. As in [25] we will provide a general solution based on the following Definition 4.3.

Definition 4.3 (Synchrony). *Real-time t is in synchrony with $C_{max}(t)$ iff $t = \sigma^k_{first}$ for some real-time σ^k_{first} (for some arbitrary k) as defined in Definition 3.2. Let the indicator function of non-synchrony be defined as*

$$I_{t\neq\sigma} = I_\sigma(t) = \begin{cases} 0 \text{ if } t \text{ is in synchrony with } C_{max}(t), \\ 1 \text{ otherwise}. \end{cases}$$

Lemma 4.4 (Maximum Increase of C_{max} within Time Interval). *Given any two real-times $t_2 \geq t_1$. $C_{max}(t_2) - C_{max}(t_1) \leq \lfloor \frac{t_2 - t_1}{2\tau^-} \rfloor + I_\sigma(t_1)$.*

Proof. Let $k = C_{max}(t_1) - 1$. We have to distinguish the two cases $\sigma^k_{first} = t_1$ and $\sigma^k_{first} < t_1$.

Let $\sigma^k_{first} = t_1$ such that $I_\sigma(t_1) = 0$. From Lemma 4.2 follows that C_{max} may increase every $2\tau^-$ time-units, hence $\lfloor \frac{t_2 - t_1}{2\tau^-} \rfloor$ times before t_2. Since $I_\sigma(t_1) = 0$, Lemma 4.4 is true for this case.

Now let $\sigma^k_{first} < t_1$, such that $I_\sigma(t_1) = 1$, and let the real-time $t' = \sigma^{k+1}_{first} > t_1$. We can now apply Lemma 4.2 starting from time t'. Since $t_2 - t_1 > t_2 - t'$ it follows from Lemma 4.2 that C_{max} cannot increase more often than $\lfloor \frac{t_2 - t_1}{2\tau^-} \rfloor$ times between t' and t_2. At instant t', C_{max} has already increased once such that $C_{max}(t_2) - C_{max}(t_1) \leq \lfloor \frac{t_2 - t_1}{2\tau^-} \rfloor + 1$. Since $I_\sigma(t_1) = 1$ Lemma 4.4 is also true for this case. □

In the following major Theorem 4.5 we give a bound for the precision requirement (P). We assume an instant t such that $t = \sigma^k_p$ for a correct process p. From weak relay we can derive a bound for σ^{k+2}_q for any other correct process q, such that $\sigma^{k+2}_q = \sigma^{k+2}_{first}$. Using Lemma 4.4 we can give a bound for $C_{max}(t)$ and hence bound the achievable precision D_{MCB}.

Theorem 4.5 (Precision in Degraded Mode). *Given a system of $n \geq 3f + 1$ processes with n_{up} processes being up, where $0 \leq n_{up} \leq n$. Let the correct ones among them be initially synchronized to 0. Then the algorithm of Figure 1 satisfies the precision requirement (P) with $D_{MCB} = \lfloor \frac{1}{2}\Theta + \frac{5}{2} \rfloor$.*

Proof. If no correct process advances its clock beyond $k = 2$, precision $D_{MCB} \geq 2$ is automatically maintained since all clocks are initially synchronized to $k = 0$.

Assume that a correct process p has a local clock value $k \geq 0$ within a still unknown precision D_{MCB} with respect to all other correct processes—and therefore also to $C_{max}(t')$—at real-time t'. We now use weak relay (P3W), Definition 4.3 and Lemma 4.4 to reason about D_{MCB} by calculating $C_{max}(t)$ for some time $t > t'$.

Let process p advance its clock to $k + 1$ such that $\sigma^k_p = t > t'$. Since p has not done so before t, no other correct process has set its clock to $k + 3$ before $t - \varepsilon$, following directly from weak relay (P3W), thus $\sigma^{k+2}_{first} \geq t - \varepsilon$.

From Lemma 4.4 follows that $C_{max}(t) \leq \lfloor \frac{t-(t-\varepsilon)}{2\tau^-} \rfloor + I_\sigma(t - \varepsilon) + C_{max}(t - \varepsilon)$. Let us now take a closer look at the term $I_\sigma(t - \varepsilon) + C_{max}(t - \varepsilon)$: If $\sigma^{k+2}_{first} = t - \varepsilon$ and therefore $t - \varepsilon$ is synchronized with C_{max}, $C_{max}(t - \varepsilon) = k + 3$ and $I_\sigma(t - \varepsilon) = 0$ (following Definition 4.3). If on the other hand $\sigma^{k+2}_{first} > t - \varepsilon$, $C_{max}(t - \varepsilon) = k + 2$ and $I_\sigma(t - \varepsilon) = 1$. In both cases $I_\sigma(t - \varepsilon) + C_{max}(t - \varepsilon) = k + 3$ such that $C_{max}(t) \leq \lfloor \frac{\varepsilon}{2\tau^-} \rfloor + k + 3$ thus $C_{max}(t) \leq \lfloor \frac{1}{2}\Theta + \frac{5}{2} \rfloor + k$.

Process p has clock value $C_p(t') = k$ at time $t' < t$ which is by assumption within precision. Since $C_p(t') < C_p(t)$ and $C_{max}(t') \leq C_{max}(t)$, we get a bound for our precision D_{MCB} from the difference $C_{max}(t) - C_p(t') = C_{max}(t) - k \leq \lfloor \frac{1}{2}\Theta + \frac{5}{2} \rfloor$. □

5 Integration

In the previous section, we discussed the behavior of the early starters. We now turn our attention to correct late starters which get up after there was possibly some progress of C_{max}. The bound for the resulting precision is a worst case analysis, where it is assumed that a correct process changes to active mode (and

hence must satisfy precision) right after booting based on the worst possible messages. Due to space restrictions, this section just provides a proof sketch[10]. When considering integration we must of course guarantee that the required clock synchronization conditions (P) and (A) are satisfied after enough correct processes are up. We do so later in this section in Theorem 5.3.

Theorem 5.1 (Precision). *For the described system with $n \geq 3f + 1$, there exists a constant D_{max} such that $\mid C_p(t) - C_q(t) \mid \leq D_{max}$ for all active correct processes p and q at every time t. $D_{max} = \lfloor 2\Theta + \frac{11}{2} \rfloor$.*

Proof Sketch. Since *(init)* messages are used for the change from passive to active mode we must first confirm that no *(init, k)* messages for arbitrarily small k are sent by correct processes, i.e. $C_{max} - k$ is bounded. This is true since every correct process requires $n - f$ *(echo)* messages in order to send *(init)*, such that there must be at least one correct process whose message is used to both increase C_{max} and send *(init, k)*.
For the worst case precision, we must consider a newly started process p that receives $f + 1$ *(init, k)* messages with k as small as possible compared to C_{max}. Then we give a bound on how much C_{max} can increase before p must change to active at time t, with clock value $k - 1$. After time t, process p's clock value becomes and stays better, thus $C_{max}(t) - (k - 1)$ is our bound for precision D_{max}. □

Theorem 5.1 shows that (P) is always maintained. In order to show that our algorithm also satisfies (A) when sufficiently many correct processes are up, we give a bound on the maximum time interval it takes to get progress into the system after the $n - f^{th}$ correct processes got up at time t. Since at time t at least one correct process has a clock value of $C_{max}(t)$, *(init, $C_{max}(t)$)* messages may have been missed by the initializing process. The first tick, for which it is guaranteed that all correct processes receive at least $f + 1$ *(init)* messages, is $C_{max}(t) + 1$. The following Theorem 5.3 gives the latest possible instant when those *(init, $C_{max}(t) + 1$)* messages are received by all correct processes.

Remark The necessity of $n - f$ messages by distinct processes instead of $2f + 1$ is a drawback of our solution that cannot be avoided in partially synchronous systems, however: Assume a system of $n = 5f + 2$, thus $4f + 2$ correct processes. Assume further that $2f + 1$ correct processes start early and reach some arbitrary clock value. Then there could be $2f + 1$ correct late starters which could learn about the early starters after they reached a smaller common clock value. During this interval, (P) could be violated.

Lemma 5.2. *For the algorithm given in Figure 1 for $n \geq 3f + 1$ and $n_{up} \geq n - f$, there are at least $f + 1$ correct processes with local clock values $C_{max}(t)$ or $C_{max}(t) - 1$ at any time t.*

Proof. If no correct process advances its clock beyond 1 the lemma is true. Let p be the first correct process that sets its clock to $C_{max}(t) > 0$ at instant t such that $C_p(t) = C_{max}(t)$. It does so because it has received at least $n - f \geq 2f + 1$ *(echo, $C_p(t) - 1$)* messages, i.e. at least $f + 1$ sent by distinct correct processes. Correct processes never send *(echo)* messages for ticks larger than their local ones. Therefore at least $f + 1$ correct processes must have a clock value of $C_p(t)$ or $C_p(t) - 1$ at time t. □

Theorem 5.3 (Initialization Time). *Let t be the time the $n - f^{th}$ correct process p gets up. By time $t + \Delta_{init}$, at least $n - f$ correct processes are running in active mode, where $\Delta_{init} = 8\tau^+$.*

Proof. Process p sends its first *(echo, 0)* message at time t. If there is already progress in the system, p will be initialized very quickly because it receives the necessary *(init)* messages τ^+ after they were sent.

If the processes in front are not making progress, the clock value of p at time $t + 2\tau^+$ is $C_p(t + 2\tau^+) \geq C_{max}(t) - 2$ and is reached using the catch-up rule (`line 19`) as the answers from the $f + 1$ most advanced processes (recall Lemma 5.2 and the fact that *(echo, k)* messages have a history, as described in Section 2.2) must be received by then. Process p then sends *(echo, $C_{max}(t) - 2$)*, such that by time $t + 3\tau^+$ every running correct process must have received $n - f$ *(echo, $C_{max}(t) - 2$)* messages.

Every running correct process now sets its clock to $C_{max}(t) - 1$ (if it has not yet done so) and sends an *(init, $C_{max}(t) - 1$)* message. It could be that p does not receive $f + 1$ *(init, $C_{max}(t) - 1$)* messages, because there were to many processes that already had the clock value $C_{max}(t) - 1$. So p does not necessarily switch to active mode.

By time $t + 5\tau^+$, however, all running correct processes set their clocks to $C_{max}(t)$ and by time $t + 7\tau^+$

[10] Detailed analysis can be found in Appendix A

to $C_{max}(t)+1$. Therefore p and all other running correct processes receive at least $f+1$ *(init,* $C_{max}(t)+1$*)* messages by time $t+8\tau^+$ or earlier. Then, at least $n-f$ correct processes run in active mode. □

6 Envelope Synchronization

With $n-f$ correct processes being active we can give stronger properties than the ones following from P1W, P2 and P3W in Theorem 4.1. The following envelope condition can be derived from these stronger properties, which assure progress. The analysis has been omitted due to space restrictions[11].

Theorem 6.1 (Envelope Condition). *The described system, with* $n \geq 3f+1$*, where at least* $n-f$ *processes are correct and active, satisfies the following envelope condition*

$$\frac{t_2-t_1}{2\tau^+} - 4 + \frac{1}{\Theta} < C_p(t_2) - C_p(t_1) < \frac{t_2-t_1}{2\tau^-} + D_{max} + 1$$

for every correct active process p *and all times* $t_2 \geq t_1$*.*

Acknowledgments

I am grateful to Ulrich Schmid, my Ph.D. supervisor, for his support and encouragement.

References

[1] M. K. Aguilera, W. Chen, and S. Toueg. Failure detection and consensus in the crash-recovery model. *Distributed Computing*, 13(2):99–125, Apr. 2000.

[2] A. Arora, S. Dolev, and M. G. Gouda. Maintaining digital clocks in step. In S. Toueg, P. G. Spirakis, and L. M. Kirousis, editors, *Distributed Algorithms, 5th International Workshop*, volume 579, pages 71–79, Delphi, Greece, 7–9 1991. Springer-Verlag.

[3] H. Attiya, D. Dolev, and J. Gil. Asynchronous byzantine consensus. *Proceedings of the 3rd ACM Symposium of Distributed Computing*, pages 119–133, Aug. 1984.

[4] H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *Journal of the ACM (JACM)*, 41(1):122–152, 1994.

[5] B. Barak, S. Halevi, A. Herzberg, and D. Naor. Clock synchronization with faults and recoveries (extended abstract). In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 133–142. ACM Press, 2000.

[11] The analysis can be found in Appendix B.

[6] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):642–657, 1999.

[7] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[8] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM (JACM)*, 34(1):77–97, 1987.

[9] D. Dolev, J. Y. Halpern, and H. R. Strong. On the possibility and impossibility of achieving clock synchronization. *Journal of Computer and System Sciences*, 32:230–250, 1986.

[10] S. Dolev and J. L. Welch. Wait-free clock synchronization. In *Proceeding of the 12th Annual ACM Symposium on Principles of Distributed Computing (PODC'93)*, pages 97–108, 1993.

[11] S. Dolev and J. L. Welch. Self-stabilizing clock synchronization in the presence of byzantine faults. In *Proc. of the 2nd Workshop on Self-Stabilizing Systems*, May 1995.

[12] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, Apr. 1988.

[13] R. M. Kieckhafer, C. J. Walter, A. M. Finn, and P. M. Thambidurai. The MAFT architecture for distributed fault tolerance. *IEEE Transactions on Computers*, 37:398–405, Apr. 1988.

[14] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[15] G. L. Lann and U. Schmid. How to implement a timer-free perfect failure detector in partially synchronous systems. Technical Report 183/1-127, Department of Automation, Technische Universität Wien, January 2003. (submitted).

[16] B. Liskov. Practical uses of synchronized clocks in distributed systems. *Distributed Computing*, 6:211–219, 1993.

[17] J. Lundelius-Welch and N. A. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, 1988.

[18] N. Lynch. *Distributed Algorithms*, pages 733–827. Morgan Kaufman, 1996.

[19] S. R. Mahaney and F. B. Schneider. Inexact agreement: Accuracy, precision, and graceful degradation. In *Proceedings 4th ACM Symposium on Principles of Distributed Computing*, pages 237–249, Minaki, Canada, Aug. 1985.

[20] P. S. Miner. Verification of fault-tolerant clock synchronization systems. *NASA Technical Paper 3349*, Nov. 1993.

[21] M. Papatriantafilou and P. Tsigas. Self-stabilizing wait-free clock synchronization. Technical Report CS-R9421, Centrum voor Wiskunde and Informatica, Netherlands, 1994.

[22] S. Ponzio and R. Strong. Semisynchrony and real time. In *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG'92)*, pages 120–135, November 1992.

[23] P. Ramanathan, K. G. Shin, and R. W. Butler. Fault-tolerant clock synchronization in distributed systems. *IEEE Computer*, 23(10):33–42, October 1990.

[24] U. Schmid, editor. *Special Issue on The Challenge of Global Time in Large-Scale Distributed Real-Time Systems*, Real-Time Systems 12(1–3), 1997.

[25] U. Schmid and K. Schossmaier. Interval-based clock synchronization. *Real-Time Systems*, 12(2):173–228, Mar. 1997.

[26] F. B. Schneider. Understanding protocols for byzantine clock synchronization. Technical Report 87-859, Cornell University, Department of Computer Science, Aug. 1987.

[27] B. Simons, J. L. Welch, and N. Lynch. An overview of clock synchronization. In *Fault-Tolerant Distributed Computing*, LNCS 448, pages 84–96, 1990.

[28] T. K. Srikanth and S. Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, July 1987.

[29] W. Steiner and M. Paulitsch. The transition from asynchronous to synchronous system operation: An approach for distributed fault-tolerant systems. *Proceedings of the The 22nd International Conference on Distributed Computing Systems*, July 2002.

Appendix

A Analysis of Integration

This section contains a detailed analysis of the overall precision D_{max}, which holds during whole system operation for all active processes. Precision D_{max} is based upon very little information on the system state, that is only a few messages arriving at a process that just got up: The presented solution for the change from passive to active mode[12] allows a correct passive process p to change to active mode based on only one *(init, k)* message sent by a correct process (and the "help" of faulty processes). Since process p computes its clock value based on k, we get a bound for precision from the maximum difference of $C_{max}(t)$ and k at instant t, being t the time when p changes to active. We start the analysis with the following Lemma A.1, which tells us that there is at least one correct process whose messages are required both for increasing C_{max} and sending *(init, k)* with the smallest possible k.

Lemma A.1. *Any two correct processes p and q that decide to send (init, x) and (init, y), respectively, must use at least one message for this decision sent by a common correct process.*

Proof. Both processes p and q must have received at least $m' = n - f$ *(echo)* messages from distinct processes in order to send *(init)*. Among those could be f messages sent by Byzantine processes such that $m = m' - f$ is the minimal number of messages sent by correct processes. The total number of correct processes in the system is $n - f$. We now show that $2m > n - f$ and that therefore at least one of p and q's *(echo)* messages are sent by the same correct process. Since $n \geq 3f + 1$, we have $2n - 4f - n + f > 0$ as asserted. □

For our worst case analysis, we require some information on what can happen near the special time t_{up} when a late starter p gets up. Of special interest is the instant t_{sync} from when on p is synchronized within D_{MCB}. The following Lemma A.2 shows how often C_{max} could increase from the time t_{up} to t_{sync}.

Lemma A.2 (First Synchronization). *Let p be a late starting correct process that gets up at instant t_{up} and $k = C_{max}(t_{up} - \tau^-) + 1$. Assume that C_{max} increases to $k + 1$ at time $t' = \sigma^k_{first}$. Then $C_p(t) \geq C_{max}(t) - D_{MCB}$ for all times $t \geq t_{sync} = t' + \varepsilon$. Moreover, $C_{max}(t_{sync}) \leq C_{max}(t_{up} - \tau^-) + \lfloor \frac{1}{2}\Theta + \frac{3}{2} \rfloor$.*

Proof. Let q be the first correct process that sets its clock to $k+1 = C_{max}(t_{up}-\tau^-)+2$ at time $t' = \sigma^k_q = \sigma^k_{first}$. It does so because, by time t', it has received at least $n - f \geq 2f + 1$ *(echo, k)* messages. Thus at least $f+1$ correct process have sent *(echo, k)* after $t_{up} - \tau^-$, because all correct processes set their clocks to k after that and do not send messages for greater ticks than their clock value. Therefore p receives all messages sent by correct processes for all ticks $l \geq k$, such that P1W, P2 and P3W can be applied to p starting from time t' (tick $k + 1$) on. Therefore—caused by weak relay (P3W)—process p catches up by time $t_{sync} = t' + \varepsilon$, and $C_p(t) \geq C_{max}(t) - D_{MCB}$ for all times $t \geq t_{sync}$. Hence the first part of Lemma A.2 is true.

By assumption $C_{max}(t') = C_{max}(t_{up} - \tau^-) + 2 = k + 1$ and $t' = \sigma^k_{first}$ such that $I_\sigma(t') = 0$ according to Definition 4.3. From Lemma 4.4, we know that $C_{max}(t' + \varepsilon) \leq C_{max}(t') + \lfloor \frac{t'+\varepsilon-t'}{2\tau^-} \rfloor + I_\sigma(t') = C_{max}(t_{up} - \tau^-) + 2 + \lfloor \frac{1}{2}\Theta - \frac{1}{2} \rfloor = C_{max}(t_{up} - \tau^-) + \lfloor \frac{1}{2}\Theta + \frac{3}{2} \rfloor$. Since $t_{sync} \leq t' + \varepsilon$, the second statement of Lemma A.2 follows. □

Since *(init, k)* messages are used by passive processes to switch to active mode, we must bound the difference of k and C_{max}. In the worst case some correct process p sends the worst possible *(init, k)* message based on messages from Byzantine processes or processes that are passive and therefore may send arbitrarily small *(echo)* messages. Still, one message that p uses must also be sent by a correct processes, whose messages are also used to increase C_{max} by Lemma A.1.

Remark The following two proofs of Lemma A.3 and Theorem 5.1 are very similar. Both investigate the case of a process p that decides upon actions (sending *(init, k)* in Lemma A.3 or changing to active in Theorem 5.1 respectively) after getting up at time t_{up} but before t_{sync}, the time p's clock is within precision D_{MCB}. Before t_{sync}, p's clock value may be worse than the clock values of early starters. Nevertheless process p requires messages from correct processes, and we will show that our algorithm guarantees that the clock value p decides on is bounded in its difference to C_{max}. This is sufficient to ensure a precision D_{max} for late starters when changing to active. We do this by using Lemma A.2, which provides a bound on the increase of

[12] Another solution with better precision but possibly later initialization is subject of ongoing work.

C_{max} in the real-time interval $[t_{up}-\tau^-, t_{sync}]$. We just have to give a bound on $C_{max}(t_{up}-\tau^-)$ with respect to p's clock value k when performing its action. The two proofs differ in the derived value of $C_{max}(t_{up}-\tau^-)$, which depends on what messages are required for the actions of process p, and when they were sent in worst case.

Lemma A.3. *Every correct process p that sends (init, k) at instant t does so with $k \geq C_{max}(t) - \lfloor \Theta + \frac{5}{2} \rfloor$.*

Proof. We have to assume that p sends *(init, k)* on possibly few and bad information on the system state. Still, Lemma A.1 ensures that at least one message p used is sent by another correct process r whose messages contribute to increasing C_{max} as well.

Let t_{up} be the time when process p changes to up. Process p sends the worst possible *(init, k)* message at instant $t \geq t_{up}$ based on an *(echo, $k+1$)* message (which implies—as defined in Section 2.2—also the messages *(echo, $k-1$)* and *(echo, k)*) sent by a correct process r that is received at p at time $t_1 \geq t_{up}$. We assume r to be the common process whose message was used to increase C_{max} at instant $t_2 = \sigma_{first}^{k+1} \geq t_1 - \varepsilon$.

The worst case is that r has sent this messages as far as possible in the past, such that C_{max} could have increased as much as possible before p got up (in fact the interesting instant is $t_{up} - \tau^-$ since messages sent after this time must be received by p; faults excluded). Thus we must assume $t_1 = t_{up}$ and $t_2 = t_{up} - \varepsilon$. Since $t_2 = \sigma_{first}^{k+1}$, t_2 is synchronized with C_{max} and hence $I_\sigma(t_2) = 0$ according to Definition 4.3. It follows from Lemma 4.4 that $C_{max}(t_{up}-\tau^-) \leq C_{max}(t_2) + \lfloor \frac{(t_{up}-\tau^-)-(t_2)}{2\tau^-} \rfloor + I_\sigma(t_2) = k + 2 + \lfloor \frac{\varepsilon-\tau^-}{2\tau^-} \rfloor = \lfloor \frac{1}{2}\Theta + 1 \rfloor + k$.

Now we have a bound for $C_{max}(t_{up}-\tau^-)$. For our worst case setting, we must further assume that p sends *(init, k)* as late as possible, before it will be forced to catch up at time t_{sync}, while C_{max} increases at maximum speed. Lemma A.2 provides us with a bound for $C_{max}(t_{sync})$ with respect to $C_{max}(t_{up}-\tau^-)$. We must just consider that p sends *(init, k)*—based on messages with malign origin or sent by passive processes—shortly before t_{sync}. From Lemma A.2, it follows that $C_{max}(t_{sync}) \leq C_{max}(t_{up}-\tau^-) + \lfloor \frac{1}{2}\Theta + \frac{3}{2} \rfloor \leq \lfloor \frac{1}{2}\Theta + 1 \rfloor + k + \lfloor \frac{1}{2}\Theta + \frac{3}{2} \rfloor = \lfloor \Theta + \frac{5}{2} \rfloor + k$ such that $k \geq C_{max}(t_{sync}) - \lfloor \Theta + \frac{5}{2} \rfloor$ as asserted. □

Since we succeeded with bounding the quality of *(init)* messages (which are used to change from passive to active mode), we can now go on with finding a bound for the precision D_{max}. In the following proof of Theorem 5.1, we start with giving a bound for the first clock value a correct process sets in relation to $C_{max}(t)$: We assume that a process sends a worst possible *(init, k)* message to the initializing correct process p. Byzantine processes then send *(init, k)* shortly before p receives other messages from good ones (which would lead to a better clock value). Hence changing to active mode could be based on only one message by a correct process. Still, we can bound the quality of this first clock value. Since process p catches up immediately thereafter, we have found the worst-case precision bound.

Proof of Theorem 5.1. We have to assume that p changes to active on possibly few and bad information on the system state. Still Lemma A.3 provides us with the worst possible *(init, k)* message correct processes may send with respect to C_{max}.

Let t_{up} be the time when process p gets up. Process p changes to active based on at least one *(init, k)* message sent by a correct process r that is received at p at time $t_1 \geq t_{up}$.

The worst case is that r has sent this messages as far as possible in the past at instant t_2, such that C_{max} could have increased as much as possible before p got up (in fact the interesting instant is $t_{up} - \tau^-$ since messages sent after this time must be received by p; faults excluded). Thus we must assume $t_1 = t_{up}$ and $t_2 = t_{up} - \tau^+$. From Lemma 4.4 follows that $C_{max}(t_{up} - \tau^-) \leq C_{max}(t_2) + \lfloor \frac{(t_{up}-\tau^-)-(t_2)}{2\tau^-} \rfloor + I_\sigma(t_2)$. Since t_2 is not synchronized with C_{max}, $I_\sigma(t_2) = 1$ by Definition 4.3. A bound for $C_{max}(t_2)$ with respect to k is given by Lemma A.3, which states that $C_{max}(t_2) \leq k + \lfloor \Theta + \frac{5}{2} \rfloor$. Thus $C_{max}(t_{up}-\tau^-) \leq k + \lfloor \Theta + \frac{5}{2} \rfloor + \lfloor \frac{\varepsilon}{2\tau^-} \rfloor + 1 = k + \lfloor \frac{3}{2}\Theta + 3 \rfloor$.

Now we have a bound for $C_{max}(t_{up}-\tau^-)$. We must further assume that p changes to up as late as possible before it must catch up at time t_{sync}. Lemma A.2 provides us with a bound for $C_{max}(t_{sync})$ with respect to $C_{max}(t_{up} - \tau^-)$. We must just consider that p changes to active—based on messages with malign origin—at time t_{act} shortly before t_{sync} in order to proof our lemma. From Lemma A.2 follows that $C_{max}(t_{sync}) \leq C_{max}(t_{up} - \tau^-) + \lfloor \frac{1}{2}\Theta + \frac{3}{2} \rfloor \leq k + \lfloor \frac{3}{2}\Theta + 3 \rfloor + \lfloor \frac{1}{2}\Theta + \frac{3}{2} \rfloor = \lfloor 2\Theta + \frac{9}{2} \rfloor + k$ such that $C_{max}(t_{act}) \leq C_{max}(t_{sync}) \leq k + \lfloor 2\Theta + \frac{9}{2} \rfloor$. Since $C_p(t_{act}) = k - 1$, the value of D_{max} given in our theorem follows. □

B Analysis of Envelope Synchronization

We now consider a system with $n \geq 3f + 1$, where at least $n - f$ correct processes are active. The following two properties from Theorem B.1 are required in order to establish the lower bounds from Lemma B.4. Note that the properties P1W, P2 and P3W from Theorem 4.1 are implied by P1F and P3.

Theorem B.1 (Clock Synchronization Properties). *The described system with $n \geq 3f + 1$, where at least $n - f$ processes are correct and active, satisfies the following properties:*

P1F Full Correctness. If all correct processes set their clocks to k by time t, then every correct process sets its clock at least to $k + 1$ by time $t + 2\tau^+$.

P3 Relay. If a correct process sets its clock to k at time t, then every correct process sets its clock at least to k by time $t + 2\tau^+ + \varepsilon$.

Proof. Full Correctness. If $k = C_{max}(t)$ all correct processes must have sent *(init, k)* by time t, which will be received by every correct process by time $t + \tau^+$. Because all correct processes have clock values k by time $t + \tau^+$ by assumption of full correctness, they all send *(echo, k)*. These messages will be received by time $t + 2\tau^+$ by every correct process, which then sets its clock to $k + 1$.
If $k < C_{max}(t)$ at least $f + 1$ correct processes must have sent *(echo, k)* by time $t - \tau^-$. All correct processes set their clocks to k by time t (by assumption). On the reception of the $f + 1$ *(echo, k)* messages by time $t + \varepsilon$ all correct processes send *(echo, k)*. These messages are received by all correct processes by time $t + \tau^+ + \varepsilon$, which then set their clocks to $k + 1$.

Relay. Assume $k = C_{max}(t)$. From Weak Relay (see Theorem 4.1) follows that every correct process sets its clock to $k - 2$ by time $t + \varepsilon$ caused by $f + 1$ *(echo, $k - 1$)* messages sent by correct processes. These messages force all correct processes to send *(echo, $k-2$)* by time $t+\varepsilon$ which must be received by time $t + \tau^+ + \varepsilon$, by all correct processes which set their clocks to $k-1$. The $f+1$ *(echo, $k-1$)* messages now force all correct processes to send *(echo, $k - 1$)* such that by time $t + 2\tau^+ + \varepsilon$ all correct processes set their clocks to k.
If $k < C_{max}(t)$ at least one correct process has already set its clock to $l > k$ at time $t' < t$. In the previous paragraph we have seen that all correct processes must set their clocks to l by time $t' + 2\tau^+ + \varepsilon$ so all correct processes must also set their clocks to k by time $t + 2\tau^+ + \varepsilon$. □

The following two preparative lemmas are building blocks for the lower envelope bound from Lemma B.4.

Lemma B.2. *For the described system with $n \geq 3f + 1$, where at least $n-f$ processes are correct and active, let p be an active correct process that sets its clock to k at instant t. Then, p sets its clock to $k + 1$ by time $t + 4\tau^+ + \varepsilon$.*

Proof. The proof uses the properties P1F and P3 from Theorem B.1. P3 (relay) guarantees that all correct processes set their local clocks to k by time $t' = t + 2\tau^+ + \varepsilon$. P1F (full correctness) guarantees that all correct processes, including p, must set their clocks to $k + 1$ by time $t' + 2\tau^+$. Hence p sets its clock by time $t + 4\tau^+ + \varepsilon$ □

Lemma B.3 (Slowest Progress). *Let p be the last correct process that sets its clock to k at time t. Then no correct process can have a smaller clock value than $k' > k$ at time $t + 2\tau^+(k' - k)$.*

Proof. By induction on $l = k' - k$. For $l = 1$ Lemma B.3 is identical to full correctness and therefore true. Assume that the last correct process sets its clock to $k + l$ at time $t + 2\tau^+ l$ for some l. Every correct process must set its clock to $l + 1$ by time $t + 2\tau^+ l + 2\tau^+ = t + 2\tau^+(l + 1)$ according to full correctness. Hence Lemma B.3 is true for $l + 1$ as well. □

Lemma B.4 (Lower Envelope Bound). *For the described system with $n \geq 3f + 1$ where at least $n - f$ processes are correct and active, $\frac{t_2-t_1}{2\tau^+} - 4 + \frac{1}{\Theta} < C_p(t_2) - C_p(t_1)$ for every correct active process p and all times $t_2 \geq t_1$.*

Proof. Let $C_p(t_1) = k + 1$ and $C_p(t_2) = k + l + 1$. p has set its clock to $k + 1$ at instant $\sigma_p^k \leq t_1$ and to $k + l + 1$ at instant $\sigma_p^{k+l} \leq t_2$. Hence $t_2 - t_1 < \sigma_p^{k+l+1} - \sigma_p^k \leq \sigma_p^{k+l} - \sigma_p^k + 4\tau^+ + \varepsilon$ according to Lemma B.2. Thus $t_2 - t_1 - 4\tau^+ - \varepsilon < \sigma_p^{k+l} - \sigma_p^k$.

Using relay in conjunction with Lemma B.3 it follows that $\sigma_p^{k+l} - \sigma_p^k \leq 2l\tau^+ + 2\tau^+ + \varepsilon$.

Bringing our bounds on $\sigma_p^{k+l} - \sigma_p^k$ together we get $t_2 - t_1 - 4\tau^+ - \varepsilon < 2l\tau^+ + 2\tau^+ + \varepsilon$. Since $l =$

$C_p(t_2) - C_p(t_1)$ our lower envelope bound is $\frac{t_2-t_1}{2\tau^+} - 4 + \frac{1}{\Theta} < C_p(t_2) - C_p(t_1)$. □

A straightforward approach for giving the upper bound for the envelope condition would be to give the shortest possible interval between clock updates as in Lemma 4.2. Lemma 4.2, however, only refers to C_{max}. Due to the catch-up rule, no shortest possible interval between consecutive clock updates can be given for clocks that are behind. In Lemma B.5, we use the precision of the clocks in conjunction with Lemma 4.2 to get an upper envelope bound.

Lemma B.5 (Upper Envelope Bound). *For the described system with $n \geq 3f + 1$ where at least $n - f$ processes are correct and active, $C_p(t_2) - C_p(t_1) < \frac{t_2-t_1}{2\tau^-} + D_{max} + 1$ for every correct active process p and all times $t_2 \geq t_1$.*

Proof. From Theorem 5.1 (precision) follows that $C_{max}(t) - D_{max} \leq C_p(t) \leq C_{max}(t)$ at all times t for all correct processes p. Specifically, at instant t_1 the clock value of any correct process p is bounded by $C_{max}(t_1) - D_{max} \leq C_p(t_1)$, and at instant t_2 there is an upper bound of $C_p(t_2) \leq C_{max}(t_2)$. Thus $C_p(t_2) - C_p(t_1) \leq C_{max}(t_2) - C_{max}(t_1) + D_{max}$.

$C_{max}(t_2) - C_{max}(t_1)$ needs to be bounded: Let $C_{max}(t_1) = k + 1$ and $C_{max}(t_2) = k + l + 1$. C_{max} is set by any correct process to k at instant $\sigma^k_{first} \leq t_1$ and to $k + l$ at instant $\sigma^{k+l}_{first} \leq t_2$. Hence $t_2 - t_1 > \sigma^{k+l}_{first} - \sigma^{k+1}_{first} \geq \sigma^{k+l}_{first} - \sigma^k_{first} - 2\tau^-$ (by unforgeability), such that $t_2 - t_1 + 2\tau^- > \sigma^{k+l}_{first} - \sigma^k_{first}$.

$C_{max}(t_1) = C_{max}(\sigma^k_{first})$ and $C_{max}(t_2) = C_{max}(\sigma^{k+l}_{first})$ such that the number of steps $l = C_{max}(\sigma^{k+l}_{first}) - C_{max}(\sigma^k_{first})$. l is always smaller than the largest possible number of steps within any time interval of the size $t_2 - t_1 + 2\tau^-$. Using Lemma 4.4 we get $C_{max}(t_2) - C_{max}(t_1) < \frac{t_2-t_1}{2\tau^-} + 1$. Thus $C_p(t_2) - C_p(t_1) < \frac{t_2-t_1}{2\tau^-} + D_{max} + 1$. □

Proof of Theorem 6.1. See Lemma B.4 and Lemma B.5. □