An extended abstract of this report appeared in the proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC 2000), July 2000. Updated versions of this report can be found on http://www.hrl.il.ibm.com/Proactive/papers.html

# Clock Synchronization with Faults and Recoveries

Boaz Barak*
The Weizmann Institute of Science
boaz@wisdom.weizmann.ac.il

Shai Halevi
IBM Watson Research Center
shaih@watson.ibm.com

Amir Herzberg
IBM Haifa Research Lab
amir@il.ibm.com

Dalit Naor*
IBM Almaden Research Center
dalit@almaden.ibm.com

July 13, 2000

**Abstract**

We present a convergence-function based clock synchronization algorithm, which is simple, efficient and fault-tolerant. The algorithm is tolerant of failures and allows recoveries, as long as less than a third of the processors are faulty 'at the same time'. Arbitrary (Byzantine) faults are tolerated, without requiring awareness of failure or recovery. In contrast, previous clock synchronization algorithms limited the total number of faults throughout the execution (which is not realistic) or assumed fault detection.

The use of our algorithm ensures secure and reliable time services, a requirement of many distributed systems and algorithms. In particular, secure time is a fundamental assumption of *proactive secure* mechanisms, which are also designed to allow recovery from (arbitrary) faults. Therefore, our work is crucial to realize these mechanisms securely.

**Keywords:** Clock synchronization, Mobile adversary, Proactive systems

Work was done while at the IBM Haifa Research Lab.

# 1 Introduction

Accurate and synchronized clocks are extremely useful to coordinate activities between cooperating processors, and therefore essential to many distributed algorithms and systems. Although computers usually contain some hardware-based clock, most of these are imprecise and have some drift, as highly precise clocks are expensive and cumbersome. Furthermore, hardware-based clocks are prone to faults and/or malicious resetting. Hence, a clock synchronization algorithm is needed, that lets processors adjust their clocks to overcome the effects of drifts and failures. Such algorithm maintains in each processor a logical clock, based on the local physical clock and on messages exchanged with other processors. The algorithm must deal with communication delay uncertainties, clock imprecision and drift, as well as link and processor faults. In many systems, the main need for clock synchronization is to deal with faults rather than with drift, as drift rates are quite small (some works e.g. [11, 12] actually ignore drifts).

It should be noted that clock synchronization is an on-going task that never terminates. In a long-lived system, it may not be realistic to assume a limit on the total number of faults during the system's lifetime. On the contrary, it seems that the only reasonable assumption to make is that every processor will be faulty at some time or another. Hence it is crucial that a clock synchronization protocol allows processors to *recover* from faults and re-synchronize their clocks. The contribution of this work is a protocol that supports synchronization and recovery, as long as 'not too many processors are faulty at once'. Our protocol withstands arbitrary (or Byzantine) faults, where affected processors may deviate from their specified algorithm in an arbitrary manner, potentially cooperating maliciously to disrupt the goal of the algorithm or system.

A synchronization protocol that allows recovery and withstands Byzantine faults is a much needed tool in the design of secure systems, which are supposed to resist malicious attacks. In many systems, relying on synchronized clocks without an appropriate synchronization protocol may become a weak spot exploited by an attacker who maliciously changes clocks. For this reason, several systems try not to rely on synchronized clocks at all (e.g. use instead freshness in authentication protocols, such as in Kerberos [23]). However, there are systems that must rely on synchronized clocks for efficiency or functionality, and some tasks require synchronized clocks by their very definition. For example time-stamping [15] and e-commerce applications such as payments and bids with expiration dates. Therefore, secure time services are an integral part of secure systems such as DCE [26], and there is on-going work to standardize a secure version of the Internet's Network Time Protocol in the IETF [29].[1]

The original motivation for this work came from the need to implement secure clock synchronization for a *proactive security* toolkit [1]: Proactive security allows arbitrary faults in any processor - as long as no more than $f$ processors are faulty during any (fixed length) period. Namely, proactive security makes use of processors which were faulty and later recovered. Since malicious attackers can avoid detection in some settings, the proactive security framework calls for solutions that work even when there is no indication that a processor failed or recovered. To achieve that, existing algorithms for proactive security periodically perform some 'corrective/maintenance' action. For example, they may replace secret keys which may have been exposed to the attacker. Clearly, the security and reliability of such periodical protocols depend on securely synchronized clocks, to ensure that the maintenance protocols are indeed performed periodically. There is substantial amount of research on proactive security, including basic services such as agreement [25], secret

[1] It should be noted that existing 'secure time' protocols simply authenticate clock synchronization messages. Such protocols do not withstand a malicious clock-resetting attacks, even if the authentication itself is secure.

sharing [24, 18], signatures e.g. [17] and pseudo-randomness [4, 5]; see survey in [3]. All these results assumes synchronous clocks, and so they all must rely on a secure clock synchronization mechanism. The current work therefore provides a missing foundation to these and future proactive security works.

### 1.1 Relations to prior work

There is a very large body of research on clock synchronization, much of it focusing on fault-tolerance. Below we focus on the most relevant works.

A number of works focus on handling processor faults, but ignore drifts. Dolev and Welch [11, 12] analyzed clock synchronization under a hybrid faults model, with recovery from arbitrary initial state of all processors (self stabilization) as well as napping (stop) failures in any number of processors in [11] or Byzantine faults in up to third of the processors in [12]. Both works assume a synchronous model, and synchronize logical clocks - the goal is that all clocks will have the same number at each pulse.

The model of time-adaptive self stabilization as suggested by Kutten and Patt-Shamir [19] is closer to ours; there, the goal is to recover from arbitrary faults at $f$ processors in time which is a function of $f$. We notice this is a weaker model than ours in the sense that it assumes periods of no faults. A time-adaptive, self-stabilizing clock synchronization protocol, under asynchronous model, was presented by Herman [16]. This protocol is not comparable to ours as it does not allow drifts and does not synchronize to real time.

Among the works dealing with both processor faults and drifts, most assume that once a processor failed, it never recovers, and that there is a bound $f$ on the number of failed processors throughout the lifetime of the system. Many such works are based on local convergence functions. An early overview of this approach can be found in Schneider's report [27]. A very partial list of results along this line includes [13, 7, 8, 9, 22, 2, 21]. The Network Time Protocol, designed by Mills [22], may allow processors to recover, but no formal analysis or proof of security is known for that protocol. Other work on 'secure clock synchronization' concentrate on authentication of the protocol messages. For example, some authenticated versions of [22] were proposed. However, these proposals do not attempt recovery from malicious faults at the processors themselves.

Our algorithm uses a convergence function similar to that of Fetzer and Cristian [9] (which, in turn, is a refinement of that of Welch and Lynch [21]). However, it seems that one of the design goals of the solution in [9] is incompatible with processor recoveries. Specifically, [9] try to minimize the change made to the clocks in each synchronization operation. Using such small correction may delay the recovery of a processor with a clock very far from the correct one (with [9] such recovery may never complete). This problem accounts for much of the difference between our convergence function and the one in [9]. In the choice between small maximum correction value, and fast recovery time, we chose the latter.

Another aspect in which [9] is optimal, is the maximum logical drift (see Definition 3 in Section 2.3). In their solution, the logical drift equals the hardware drift, whereas our solution there has some small additional factors. We conjecture that strictly 'optimal' logical drift can not be achieved in the mobile faults model.

Another difference between our algorithm and several traditional convergence function based clock synchronization algorithms, is that many such solutions proceed in *rounds*, where correct processes approximately agree on the time when a round starts. At the end of each round, every processor reads all the clocks and adjusts its own clock accordingly. In contrast to this, our protocol (and also NTP [22]) do not proceed in rounds. We believe that implementing a round-based synchronization protocol across a large network (such as the Internet) could be difficult.

A previous work to address faults and recoveries for clock synchronization is due to Dolev, Halpern, Simons and Strong [10]. In that work it is assumed that faults are *detected*. In practice, faults are often undetected - especially malicious faults, where the attacker makes every effort to avoid detection of attack. Handling undetected faults and recoveries is critical for (proactive) security, and is not trivial, as a recovering processor may have its clock set to a value 'just a bit' outside the permitted range. The solution in [10] rely on signatures rather than authenticated links, and therefore also limit the power of the attacker by assuming it cannot collect too many 'bad' signatures (assumption A4 in [10]).

The algorithm of Dolev *et al.* [10] is based on broadcast, and require that all processors sign and forward messages from all other processors. This has several practical disadvantages compared to local convergence function based algorithms such as in the present paper. Some of these disadvantages, which mostly result from its 'global' nature, are discussed by Fetzer and Cristian in [13]. Additional practical disadvantages of broadcast-based algorithms include sensitivity to transient delays, inability to take advantage of realistic knowledge regarding delays, and the overhead and delay resulting from depending on broadcasts reaching the entire network. On the other hand, being a broadcast based algorithm , Dolev *et al.* [10] require only a majority of the processors to be correct (we need two thirds). Also [10] only requires that the subnet of non faulty processors be connected, rather than demanding a direct link between any two processors. (But implementing the broadcast used by [10] has substantial overhead and requires two-third of processors to be correct and connected.)

### 1.2 Clock Synchronization using GPS

In some settings, some or all the processors in the network may have access to an external source of timing signal which they can use to synchronize their clocks. Specifically, the Global Positioning System (GPS) also provides time data. As prices for GPS receivers go down (and since the U.S. government has turned off the intentional degradation of the standard positioning service), this seems to offer an attractive alternative to clock synchronization protocols.

However, there are several reasons why GPS alone cannot be used as an alternative to clock synchronization protocols. The main reason is that the GPS signal can be easily spoofed by radio transmitters. In fact, spoofers for the civilian signal (C/A code) are actually available and used by companies and laboratories in building and testing GPS receivers. In addition, if a GPS receiver is connected to a computer which broadcasts its reading to the network, then we have to consider the possibility of this computer being broken into. Finally, there may also be inaccuracies in the reading because of radio interference or malfunctions in the satellites themselves.

Therefore, the presence of GPS by itself cannot replace the need for a clock synchronization protocol. Instead, one can view GPS as an additional tool that can be used in the design of better clock synchronization protocols. Such protocol were considered in the literature under the name *external clock synchronization*. In particular, Fetzer and Cristian [14] describe a protocol for a network consisting of reference and non-reference time servers (i.e. those with and without access to GPS, respectively). In their model, the non-reference server are not required to recover from faults, while the reference servers may recover.

### 1.3 Informal Statement of the Requirements

A clock synchronization algorithm which handles faults and recoveries should satisfy:

SYNCHRONIZATION Guarantee that at all times, the clock values of the non-faulty processors are close to each other.[2]

ACCURACY Guarantee that the clock rates of non-faulty processors are close to that of the real-time clock. One reason for this requirement is that in practice, the set of processors is not an island and will sometimes need to communicate and coordinate with processors from the "outside world".

RECOVERY Guarantee that once a processor is no longer faulty, this processor recovers the correct clock value and rejoins the "good processors" within a fixed amount of time.

We present a formalization of this model and goals, and a simple algorithm which satisfies these requirements. We analyze our algorithm in a model where an attacker can temporarily corrupt processors, but not communication links. We also briefly describe how this analysis can be generalized to an adversarial model, where the adversary is also capable of corrupting a few communication links (as long as a strong enough connectivity is maintained in the network).

## 2 Formal Model

### 2.1 Network and Clocks

Our network model is a communication graph of $n$ processors, where each processor has its own local clock. We denote the processor names by $1, 2, \ldots n$ and assume that each processor knows its name and the names of all its neighbors. In addition to the processors, the network model also contains an adversary, who may occasionally corrupt processors in the network for a limited time. We denote by $N(p)$ the set of neighbors in the communication graph of processor $p$. We assume the following *connectivity assumption*:

$$\forall_{p,q} |N(p) \cap N(q)| \geq 3f + 1 \tag{1}$$

for some parameter $f$. (This parameter denotes the number of faulty processor, see Definition 2).

Throughout the discussion below we assume some bound $\rho$ on the clock drift between neighboring "good processors", and a bound $\lambda$ on the time that it takes to send a message between two neighboring good processors. We refer to $\rho$ as the *drift bound* and to $\lambda$ as the *message delivery bound*.

We envision the network in an environment with *real time*. A convenient way of thinking about the real time is as just another clock, which also ticks more or less at the same rate as the processors' clocks. For the purpose of analysis, it is convenient to view the local clock of a processor $p$ as consisting of two components. One is an unresettable *hardware clock* $H_p$, and the other is an *adjustment variable* $\mathsf{adj}_p$, which can be reset by the processor. The clock value of $p$ at real time $\tau$, denoted $C_p(\tau)$ is the sum of its hardware clock and adjustment factor at this time, $C_p(\tau) =$

[2]Note that a trivial solution of setting all local clocks to a constant value achieves the synchronization goal. The accuracy requirement prevents this from happening.

$H_P(\tau) + \mathsf{adj}_p$ (these are the same notations as in [27]).[3] We stress that $H_p$ and $\mathsf{adj}_p$ are merely a mathematical convenience, and that the processors (and adversary) do not really have access to their values. Formally, the only operations that processor $p$ can perform on $H_p$ and $\mathsf{adj}_p$ are reading the value $H_p(\tau) + \mathsf{adj}_p$ and adding an arbitrary factor to $\mathsf{adj}_p$. Other than these changes, the value of $H_p$ changes continuously with $\tau$ (and the value of $\mathsf{adj}_p$ remains fixed).

**Definition 1 (Clocks)** *The* hardware clock *of a processor p is a smooth, monotonically increasing function, denoted $H_p(\tau)$. The* adjustment factor *of p is a discrete function* $\mathsf{adj}_p(\tau)$ *(which only changes when p adds a value to the its adjustment variable). The* local clock *of p is defined as*

$$C_p(\tau) \stackrel{def}{=} H_p(\tau) + \mathsf{adj}_p(\tau) \tag{2}$$

We assume an upper bound $\rho$ on the drift rate between processors' hardware clocks and the real time. Namely, for any $\tau_1 < \tau_2$, and for every processor $p$ in the network, it holds that

$$(\tau_2 - \tau_1)/(1+\rho) \ \leq \ H_p(\tau_2) - H_p(\tau_1) \ \leq \ (\tau_2 - \tau_1) \cdot (1+\rho) \tag{3}$$

We note that in practice, $\rho$ is usually fairly small (on the order of $10^{-6}$).

### 2.2 Adversary Model

As we said above, our network model comes with an *adversary*, who can occasionally *break into* a processor, resetting its local clock to an arbitrary value. After a while, the adversary may *leave* that processor, and then we would like this processor to *recover* its clock value.

We envision an adversary who can see (but not modify) all the communication in the network, and can also break into processors and leave them at wish. When breaking into a processor $p$, the adversary learns the current internal state of that processor. Furthermore, from this point and until it leaves $p$, the adversary may send messages for $p$, and may also modify the internal state of $p$, including its adjustment variable $\mathsf{adj}_p$. Once the adversary leaves a processor $p$, it has no more access to $p$'s internal state. We say that $p$ is *faulty* (or *controlled* by the adversary), if the adversary broke into $p$ and did not leave it yet.

We assume reliable and authenticated communication between neighboring processors $p$ and $q$ that are not faulty. More precisely, let $\lambda$ denote the message delivery bound. Then for any neighboring processors $p$ and $q$ not faulty during $[\tau, \tau+\lambda]$, if $p$ sends a message to $q$ at time $\tau$, then q receives exactly the same message from $p$ during $[\tau, \tau+\lambda]$. Furthermore, if a non-faulty processor $q$ receives a message from processor $p$ at time $\tau$, then either $p$ has sent exactly this message to $q$ during $[\tau - \lambda, \tau]$, or else it was faulty at some time during this interval.[4]

The power of the adversary in this model is measured by the number of processors that it can control within a time interval of a certain length. This limitation is reasonable because otherwise, even an adversary that can control only one processor at a time, can corrupt all the clocks in the system by moving fast enough from processor to processor.

[3]In general $\mathsf{adj}_p$ does not have to be a discrete variable, and it could also depend on $\tau$. We don't use that generality in this paper, though.

[4]This formulation of "good links" does not completely rule out replay of old messages. This does not pause a problem for our application, however.

**Definition 2 (Limited Adversary)** *Let $\theta > 0$ and $f \in \{1, 2, \ldots, n\}$ be fixed. An adversary is $f$-limited (with respect to $\theta$) if during any time interval $[\tau, \tau + \theta]$, it controls at most $f$ processors. We refer to $\theta$ as the* time period *and to $f$ as the* number of faulty processors.

Notice that Definition 2 implies in particular that an $f$-limited adversary who controls $f$ processors and wants to break into another one, must leave one of its current processors at least $\theta$ time units before it can break into the new one. In the rest of the paper we assume that $n \geq 3f + 1$.

### 2.3 Clock Synchronization Protocols

Intuitively, the purpose of a clock synchronization algorithm is to ensure that processors' local clocks remain close to each other and close to the real time, and that faulty processors become synchronized again quickly after the adversary leaves them. It is clear, however, that no protocol can achieve instantaneous recovery, and we must allow processors some time to recover. Typically we want this recovery time to be no more than $\theta$, so by the time the adversary breaks into the new processors, the ones that it left are already recovered.

**Definition 3 (Clock Synchronization)** *Consider a clock synchronization protocol $\pi$ that is executed in a network with drift rate $\rho$ and message delivery bound $\lambda$, and in the presence of an $f$-limited adversary with respect to time period $\theta$.*

*i. We say that $\pi$ ensures* synchronization *with maximal deviation $\delta$, if at any time $\tau$ and for any two processors $p, q$ not faulty during $[\tau - \theta, \tau]$, it holds that $|C_p(\tau) - C_q(\tau)| \leq \delta$.*

*ii. We say that $\pi$ ensures* accuracy *with maximal drift $\tilde{\rho}$ and maximal discontinuity $\mu$, if whenever $p$ is not faulty during an interval $[\tau_1 - \theta, \tau_2]$ (with $\tau_1 \leq \tau_2$), it holds that*

$$(\tau_2 - \tau_1)/(1 + \tilde{\rho}) \ - \mu \ \leq \ C_p(\tau_2) - C_p(\tau_1) \ \leq \ (\tau_2 - \tau_1) \cdot (1 + \tilde{\rho}) \ + \mu \tag{4}$$

## 3 A Clock Synchronization Protocol

As in most (practical) clock synchronization protocols, the most basic operation in our protocol is the estimation by a processor of its peers' clocks. We therefore begin in Subsection 3.1 by discussing the requirements from a clock estimation procedure and describing a simple (known) procedure for doing that. Then, in Subsection 3.2 we describe the clock synchronization protocol itself. In that description we abstract the clock estimation procedure, and view it a "black box" that provides only the properties that were discussed before. Finally, in Subsection 3.3 we elaborate on some aspects of our protocol, and compare it with similar synchronization protocols for other models.

### 3.1 Clock Estimation

Our protocol's basic building block is a subroutine in which a processor $p$ estimates the clock value of another processor $q$. The (natural) requirements from such a procedures are

> *Accuracy.* The value returned from this procedure should not be too far from the actual clock value of processor $q$.

*Bounded error.* Along with the estimated clock value, $p$ also gets some upper bound on the error of that estimation.

For technical reasons it is also more convenient to have this procedure return the *distance* between the local clocks of $p$ and $q$, rather than the clock value of $q$ itself. Hence we define a clock estimation procedure as a two-party protocol, such that when a processor $p$ invokes this protocol, trying to estimate the clock value of another processor $q$, the protocol returns two values $(d_q, a_q)$ (for **d**istance and **a**ccuracy). These values should be interpreted as "since the procedure was invoked, there was a point in which the difference $C_q - C_p$ was about $d_q$, up to an error of $a_q$". Formally, we have

**Definition 4** *We say that a clock estimation routine has reading error $\gamma$ and timeout* $\mathsf{MaxWait}$*, if whenever a processor $p$ is non-faulty during time interval $[\tau, \tau+\mathsf{MaxWait}]$, and it calls this routine at real time $\tau$ to estimate the clock of $q \in N(p)$, then the routine returns at real time $\tau' \le \tau+\mathsf{MaxWait}$, with some values $(d, a)$. Moreover, if $q$ and $q$ were non-faulty during the interval $[\tau, \tau']$, then the values $(d, a)$ satisfy the following:*

- $a \le \gamma$, *and*
- *There was a time $\tau'' \in [\tau, \tau']$ at which $C_q(\tau'') - C_p(\tau'') \in [d - a, d + a]$.*

We now describe a simple clock estimation algorithm. The requestor $p$ sends a message to $q$, who returns a reply to $p$ containing the time according on the clock of $q$ (when sending the reply). If $p$ does not receive a reply within some local time $\mathsf{MaxWait}$, then $p$ aborts the estimation and sets $d_q = 0$ and $a_q = \infty$. Otherwise, if $p$ sends its "ping" message to $q$ at local time $S$, and receives an answer $C$ at local time $R$, it sets $d_q = C - \frac{R+S}{2}$ and $a_q = \frac{R-S}{2}$.

The timeout should be set so that $\mathsf{MaxWait} \ge 2(1+\rho)\lambda$ (where $\lambda$ is the message delivery bound and $\rho$ is the drift bound), and then this procedure has reading error of at most $\tilde{\lambda} \stackrel{def}{=} (1+\rho)\lambda$. Intuitively, $p$ estimates that at its local time $\frac{R+S}{2}$, $q$'s time was $C$. If the network is totally symmetric, that is the time for the message to arrive was identical on the way from $p$ to $q$ and on the way back from $q$ to $p$, and $p$'s clock progressed between $S$ and $R$ at constant rate, then the estimation would be totally accurate. In any case, if $q$ returned an answer $C$, then at some time between $p$ local time $S$ and $p$ local time $R$, $q$ had the value $C$, so the estimation of the offset can't miss by more than $\frac{R-S}{2}$. Since the round-trip time is at most $2\lambda$ and the drift rate is bounded by $\rho$, we have $\frac{R-S}{2} \le \tilde{\lambda}$.

This simple procedure can be "optimized" in several ways. A common method, which is used in practice to decrease the error in estimating the peer's clock (at the expense of worse timeliness), is to repeatedly ping the other processor and choose the estimation given from the ping with the least round trip time. This is used, for example, in the NTP protocol [22].

Also to reduce network load it may be possible to piggyback clock querying messages on other messages, or to perform them in a different thread which will spread them across a time interval. Of course, if we implement the latter idea in the mobile adversary setting, a clock synchronization protocol should periodically check that this thread exists and restart it otherwise (to protect against the adversary killing that thread). We note that when implemented this way, we cannot guarantee the conditions of Definition 4 anymore, since the separate thread may return an old cached value which was measured before the call to the clock estimation procedure. Hence, the analysis in this paper cannot be applied "right out of the box" to the case where the time estimation is done in a separate thread.

Figure 1: Algorithm SYNC for processor $p$

```
Parameters: SyncInt        // time between sunchronizations
            WayOff         // bound for clocks which are very far from the rest

Every SyncInt time units call sync()

function sync () {
    For each q ∈ N(p) do
        (d_q, a_q) ← estimateOffset(q)
        overline{d}_q = d_q + a_q              // overestimate d_q
        underline{d}_q = d_q − a_q             // underestimate d_q
    m ← the f + 1'st smallest overline{d}_q
    M ← the f + 1'st largest underline{d}_q
    If m ≥ −WayOff and M ≤ WayOff
        then adj_p ← adj_p + (min(m, 0) + max(M, 0))/2
        else adj_p ← adj_p + (m + M)/2
}
```

### 3.2 The Protocol

Our clock synchronization protocol uses a clock estimation procedure such as the one described in Section 3.1, which we denote by *estimateOffset*, with the time-out bound denoted by MaxWait and maximal error $\tilde{\lambda} = (1+\rho)\lambda$. Other parameters in this protocol are the (local) time SyncInt between two executions of the synchronization protocol, and a parameter WayOff, which is used by a processor to gauge the distance between its clock and the clocks of the other processors. These parameters are computed from some estimates of the network model parameters $\rho, \lambda$ and $\theta$. Roughly, they should satisfy the following constraints:

$\mathsf{MaxWait} \geq 2\tilde{\lambda}$.

$\theta/5 > \mathsf{SyncInt} > 2\mathsf{MaxWait}$.

$\mathsf{WayOff} > 17\tilde{\lambda} + 18\rho(1+\rho)(\mathsf{SyncInt} + 2\mathsf{MaxWait})$.

These settings are further discussed in the analysis (Section 4) and in Section 3.3.

The protocol, SYNC, is described in Figure 1. The basic idea is that each processor $p$ uses *estimateOffset* to get an estimate for the clocks of its neighbors. Then $p$ eliminates the $f$ smallest and $f$ largest values, and use the remaining values to adjust its own clock. Roughly, $p$ computes a "low value" $C^m$ which is the $f+1$'st smallest estimate, and a "high value" $C^M$ which is the $f+1$'st largest estimate. If $p$'s own clock $C_p$ is more than WayOff away from the interval $[C^m, C^M]$, then $p$ knows that its clock is too far from the clocks of the "good processors", so it ignores its own clock and resets it to $(C^m + C^M)/2$. Otherwise, $p$'s clock is "not too far" from the other processors, so we would like to limit the change to it. In this case, instead of completely ignoring the old clock value, $p$ resets its clock to $(\min(C^m, C_p) + \max(C^M, C_p))/2$. (So if $p$'s clock was below $C^m$ or above $C^M$, it will only move half-way towards these values.)

The details of the SYNC protocol are slightly different, though, specifically in the way that the "low value" and "high value" are chosen. Processor $p$ first uses the error bounds to generate overestimates and underestimates for its neighbors' clock values, and then computes the "low value" $C^m$ as the $f+1$'st smallest overestimate, and the "high value" the $f+1$'s largest underestimate. In the analysis we also assume that all the clock estimations are done in parallel, and that the time that it takes to make the local computations is negligible, so a run of SYNC takes at most MaxWait time on the local clock. (This is not really crucial, but it saves the introduction of an extra parameter in the analysis.)

### 3.3 Discussion

Our SYNC protocol follows the general framework of "convergence function synchronization algorithms" (see [27]), where the next clock value of a processor $p$ is computed from its estimates for the clock values of other processors, using a fixed, simple convergence function.

**No rounds.** As mentioned in the introduction, one notable differences between our protocol and other protocols that have been proposed in the literature, is that many convergence function protocols (for example [8, 9]) proceed in *rounds*, where each processor keeps a different logical clock for each round. (A round is the time between two consecutive synchronization protocols.) In these protocols, if a processor is asked for a "round-$i$" clock when this processor is already in its $i+1$'st round, it would return the value of its clock "as if it didn't do the last synchronization protocol".

In contrast, in our SYNC protocol a processor $p$ always responds with its current clock value. This makes the analysis of the protocol a little more complicated, but it greatly simplifies the implementation, especially in the mobile adversary setting (since variables such as the current round number, last round's clock, and the time to begin the next round have to be recovered from a break in).

**Estimating the network parameters.** A practical advantage of our protocol is that it does not require to know exactly the values of parameters such as the message delivery bound $\lambda$ , the hardware drift $\rho$, or the maximum deviation $\delta$. Such parameters may be hard to measure in practice – and in fact may even change during the course of the execution. In this work we only use the values of the network model parameters in the analysis of the protocol. The protocol itself depends only on some constants such as MaxWait, SyncInt and WayOff, which can be set by overestimating the network model parameters without much harm.

In particular, over-estimating the model parameters by a multipliciative factor of $C$ would result only in a delay of at most $\log C$ synchronization operations before processor's clock is recovered. Also, for a reasonable setting of parameters, such overestimate would worsen the maximum deviation and logical drift guarantees by a multipliciative factor of at most $(1+C/2^{20})$.

**When to perform SYNC?** In our protocol, a processor executes the SYNC protocol every SyncInt time units of local time, and we do not make any assumptions about the relative times of SYNC executions in different processors. A common way to implement this is to set up an alarm at the end of each execution, and to start the new execution when this alarm goes off. In the mobile adversary setting, one must make sure that this alarm is recovered after a break-in.

We note that our analysis does not depend on the processors executing a SYNC exactly every SyncInt time units. Rather, all we need is that during a time interval of $(1+\rho)(\mathsf{SyncInt}+\mathsf{MaxWait})$ real time, each processor completes at least one and at most two SYNC's.

# 4 Analysis

In this section we prove that the simple SYNC protocol from Section 3.2 provides synchronization and accuracy, and compute some bounds on the maximum deviation and logical drift of this protocol. We first present our main theorem in Subsection 4.1. Then we present in Subsection 4.2 an alternate view of the SYNC protocol, which is more convenient for the analysis. Next, in Subsection 4.3 we sketch a high-level overview of the main ideas in the proof. In Subsection 4.4 we introduce some syntax which is useful to handle clock drifts, in Subsection 4.5 we prove our main technical lemma, and finally in Subsection 4.6 we complete the formal proof or the main theorem.

## 4.1 Main Theorem

Let $T$ denote some value such that every non-faulty processor completes at least one and at most two full SYNC's during any interval of length $T$. Specifically, setting $T = (1+\rho)(\mathsf{SyncInt}+2\mathsf{MaxWait})$ is appropriate for this purpose (where $\mathsf{SyncInt}$ is the time that is specified in the protocol, $\mathsf{MaxWait}$ is a bound on the execution time of a single SYNC, and $\rho$ is the drift rate). The following theorem characterizes the performance achieved by our protocol:

**Theorem 5** *Let $T$ be as defined above. Assume that $T < 2(1-\rho)\mathsf{SyncInt}$ and that $\theta > 10T$, and denote $k \stackrel{def}{=} \lfloor \frac{\theta}{2T} \rfloor$ (so $k \geq 5$).*
*Also assume that $\mathsf{WayOff} \geq 17\left(1+\frac{18}{2^k-18}\right)\tilde{\lambda} + 18\left(1+\frac{16}{2^k-18}\right)\rho T$ (with $\tilde{\lambda} = (1+\rho)\lambda$). Then*

*i. The SYNC protocol fulfills the synchronization requirement with maximum deviation*

$$\delta = 16\tilde{\lambda} + 16\rho T + 16C$$

*where $C = \frac{\mathsf{WayOff}+2\tilde{\lambda}}{2^k-2}$.*

*ii. The SYNC protocol fulfills the accuracy requirement with logical drift and discontinuity*

$$\tilde{\rho} = \rho + \frac{16\tilde{\lambda}+C}{5\Theta} \quad and \quad \mu = \delta$$

In a typical setting $\rho$ would be very small, and $T$ would be small compared to $\Theta$. Therefore the significant term in the maximum deviation would be $16\tilde{\lambda}$ and the logical drift $\tilde{\rho}$ would be close to the hardware drift $\rho$.

## 4.2 Clock Bias

For the purpose of analysis, it is more convenient to consider the *bias* of the clocks, rather than the clock values themselves. The bias of processor $p$ at time $\tau$ is the difference between its logical clock and the real time, and is denoted by $B_p(\tau)$. Namely,

$$B_p(\tau) \stackrel{def}{=} C_p(\tau) - \tau \tag{5}$$

When the real-time $\tau$ is implied in the context, we often omit it from the notation and write just $B_p$ instead of $B_p(\tau)$.

Figure 2: Algorithm Sync for processor $p$: bias formulation

Parameters: SyncInt // time between sunchronizations
WayOff // Bound for clocks which are very far from the rest

1. Every SyncInt time units call *sync()*

3. function *sync* () {
4. For each $q \in N(p)$ do
5. $(d_q, a_q) \leftarrow$ *estimateOffset(q)*
6. $\overline{B}_q = B_p + d_q + a_q$ // overestimate $B_q$
7. $\underline{B}_q = B_p + d_q - a_q$ // underestimate $B_q$
8. $B_p^{(m)} \leftarrow$ the $f+1$'st smallest $\overline{B}_q$
9. $B_p^{(M)} \leftarrow$ the $f+1$'st largest $\underline{B}_q$
10. If $B_p - B_p^{(m)} \leq \mathsf{WayOff}$ and $B_p^{(M)} - B_p \leq \mathsf{WayOff}$
11. then $B_p \leftarrow \left(\min(B_p^{(m)}, B_p) + \max(B_p^{(M)}, B_p)\right)/2$
12. else $B_p \leftarrow \left(B_p^{(m)} + B_p^{(M)}\right)/2$
13. }

In the analysis below we view the protocol Sync as affecting the biases of processors, rather than their clock values. In particular, in an execution of Sync by processor $p$, we can view $d_q$ as an estimate for $B_p - B_q$ rather than an estimate for $C_p - C_q$, and we can view the modification of $\mathsf{adj}_p$ in the last step as a modification of $B_p$. We can therefore re-write the protocol in terms of biases rather than clock values, as in Figure 2.

We note that by referencing $B_p$ in the protocol, we mean $B_p(\tau)$ where $\tau$ is the real time where this reference takes place. We stress that the protocol cannot be implemented as it is described in Figure 2, since a processor $p$ does not know its bias $B_p$. Rather, the above description is just an alternative view of the "real protocol" that is described in Figure 1.

## 4.3 Proof Overview of the Main Theorem

We start with an informal, high level overview of the proof. For simplicity, in this overview we only look at the case with no drifts and no clock-reading errors, namely $\rho = \lambda = 0$, and also assume that the network is fully connected (that is $N(p) = \{1, \ldots, n\}$. (Note that since $\rho = \lambda = 0$, we always have $a_q = 0$, so in Steps 6-7 of the protocol we get $\underline{B}_q = \overline{B}_q = B_p + d_q$.)

The analysis looks at consecutive time intervals $I_0, I_1, \ldots$, each of length $T$, and proceeds by induction over these intervals. For each interval $I_i$ we prove "in spirit" the following claims:

i. The bias values of the "good processors" get closer together: If they were at distance $\delta$ from each other at the beginning of $I_i$, they will be at distance $7\delta/8$ at the end of it.

ii. The bias values of the "recovering processors" gets (much) closer to those of the good processors. If a recovering processor was at distance $\xi$ from the "range of good processors" at the beginning of $I_i$, it would be at distance at most $\xi/2$ from that range at the end of $I_i$.

It therefore follows that after a few such intervals, the bias of a "recovering processor" will be at most $\delta$ away from those of the "good processors".

To prove the above claims, our main technical lemma considers a given interval $I_i$, and assumes that there is a set $G$ of at least $n-f$ processors, which are all non-faulty throughout $I_i$, and all have bias values in some small range at the beginning of $I_i$ (w.l.o.g., this can be the range $[-D, D]$). Then, we prove the following three properties:

**Property 1** We first show that the biases of the processors in $G$ remain in the range $[-D, D]$ throughout the interval $I_i$. This is so because in every execution of SYNC, a processor $p \in G$ always gets biases in that range from all other processors in $G$. Since $G$ contains more than $2f$ processors, then both $B_p^{(m)}$ and $B_p^{(M)}$ are in that range, and so $p$'s bias remains in that range also after it completes the SYNC protocol.

Also, if follows from the same argument that we always have $B_p - B_p^{(m)} \leq \mathsf{WayOff}$ and $B_p^{(M)} - B_p \leq \mathsf{WayOff}$, so processor $p$ never ignores its own current bias in Step 11 of the protocol.

**Property 2** Next we consider processors $p \in G$ whose initial bias values are low (say, below the median for $G$). Since $p$ executes at most two SYNC's during the time interval $I_i$, and in each SYNC it takes the average of its own current bias and another bias below $D$, the bias of $p$ remains bounded strictly below $D$. Specifically, one can show that the resulting bias values cannot be larger than $(Z_0 + 3D)/4$ (where $Z_0$ is the initial median value).

Similarly, for the processors $q \in G$ with high initial bias values, the bias values remain bounded strictly above $-D$, specifically at least $(Z_0 - 3D)4$.

**Property 3** Last, we use the result of the previous steps to show that at the end of the interval, the bias of every processor in $G$ is between $(Z_0 - 7D)/8$ and $(Z_0 + 7D)/8$. (Hence, in the case of no errors or drifts, the size of the interval that includes all the processors in $G$ shrunk from $2D$ to $7D/4$.)

To see this, recall that by the result of the previous step, whenever a processor $p \in G$ executes a SYNC, it gets bias values which are bounded by $(Z_0+3D)/4$ from all the processors with low initial biases – and so its low estimate $B_p^{(m)}$ must also be smaller than $(Z_0+3D)/4$. Similarly, it gets bias values which are bounded above $(Z_0 - 3D)/4$ from all the processors with high initial biases – and so its high estimate $B_p^{(M)}$ must be larger than $(Z_0 - 3D)/4$. Since the bias of $p$ after its SYNC protocol is computed as $(\min(B_p^{(m)} + B_p) + \max(B_p^{(M)} + B_p))/2$, and since $B_p$ is in the range $[-D, D]$ by the result of the first step, the result of this step follows.

Moreover, a similar argument can be applied even to a processor outside $G$, whose initial bias is not in the range $[-D, D]$. Specifically, we can show that if at the beginning of interval $I_i$, a non-faulty processor $p$ has high bias, say $D+\xi$ for some $\xi > 0$, then at the end of the interval the bias of $p$ is at most $(Z_0 + 7D)/8 + \xi/2$. Hence, the distance between $p$ and the "good range" shrinks from $\xi$ to $\xi/2$.

A formal analysis follows, including the effects of drifts and reading errors.

## 4.4 Envelopes

It is useful to think of the bias values of processors as if they are drawn on a two-dimensional plane, where one axis is real time values and the other axis is bias values. (We call this the $(\tau, \beta)$-plane.)

Note that since the drift of processors is bounded by $\rho$, the bias of a processor that does not reset its clock during a time interval of length $\tau$ cannot change by more than $\rho\tau$. Hence, if we know that the bias of a processor at time $\tau$ is in the interval $[a, b]$, and that this processor did not reset its clock between time $\tau$ and $\tau'$, then its bias at time $\tau'$ must be in the interval $[a - \rho(\tau' - \tau),\ b + \rho(\tau' - \tau)]$. This motivates the following definition of an *envelope* in the $(\tau, \beta)$-plane.

**Definition 6** *An* envelope *in the $(\tau, \beta)$-plane is a region of the form*

$$E = \{(\tau, \beta) \mid \tau \geq \tau_0,\ a - \rho(\tau - \tau_0) \leq \beta \leq b + \rho'(\tau - \tau_0)\}$$

*For some real values $a \leq b$ and $\tau_0$.*

*We sometimes write $E = \textsc{Env}\{\tau_0, [a, b]\}$ when we want to stress the parameters $\tau_0$, $a$ and $b$.*

See Figure 3 for examples of a few envelopes. Some notations and facts that we use throughout the proof are summarized below.

- If $E$ is an envelope, we denote by $E(\tau')$ the interval on the $\beta$-axis that corresponds to the intersection of $E$ and the line $\tau = \tau'$. For instance, in the notation above $E(\tau_0) = [a, b]$.

  More formally, if $E = \textsc{Env}\{\tau_0, [a, b]\}$, then for all $\tau \geq \tau_0$ we have $E(\tau) = [a - \rho(\tau - \tau_0),\ b + \rho(\tau - \tau_0)]$. We also denote by $|E(\tau)|$ the size of the interval $E(\tau)$, namely for any $t > 0$ we have $|E(\tau_0 + t)| = b - a + \rho t$.

- We say that the bias of processor $p$ is in the envelope $E$ during interval $[\tau_1, \tau_2]$, if $B_p(\tau) \in E(\tau)$ for any $\tau \in [\tau_1, \tau_2]$. We say that the bias of $p$ is *not above* $E$ during $[\tau_1, \tau_2]$ if for any $\tau \in [\tau_1, \tau_2]$, we have $B_p(\tau) \leq \max E(\tau)$. Similarly we say that the bias of $p$ is *not bellow* $E$ during $[\tau_1, \tau_2]$ if for any $\tau \in [\tau_1, \tau_2]$, $B_p(\tau) \geq \min E(\tau)$.

- If $E = \textsc{Env}\{\tau_0, [a, b]\}$ is an envelope and $c$ is a non-negative number, then we denote by $E + c$ the envelope obtained from $E$ by extending it by $c$ at both sides. That is, $\textsc{Env}\{\tau_0, [a, b]\} + c = \textsc{Env}\{\tau_0, [a - c, b + c]\}$.

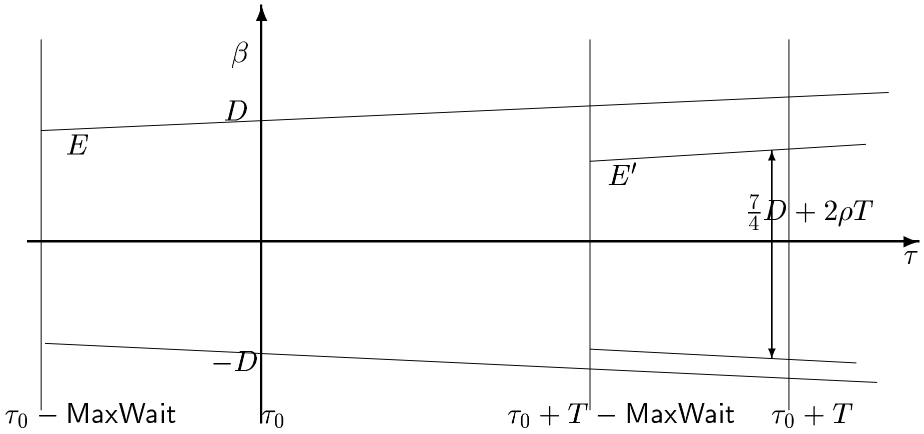
- If $E = \textsc{Env}\{\tau_0, [a, b]\}$ and $E' = \textsc{Env}\{\tau_0, [a', b']\}$ are two envelopes, then the *average* of $E$ and $E'$ is the envelope $avg(E, E') = \textsc{Env}\{\tau_0, [(a + a')/2,\ (b + b')/2]\}$.

  Note that if at some time $\tau$ we have one bias value $\beta$ in $E$ and another bias value $\beta'$ in $E'$, then the average of these bias values $(\beta + \beta')/2$ is in $avg(E, E')$. (Similarly if we only know that $\beta, \beta'$ are not above $E, E'$, respectively, then it follows that their average is not above $avg(E, E')$, and the same for "not below".)

## 4.5 Main technical lemma

Recall that we denote $T \stackrel{def}{=} (1 + \rho)(\mathsf{SyncInt} + 2\mathsf{MaxWait})$ and that we assume $T < 2(1 - \rho)\mathsf{SyncInt}$. Note that for a real time interval of length $T$, the following properties hold:

Figure 3: The envelopes $E$ and $E'$ of Lemma 7




- In any interval of length $T - (1+\rho)\mathsf{MaxWait}$, every non-faulty processor completes at least one full SYNC. (That is, a SYNC that starts and ends within the interval.)
- In any interval of length $T$, every non-faulty processor performs at most two SYNCs.

Our main technical lemma is as follows.

**Lemma 7** *Let $\tau_0$ be some real time, and denote $\tau_1 = \tau_0 + T$. Let $\mathsf{G}$ be a set of at least $n - f$ processors, all of which are non-faulty during the real time interval $[\tau_0 - \mathsf{MaxWait}, \tau_1]$. Also, let $D$ be a positive number such that $2D \leq \mathsf{WayOff} - 2\rho T - \tilde{\lambda}$.*

*If there exists an envelope $E$ such that $|E(\tau_0)| \leq 2D$, and the biases of $\mathsf{G}$'s members are in $E$ throughout the interval $[\tau_0 - \mathsf{MaxWait}, \tau_0]$, then*

*i. The biases of $\mathsf{G}$'s members remain in $E$ throughout the interval $[\tau_0, \tau_1]$.*

*ii. Furthermore, there exists an envelope $E' \subseteq E$, where $|E'(\tau_0)| = \frac{7D}{4}$, and all the biases of all members of $\mathsf{G}$ are in the envelope $(E' + \tilde{\lambda}) \cap E$ in the interval $[\tau_1 - \mathsf{MaxWait}, \tau_1]$.*

*iii. If $p$ is non-faulty in $[\tau_0, \tau_1]$ and $B_p(\tau_0)$ is in $E + \xi$ for some $\xi \geq 0$ then $p$'s bias remains in $E + \xi$ throughout $[\tau_0, \tau_1]$. Furthermore $p$'s bias is in the envelope $E' + \tilde{\lambda} + \frac{\xi'}{2}$ in the interval $[\tau_1 - \mathsf{MaxWait}, \tau_1]$, where $\xi' = \min(\xi,\ \mathsf{WayOff} + 2\tilde{\lambda})$.*

The envelopes $E$ and $E'$ are depicted in Figure 3. Part (i) of the lemma guarantees that biases of all good members remain within the drift bound. Part (ii) shows that the deviation among the good members actually shrinks (if $\frac{7D}{4} + 2\tilde{\lambda} < 2D$). Finally, part (iii) refers to recovering processors and shows that their distance from the good members halves at each interval of length $T$.

**Proof of Lemma 7** We assume $\mathsf{G}$ and $E$ are as stated in Lemma 7. We assume w.l.o.g that $E(\tau_0) = [-D, +D]$. We first prove item i of the lemma:

**Claim 8** *The bias values of all processors in $\mathsf{G}$ remain in $E$ throughout the interval $[\tau_0, \tau_1]$. Moreover, in every execution of SYNC during this interval, members of $\mathsf{G}$ always use their old clock values in Step 11 of the protocol (and do not ignore them in Step 12).*

**Proof:** We prove this by induction on the time. Suppose this is true up to some time $\tau' \in [\tau_0, \tau_0+T]$. Let $\tau$ be the first time after $\tau'$ in which some processor in $\mathsf{G}$ resets its clock value. Clearly, the biases of all processors remain in $E$ until $\tau$ (since $E$ is an envelope).

Let $p$ be a processor $p \in \mathsf{G}$ that changes its clock at time $\tau$. As $p$ is non-broken, it only resets its clock at the conclusion of a SYNC. Since $|\mathsf{G}| \geq n - f$ and $N(p) \geq 3f + 1$, then there must be at least $2f + 1$ processors in $N(p) \cap \mathsf{G}$ and at most $f$ in $N(p) \setminus \mathsf{G}$. Now notice that the bias values of all the processors in $N(p) \cap \mathsf{G}$ are in $E$, and that all of these processors are non-faulty. If follows that the overestimates that $p$ computes for them in Step 6 of the protocol are not below $E$ and the underestimates that it computes for them in Step 7 are not above $E$. Hence $B_p^{(m)}$ is not below $E$ and $B_p^{(M)}$ is not above $E$.

Since $B_p$ is in $E$, we conclude that $B_p - B_p^{(m)} \leq |E(\tau)| \leq 2D + 2\rho T \leq \mathsf{WayOff}$, and similarly $B_p^{(M)} - B_p \leq \mathsf{WayOff}$. Therefore, $p$ uses its old bias value in Step 11 of the protocol, and sets the new bias value to be the average of $\min(B_p^{(m)}, B_p)$ and $\max(B_p^{(M)}, B_p)$. As both these values are in $E$, then so is $p$'s new bias value. ■

In what follows we also use the following extension to Claim 8 which also proves part of item iii:

**Claim 9** *Let $\xi \geq 0$, If $p$'s bias is in $E + \xi$ at $\tau_0$, and $p$ is non faulty throughout $[\tau_0, \tau_1]$, then $p$'s bias remains in $E + \xi$ throughout this interval.*

**Proof:** Again since $p$ is non faulty, it only resets its clock as a result of a SYNC(say, at time $\tau \in [\tau_0, \tau_1]$), at which it gets the responses of all the members of $\mathsf{G} \cap N(p)$. We get, as before, that in this execution of SYNC the value $B_p^{(M)}$ is not above $E$ and the value $B_p^{(m)}$ is not below $E$. If $p$'s use its old bias in Step 11, then its new bias value is the average of $\min(B_p^{(m)}, B_p)$ and $\max(B_p^{(M)}, B_p)$, which are both in $E + \xi$ and we are done.

It is left therefore to check the case where $p$'s ignore its old bias value in Step 12. In this case, $p$'s new bias value is set to $(B_p^{(m)} + B_p^{(M)})/2$. We already know that $B_p^{(M)}$ is not above $E$. Recall now that the estimates that $p$ gets for the bias values of processors in $N(p) \cap \mathsf{G}$ are all in $E + \tilde{\lambda}$. Thus the overestimates are not above $E + 2\tilde{\lambda}$, and therefore also $B_p^{(m)}$ is not above $E + 2\tilde{\lambda}$. It follows that $p$'s new bias value is not above $avg(E, E + 2\tilde{\lambda}) = E + \tilde{\lambda}$. The symmetric argument shows that $p$'s new bias value is also not below $E + \tilde{\lambda}$.

Next we observe that if $p$ ignores its old bias value in Step 12, then it is either because $B_p - B_p^{(m)} > \mathsf{WayOff}$ or because $B_p^{(M)} - B_p > \mathsf{WayOff}$. Assume w.l.o.g. that we are in the former case, namely $B_p - B_p^{(m)} > \mathsf{WayOff}$. On the other hand, we know that $B_p^{(m)}$ is not below $E$, and that prior to the SYNC the value $B_p$ is in $E + \xi$, so $B_p - B_p^{(m)} \leq |E(\tau)| + \xi$. It follows that

$$\xi > \mathsf{WayOff} - |E(\tau)| \geq (2D + 2\rho T + \tilde{\lambda}) - (2D + 2\rho T) = \tilde{\lambda}$$

Hence, $E + \tilde{\lambda} \subseteq E + \xi$, and since $p$'s new bias value is in $E + \tilde{\lambda}$, the claim follows. ■

To prove items ii and iii, we need the following two claims:

**Claim 10** *Let $Z \in [-D, D]$ be a real number. Let $LT_Z$ (resp. $GT_Z$) be the sets of members of $\mathsf{G}$ with bias at most $Z$ (resp. at least $Z$) at time $\tau_0$.*

*Then, during the interval* $[\tau_0, \tau_1]$*, the bias values of all processors in* $LT_Z$ *are in the envelope* $\textsc{Env}\{\tau_0, [-D, X]\}$*, where* $X = (Z + 3D)/4$*. Similarly, during this interval the bias values of all processors in* $GT_Z$ *are in the envelope* $\textsc{Env}\{\tau_0, [W, D]\}$*, where* $W = (Z - 3D)/4$*.*

**Proof:** We prove the lemma for the processors in $LT_Z$. The proof for the processors in $GT_Z$ is symmetric.

Denote $Y = (Z+D)/2$, and consider the envelopes $E_1 = \textsc{Env}\{\tau_0, [-D, Z]\}$, $E_2 = \textsc{Env}\{\tau_0, [-D, Y]\}$, $E_3 = \textsc{Env}\{\tau_0, [-D, X]\}$, and $E = \textsc{Env}\{\tau_0, [-D, D]\}$. By definition of $Z, Y, X, D$ we have $E_2 = avg(E_1, E)$ and $E_3 = avg(E_2, E)$. Also, since $Z \leq D$, we have $Z \leq Y \leq X \leq D$, and therefore $E_1 \subseteq E_2 \subseteq E_3 \subseteq E$.

Let $p \in LT_Z$. Clearly, at time $\tau_0$ the bias value of $p$ is in $E_1$, and it remains in $E_1$ long as $p$ does not perform a $\textsc{Sync}$. After $p$ performs its first $\textsc{Sync}$, its new bias is at most the average of its old bias and its $f + 1$'st largest underestimate $B_p^{(M)}$. By Claim 8 the $f + 1^{st}$ largest underestimate is not above $E$ (since at least $2f + 1$ biases come from $N(p) \cap \mathsf{G}$), hence the new bias is not above $E_2 = avg(E, E_1)$. Similarly if $p$ performs its second $\textsc{Sync}$ in this interval, then its new bias is an average of a value in $E_2$ and a value not above $E$, hence not above $E_3$. ■

**Claim 11** *There is a value* $Z_0 \in [-D, D]$ *such that for any processor* $p$*,*

$$|N(p) \cap LT_{Z_0}|\,, |N(p) \cap GT_{Z_0}| \geq f + 1$$

**Proof:** For any processor $p$, denote by $R_p$ the range $[a_p, b_p]$ where $a_p$ (resp. $b_p$) is the $f + 1^{st}$ smallest (resp. $f + 1^{st}$ largest) bias value at $\tau_0$ of the members of $\mathsf{G} \cap N(p)$. Note that since $|\mathsf{G}| \geq n - f$ and $|N(p)| \geq 3f + 1$, then $|\mathsf{G} \cap N(p)| \geq 2f + 1$ and so $b_p \geq a_p$. What we need to prove is that $\bigcap_p R_p \neq \emptyset$, since any value $Z_0 \in \bigcap_p R_p$ would satisfy the desired properties. (Note that since each $R_p$ is a continuous interval, it is sufficient to show that the intersection of any two such intervals is non-empty.)

Let $p, q$ be any two processors. The fact that $|\mathsf{G}| \geq n - f$ and $|N(p) \cap N(q)| \geq 3f + 1$ implies that $|G \cap N(p) \cap N(q)| \geq 2f + 1$. At most $f$ of the bias values of these processors can be less than $a_p$ because $a_p$ is the $f + 1^{st}$ smallest in $\mathsf{G} \cap N(p)$. This means that there are at least $f + 1$ processors in $\mathsf{G} \cap N(p) \cap N(q)$ with bias which is at least $a_p$, and in particular $b_q \geq a_p$. A symmetric argument show that also $b_p \geq a_q$.

Since this is ture for any two processors $p, q$, we conclude that $\max_p a_p \leq \min_p b_p$, and therefore $\bigcap_p R_p = [\max_p a_p,\ \min_p b_p] \neq \emptyset$. ■

We now fix a value $Z_0$ which satisfies the properties of Claim 11, and prove item iii of Lemma 7. We note that this implies also item ii, since the latter is actually a special case of iii for the case $\xi = 0$.

**Claim 12** *Let* $p$ *be a processor whose bias at time* $\tau_0$ *is in the envelope* $E + \xi$ *for some* $\xi \geq 0$*, and assume that* $p$ *is non faulty during the interval* $[\tau_0, \tau_1]$*. Then, during the time interval* $[\tau_1 - \mathsf{MaxWait}, \tau_1]$*, the bias of* $p$ *is in the envelope* $\bar{E} = E' + \tilde{\lambda} + \frac{\xi'}{2}$*, where* $E'$ *is the envelope* $E' = \textsc{Env}\{\tau_0, [\frac{Z_0 - 7D}{8},\ \frac{Z_0 + 7D}{8}]\}$*, and* $\xi' = \min(\xi,\ \mathsf{WayOff} + 2\tilde{\lambda})$*.*

**Proof:** We show that the bias of $p$ is not above $\bar{E}$. The proof for "not below" is symmetric.

Since $p$ is non faulty in $[\tau_0, \tau_1]$, then it must complete at least one SYNC before time $\tau_1 - \mathsf{MaxWait}$. Consider then the last SYNC that it performs in this time interval. Recall that $|LT_{Z_0} \cap N(p)| \geq f+1$, and by Claim 10 all the processors in $LT_{Z_0}$ have bias in $E_3 = \text{ENV}\{\tau_0, [-D, X]\}$, (where $X = \frac{Z_0+3D}{4}$). Therefore, the $f+1^{st}$ lowest estimate of processor $p$ is not above $E_3 + \tilde{\lambda}$, and the $f+1^{st}$ lowest overestimate, $B_p^{(m)}$, is not above $E_3 + 2\tilde{\lambda}$.

There are two cases now: either $p$ uses its old bias value in Step 11 of the protocol, or it ignores it in Step 12. Recall that by Claim 9, $p$'s old bias value is in $E + \xi$. If $p$ uses this value in Step 11, then its new bias value after the SYNC is not above $avg(E+\xi, E_3 + 2\lambda) = avg(E, E_3) + \tilde{\lambda} + \frac{\xi}{2}$. This implies that it is not above $E' + \tilde{\lambda} + \frac{\xi}{2}$ (since the upper limits of $E'$ and $avg(E, E_3)$ are the same). If $p$ ignores its old bias in Step 12, then its new bias value is the average of $B_p^{(m)}$ (which is not above $E_3 + 2\tilde{\lambda}$) and $B_p^{(M)}$ (which is not above $E$, by the proof of Claim 8). Hence the new bias value is not above $avg(E, E_3) + \tilde{\lambda}$, and therefore not above $E' + \tilde{\lambda}$.

We have shown that in either case, the bias of $p$ after its last SYNC is not above $E' + \tilde{\lambda} + \frac{\xi}{2}$. We next show that even if $\xi > \mathsf{WayOff} + 2\tilde{\lambda}$, the bias value of $p$ cannot be above $E' + \tilde{\lambda} + \frac{\xi'}{2}$. First note that if $p$'s bias value just before its last SYNC is in $E + \mathsf{WayOff} + 2\tilde{\lambda}$, then the same arguments as above apply.

So assume that $p$'s bias value just before its last SYNC is outside the envelope $E + \mathsf{WayOff} + 2\tilde{\lambda}$, and w.l.o.g assume that it is above that envelope. In that execution of SYNC we therefore know that $B_p$ is above $E + \mathsf{WayOff} + \tilde{\lambda}$, and $B_p^{(m)}$ is not above $E_3 + 2\tilde{\lambda}$. Therefore we get

$$B_p - B_p^{(m)} \; > \; \left(D + \mathsf{WayOff} + 2\tilde{\lambda}\right) - \left(\frac{Z_0 + 3D}{4} + 2\tilde{\lambda}\right) \; = \; \frac{D - Z_0}{4} + \mathsf{WayOff} \; \geq \; \mathsf{WayOff}$$

Hence, in this case $p$ ignores its old bias value in Step 12, and its new bias value is not above $E' + \tilde{\lambda}$. This complete the proof for time $\tau_1 - \mathsf{MaxWait}$. Finally, we note that even if $p$ completes another SYNC during the interval $[\tau_1 - \mathsf{MaxWait}, \tau_1]$, the arguments above still hold for that SYNC as well, and so $p$'s bias value remains in the envelope $\bar{E}$ throughout that time interval. ■

### 4.6 Proof of Theorem 5

Let $C \overset{def}{=} \frac{\mathsf{WayOff}+2\tilde{\lambda}}{2^k-2}$, and let $D$ be a positive number such that

$$16\tilde{\lambda} + 16\rho T + 16C \; \leq \; 2D \; \leq \; \mathsf{WayOff} - 2\rho T - \tilde{\lambda}$$

(recall that the maximum deviation which we prove is $\delta = 2D$).

Fix any real number $t \in [0, T)$, denote $t_i = iT + t$, and consider the time intervals $I_0 = [0, t_1]$, and $I_i = [t_i - \mathsf{MaxWait}, t_{i+1}]$ for $i \geq 1$. We prove inductively the following claim:

**Claim 13** *There are envelopes $E_0, E_1, E_2, \ldots$ such that*

*i.* $|E_i(\tau_i)| \leq 2D$, *and $E_i$ is "almost contained" in $E_{i-1}$. Specifically $E_i \subseteq E_{i-1} + \frac{2\tilde{\lambda}}{k} + C$.*

*ii. If processor $p$ is non faulty in some interval $[t_j, t_i]$ (with $0 \leq j < i$), then during the interval $[t_i - \mathsf{MaxWait}, t_i]$, $p$'s bias value is in the envelope $E_i + 2\tilde{\lambda} + \xi$, where $\xi = \max\left(0, \; \frac{\mathsf{WayOff}+2\tilde{\lambda}+2C}{2^{i-j}} - 2C\right)$.*

iii. *If processor $p$ is non faulty in some interval $[t_{i+1} - \theta, t_i]$, then during the interval $[t_i - \mathsf{MaxWait}, t_i]$, $p$'s bias value is in the envelope $E_i$. If $p$ remain non faulty throughout $t_{i+1}$, then its bias value remains in $E_i$ throughout the interval $I_i$.*

**Proof:** We prove the claim by induction on $i$. For all $i = 0, 1, \ldots$, denote by $\mathsf{G}_i$ the set of processors that are non-faulty in the interval of length $\theta$ that ends at time $t_{i+1}$. By our assumption on the power of the adversary, we know that the size of each $\mathsf{G}_i$ is at least $n - f$. Note also that since $\theta > T + \mathsf{MaxWait}$, then in particular all the processors in $\mathsf{G}_i$ are non-faulty during the interval $I_i$.

For the base step, we assume that all the processors start the execution (at time 0) with initial bias in the interval $[-D + 2\rho T, D - 2\rho T]$. Since interval $I_0$ ends before time $2T$, and since $\theta > 2T$, then the processors in $\mathsf{G}_0$ are exactly those that are non-faulty throughout $I_0$, and their bias remain in the interval $[-D, +D]$ throughout this interval. Assume therefore that the claim holds through time interval $I_{i-1}$ and we prove that it still holds in time interval $I_i$.

*Proof of item i.* We first note that items i and iii imply that $E_{i-1}$ and $\mathsf{G}_{i-1}$ satisfy the conditions of Lemma 7 (for $\tau_0 = t_{i-1}, \tau_1 = t_i$). By item ii of that lemma, there exists an envelope $E'_{i-1} \subseteq E_{i-1}$, such that the bias values of all members of $\mathsf{G}_{i-1}$ are in $E'_{i-1} + \tilde{\lambda}$ in the interval $[\tau_i - \mathsf{MaxWait}, \tau_i]$, and with $|E'_{i-1}(\tau_{i-1})| = \frac{7D}{4}$ (and therefore $|E'_{i-1}(\tau_i)| = \frac{7D}{4} + 2\rho T$). Define now

$$E_i \stackrel{def}{=} ((E'_{i-1} + \tilde{\lambda}) \cap E_{i-1}) + \frac{2\tilde{\lambda}}{k} + C$$

and we get $|E_i(\tau_i)| \le \frac{7D}{4} + 2\rho T + 2\tilde{\lambda} + 2C \le 2D$, where the last inequality holds since $D \ge 8\tilde{\lambda} + 8\rho T + 8C$. Also, since $E'_{i-1} \subseteq E_{i-1}$ then $E_i \subseteq E_{i-1} + \tilde{\lambda} + C$. ■

*Proof of item ii.* Consider a processor $p$ which is non faulty during the interval $[t_j, t_i]$ for some $j < i$. If $j = i - 1$, then by item iii of Lemma 7, the bias of $p$ during the interval $[t_i - \mathsf{MaxWait},\ t_i]$ is in the envelope

$$E'_{i-1} + \tilde{\lambda} + \tfrac{\mathsf{WayOff}+2\tilde{\lambda}}{2} \;=\; (E'_{i-1} + \lambda + C) + \left(\tfrac{\mathsf{WayOff}+2\tilde{\lambda}}{2} - C\right) \subseteq E_i + \left(\tfrac{\mathsf{WayOff}+2\tilde{\lambda}+2C}{2} - 2C\right)$$

If $j < i - 1$, then by item ii of the induction hypothesis, the bias of $p$ at time $t_{i-1}$ is in $E_{i-1} + \xi'$ (where $\xi' = \max\left(0,\ \frac{\mathsf{WayOff}+2\tilde{\lambda}+2C}{2^{i-j-1}} - 2C\right)$). By item iii of Lemma 7, this means that $p$'s bias during the interval $[t_i - \mathsf{MaxWait},\ t_i]$ is in the envelope $E'_{i-1} + 2\tilde{\lambda} + \frac{\xi'}{2}$. However, by definition of $\xi, \xi'$ we have $\frac{\xi'}{2} \le C + \xi$, and therefore $E'_{i-1} + 2\tilde{\lambda} + \frac{\xi'}{2} \subseteq E'_{i-1} + 2\tilde{\lambda} + C + \xi \subseteq E_i + 2\lambda + \xi$. ■

*Proof of item iii.* Finally, we need to show that if $p$ is non faulty in $[t_i - \theta + T, t_i]$, then its bias value is in $E_i$ during the interval $[t_i - \mathsf{MaxWait}, t_i]$.

Recall that we have $\theta \ge 2kT$ (with $k \ge 5$). It follows that the $p$ is non faulty during the interval $[t_{i-2k+1},\ t_i]$.

We will first look at the interval $[t_{i-2k+1},\ t_{i-k}]$.

By item ii which we proved above, $p$'s bias value is in the envelope $E_{i-k} + 2\tilde{\lambda}$ during time $[t_{i-k} - \mathsf{MaxWait}, t_{i-k}]$, where $\xi = \max\left(0,\ \frac{\mathsf{WayOff}+2\tilde{\lambda}+2C}{2^{k-1}} - 2C\right)$. However, we defined $C = (\mathsf{WayOff} + 2\tilde{\lambda})/(2^k - 2)$, so we get

$$\frac{\mathsf{WayOff} + 2\tilde{\lambda} + 2C}{2^{k-1}} - 2C \;=\; \frac{\mathsf{WayOff} + 2\tilde{\lambda} + 2(\mathsf{WayOff} + 2\tilde{\lambda})/(2^k - 2)}{2^{k-1}} - 2\frac{\mathsf{WayOff} + 2\tilde{\lambda}}{2^k - 2}$$

$$= \quad (\mathsf{WayOff} + 2\tilde{\lambda}) \left( \frac{1 \ + 2/(2^k - 2)}{2^{k-1}} - \frac{2}{2^k - 2} \right)$$
$$= \quad (\mathsf{WayOff} + 2\tilde{\lambda}) \cdot 0 \quad = \quad 0$$

This means that the bias values of $p$ is in the envelope $E_{i-k} + 2\tilde{\lambda}$ during the interval $[t_{i-k} - \mathsf{MaxWait}, t_{i-k}]$.

We shall now consider the interval $[t_{i-k}, t_i]$

Now we can use item iii of Lemma 7 which says that if $p$'s bias is in $E+\xi$ in time $t_j$ and $p$ is non-faulty throughout $[t_j, t_{j+1}]$ then $p$'s bias is in both $E+\xi$ and $E' + \tilde{\lambda} + \xi$ throughout $[t_{j+1} - \mathsf{MaxWait}, t_{j+1}]$. Applying this and the definition of $E_i$ ($E_i \stackrel{def}{=} ((E'_{i-1} + \tilde{\lambda}) \cap E_{i-1}) + \frac{2\tilde{\lambda}}{k} + C$) repeatedly to the intervals $[t_{i-k}, t_{i-k+1}], \ldots, [t_{i-1}, t_i]$ we can see that since at time $t_{i-k}$ , $p$'s bias is in $E_{i-k} + 2\tilde{\lambda}$, at time $t_{i-k+j}$, $p$'s bias is in $E_{i-k+j} + 2\tilde{\lambda} - \frac{j}{k}2\tilde{\lambda}$. In particular for $j = k$ we see that $p$'s bias is in $E_i$ at time $t_i$.

If $p$ remains non faulty until $t_{i+1}$, then it is in $\mathsf{G}_i$, and by item i of Lemma 7, its bias value remains in the envelope $E_i$ throughout interval $I_i$. ■

**Completing the proof.** Theorem 5 follows easily from Claim 13:

- Let $p$ and $q$ be two processors that are non faulty in some interval $[\tau - \theta, \tau]$, and we write $\tau = \ell T + t$ for some values of $\ell$ and $t < T$. We use item iii of Claim 13 to show that the bias values of both $p$ and $q$ are in the envelope $E_\ell$ at time $t_\ell = \tau$. From item i of that claim, we have $|E_\ell(t_\ell)| < 2D$, so the bias values of $p$ and $q$ cannot be more than $2D$ apart.

- Assume that $p$ is non-broken from $\tau_1 - \theta$ until $\tau_2$ (with $\tau_1 < \tau_2$). We denote $\tau_1 = \ell T + t$ for some $\ell \geq 0$ and $t \in [0, T)$, and also $\tau_2 = \tau_1 + mT + t'$ for some $m \geq 0$ and $t' \in [0, T)$. Using the notations of Claim 13, we have $\tau_1 = t_\ell \in I_\ell$ and $\tau_2 = t_{\ell+m} + t' \in I_{\ell+m}$.

  By Claim 13, the bias value of $p$ at time $\tau_1$ is in the envelope $E_\ell$ and at time $\tau_2$ it is in the envelope $E_{\ell+m}$. Moreover, since for all $i$ we have $E_{i+1} \subseteq E_i + \frac{2}{k}\tilde{\lambda} + C$, then in particular $E_{\ell+m} \subseteq E_\ell + m(\frac{2}{k}\tilde{\lambda} + C)$. Therefore,

  $$|B_p(\tau_2) - B_p(\tau_1)| \leq 2D + m(\frac{2}{k}\tilde{\lambda} + C) + \rho(mT + t')$$

  where the first term in the inequality above is to account for the width of $E_i(iT)$, the second is to account for the additional factors of $\frac{2}{k}\tilde{\lambda} + C$ when moving from each $E_i$ to $E_{i+1}$. and the last term is because the envelopes expand with slope $\rho$, It therefore follows that

  $$\frac{|B_p(\tau_2) - B_p(\tau_2)| - 2D}{\tau_2 - \tau_1} \leq \frac{m(\frac{2}{k}\tilde{\lambda} + C) + \rho(mT + t')}{mT + t'}$$
  $$\leq \rho + \frac{2\tilde{\lambda} + C}{kT} \leq \rho + \frac{4\tilde{\lambda} + C}{0.8\Theta} = \rho + \frac{16\tilde{\lambda} + C}{5\Theta}$$

  where the last inequality comes from the fact that $(k+1)2T > \Theta$ and so $kt > \frac{1}{2}(\Theta - 2T) > \frac{1}{2}(\Theta - 0.1\Theta)$.

  And so we get logical drift $\tilde{\rho} = \rho + \frac{16\tilde{\lambda}+C}{5\Theta}$ and discontinuity $\mu = 2D$.

The only thing left to prove Theorem 5 is to justify the bound on $\mathsf{WayOff}$. Recall that from the analysis above we get the following three constraints:

i. $2D \leq \mathsf{WayOff} - 2\rho T - \tilde{\lambda}$ (used in the proof of Claim 9).

ii. $2D \geq 16\tilde{\lambda} + 16\rho T + 16C$ (used in the proof of item i in Claim 13).

iii. $C = (\mathsf{WayOff} + 2\tilde{\lambda})/(2^k - 2)$ (used in the proof of item iii in Claim 13).

Combining these three constraints we get that such a constant $D$ exists if and only if

$$16\tilde{\lambda} + 16\rho T + 16(\mathsf{WayOff} + 2\tilde{\lambda})/(2^k - 2) \ \leq \ \mathsf{WayOff} - 2\rho T - \tilde{\lambda}$$

Solving the last inequality for $\mathsf{WayOff}$ we get $\mathsf{WayOff} \geq 18\left(1 + \frac{16}{2^k-18}\right)\rho T + 17\left(1 + \frac{17.882}{2^k-18}\right)\tilde{\lambda}$ (where $2^k - 18$ has to be positive, which implies $k \geq 5$). ∎

### 4.7 Some extensions

We conclude this section by observing that our analysis can be used virtually unchanged to prove stronger properties than what is stated in Theorem 5. Specifically, we can still prove synchronization and accuracy even if we relax somewhat the communication model.

**Relaxing the communication model.** The arguments above can be carried through even if the communication graph is not completely static, as long as the connectivity requirement in Eq. 1 is maintained. Hence, our proof extends to a model where the adversary can corrupt both processors and links, as long as "not too many of either are corrupted at once".

In this model, we still allow the adversary to corrupt upto $f$ processors in each time period of length $\theta$, and in addition is can also corrupt communication links. We define the neighborhood of a node $p$ at time $\tau$ as the set of nodes that are connected to $p$ via links that were uncorrupted throughout the interval $[\tau - \theta, \tau]$. The requirement that we need from this communication model is that at any time $\tau$, if both $p$ and $q$ are non faulty, then $|N(p) \cap N(q)| \geq 3f + 1$.

## 5 Future directions

Our results require that at most a third of the processors are faulty during each period. Previous clock synchronization protocols assuming authenticated channels (as we do) were able to require only a majority of non-faulty processors [20, 28]. It is interesting to close this gap.

In [10] there is another weaker requirement: only that subnetwork containing non-faulty processors remain connected (but [10] also assumes signatures). The connectivity requirement in our analysis is much stronger, though, as we require that the intersection of any two neighborhoods is large. It may be interesting to examine the minimum connectivity requirement under which our protocol is still correct. It should be noted that $(3f + 1)$-connectivity is not sufficient for our protocol. One can construct a graph on $6f + 2$ nodes which is $(3f + 1)$-connected, and yet our protocol does not work for it. This graph consists of two cliques of $3f + 1$ nodes, and in addition the $i$'th node of one clique is connected to the $i$'th node of the other. Now, this graph is clearly $3f + 1$ connected, but our protocol cannot guarantee that the clocks in one cliques do not drift apart from those in the other.

Additional work will be required to explore the practical potential of our protocol. In particular, practical protocols such as the Network Time Protocol [22] involve many mechanisms which may provide better results in typical cases, such as feedback to estimate and compensate for clock drift. Such improvements may be needed to our protocol (while making sure to retain provable security!), as well as other refinements in the protocol or analysis to provide better bounds and results in typical scenarios.

## References


[1] B. Barak, A. Herzberg, D. Naor and E. Shai, The Proactive Security Toolkit and Applications, *Proceedings of the 6th ACM Conference on Computer and Communication Security*, Nov. 1999, Singapore, Pages 18–27.

[2] A. Bouzelat and Z. Mammeri, Simple reading, implicit rejection and average function for fault-tolerant physical clock synchronization, *Proceedings of 23rd EuroMicro Conference*, Sep. 1997, pages 524–531.

[3] R. Canetti, R. Gennero, A. Herzberg and D. Naor, Proactive Security: Long-term protection against break-in, *CryptoBytes*, Vol. 3 No. 1, Spring 1997.

[4] R. Canetti and A. Herzberg, Maintaining Security in the Presence of Transient Faults, *Proceedings of Crypto' 94*, pages 425–438, August 1994.

[5] C.S. Chow and A. Herzberg, Network Randomization Protocol: A proactive pseudo-random generator, *Proceedings of 5th Usenix Unix Security Symposium*, Salt Lake City, Utah, June 1995, pages 55–63.

[6] R. Canetti, S. Halevi, and A. Herzberg. Maintaining authenticated communication in the presence of break-ins. *Journal of Cryptology*, to appear. Preliminiary version appeared in *Proceedings of 16'th Annual ACM Symposium on Principles of Distributed Computing*, pages 15–24. ACM Press, 1997.

[7] F. Cristian, Probabilistic Clock Synchronization, *Distributed Computing*, Vol. 3, pp. 146–158, 1989.

[8] F. Cristian and C. Fetzer, Probabilistic Internal Clock Synchronization, *Proceedings of the 13th Symposium on Reliable Distributed Systems*, Oct. 1994, Dana Point, CA. Pages 22–31.

[9] F. Cristian and C. Fetzer, An Optimal Internal Clock Synchronization Algorithm, *Proceedings of the 10th Annual IEEE Conference on Computer Assurance*, June, 1995, Gaithersburg, MD. Pages 187–196.

[10] D. Dolev, J.Y. Halpern, B. Simons and R. Strong, Dynamic Fault-Tolerant Clock Synchronization, *JACM* Vol. 42, No. 1, Jan. 1995, pp. 143–185.

[11] S. Dolev and J. Welch, Wait-free clock synchronization, *Proceedings of 12'th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 97–108. ACM Press, 1993.

[12] S. Dolev and J. Welch, Self-Stabilizing Clock Synchronization in the Presence of Byzantine Faults, *Proc. of the 2nd Workshop on Self-Stabilizing Systems*, May 1995.

[13] C. Fetzer and F. Cristian, Lower bounds for convergence function based clock synchronization, *14th PODC*, Aug 1995, Ottowa, Canada, pp. 137–142.

[14] C. Fetzer and F. Cristian, Integrating External and Internal Clock Synchronization, *Real Time Systems*, vol. 12, no. 2, pp. 123-171, Kluwer Academic Publishers, 1997.

[15] S. Haber and W.S. Stornetta How to Time-Stamp a Digital Document, *Journal of Cryptology*, 1991, Vol. 3, pages 99–111.

[16] T. Herman, Phase Clocks for Transient Fault Repair, Technical Report 99-08, University of Iowa Department of Computer Science, 1999.

[17] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk and M. Yung, Proactive public key and signature systems, *Proceedings of the 4th ACM Conference on Computer and Communication Security*, 1997.

[18] A. Herzberg, S. Jarecki, H. Krawczyk and M. Yung, Proactive Secret Sharing, or: How to cope with perpetual leakage, *Proceedings of Crypto' 95*, August 1995, pp. 339–352.

[19] S. Kutten and B. Patt-Shamir, Time-adaptive self stabilization, *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 149–158, 1997.

[20] L. Lamport and P.M. Melliar-Smith, Synchronizing clocks in the presence of faults, *JACM* Vol. 32, No. 1, Jan. 1985, pp. 52–78.

[21] J. Lundelius-Welch and N. Lynch, A new fault-tolerant algorithm for clock synchronization, *Information and Computation*, Vol. 77, No. 1, pages 1–36, 1988.

[22] D.L. Mills, Internet time synchronization: the Network Time Protocol. In *IEEE Trans. Communications*, October 1991, pp. 1482- 1493.

[23] B.C. Neuman and T. Ts'o, Kerberos: An Authentication Service for Computer Networks, *IEEE Communications Magazine*, Sep. 1994, Vol. 32, No. 9, pp. 33–38.

[24] R. Ostovsky and M. Yung, How to withstand mobile virus attacks, *Proceedings of PODC*, 1991, pp. 51–61.

[25] R. Reischuk, A new solution to the byzantine generals problem, *Information and Control*, pages 23–42, 1985.

[26] W. Rosenberry, D. Kenney and G. Fisher, Understanding DCE, Chapter 7: DCE Time Service: Synchronizing Network Time, *O'Reilly*, Oct. 1992.

[27] Fred B. Schneider, Understanding Protocols for Byzantine Clock Synchronization *Technical Report TR87-859, CS Department, Cornell University*, 1987.

[28] T.K. Srikanth and S. Toueg, Optimal clock synchronization, *JACM* Vol. 34, No. 3, July 1987, pp. 626–645.

[29] IETF Secure Time (Stime) Working Group, http://www.stime.org/.