A Peer-to-Peer Filter-Based Algorithm for Internal Clock Synchronization in Presence of Corrupted Processes[*]

Roberto Baldoni, Marco Platania, Leonardo Querzoni, Sirio Scipioni
Dipartimento di Informatica e Sistemistica "A. Ruberti"
Sapienza - Università di Roma
Rome, Italy
{baldoni|querzoni|scipioni}@dis.uniroma1.it

Abstract

This paper proposes an internal clock synchronization algorithm for very large number of processes that is able to (i) self-synchronize their local clocks without any central control and (ii) resist to attacks of an adversary whose aim is to put out-of-synchronization as many correct processes as possible. To cope with scale the algorithm utilizes the gossip-based paradigm where each process has a limited view of the system, while to resist to attacks the algorithm employs a filtering mechanism based on the notion of α-trimmed mean *to filter out out-of-range clock values. The algorithm shows nice convergence in presence of networks errors and in absence of the adversary. When the adversary takes control of some of the processes in the system, we define two goals for the adversary, actually two predicates, to measure the strength of the attack. The first one captures the percentage of time in which at least one correct is out of synchronization and the second one when all correct processes are out of synchronization. The paper presents an extensive simulation study showing under which conditions (in terms of number of corrupted processes and size of local views) these two goals can be achieved by the adversary. Interestingly, these results can be exploited by applications that can tolerate either a certain time in which some correct process is non-synchronized or a certain percentage of correct processes that is non-synchronized.*

1. Introduction

Maintaining clocks synchronized among a set of independent machines is a key problem for many distributed applications. Each machine is equipped with an internal physical clock, but clocks on distinct machines often have slightly different phases and frequencies that can lead their values to drift. Applications running on these machines can rely upon the local physical clock and a continuous exchange of message to build and maintain a synchronized logical clock. The unreliability of machines and communication links introduces another variable in the problem that limits the synchronization accuracy obtainable with this approach.

The clock synchronization problem has been widely studied for many years, and several algorithms exist which address different scales, ranging from local area networks (LAN), to wide area networks (WAN). The work presented in this paper is motivated by an emergent class of applications and services, operating in very challenging settings, for which the problem of clock synchronization is far from being solved. These applications are required to (1) operate without any assumption on deployed functionalities, pre-existing infrastructure, or centralized control, while (2) scaling from few hundred to tens of thousands of machines and (3) being able to operate in secure manner.

A promising approach to tackle this kind of problems is to embrace a fully decentralized approach in which every machine (process) implements all the required functionalities by running so called $gossip-based$ algorithms. With this approach, due to the large scale and geography of the system, each process is provided with a local limited view representing the part of the system it can directly interact with. The algorithm running at each process computes local results by collecting information from this neighborhood. These results are periodically updated leading the system to gradually compute the expected global result. Much like real-life rumor, information disseminated by a gossip protocol spreads quickly and reliably with high probability [11]. Moreover, gossip protocols can be adapted to tolerate a certain number of node crashes by adjusting the number of selected partners in each communication round. In the pres-

[*] The work described in this paper has been partially supported by a grant from Finmeccanica, by the European Network of Excellence ”Resist” and by the European Project ”CoMiFin”.

ence of failures stronger than crashes (e.g. byzantine failures), the problem of avoiding the diffusion of false information becomes the dominant factor to face for realizing a correct solution.

In this paper we introduce a filtering solution based on a well-know statistical notion, the *α-trimmed mean*, and on a nature inspired mechanism coming from the *coupled oscillators* phenomenon. The latter mechanism is employed to cope with the large scale indeed the coupled oscillators phenomenon shows enormous systems of oscillators spontaneously locking to a common phase, despite the inevitable differences in the natural frequencies of the individual oscillators. Examples from biology include network pacemaker cells in the hearth, congregations of synchronously flashing fireflies and crickets that chirp in unison. A description of the phenomenon was pioneered by Winfree [21] and extended by Kuramoto [15]. This approach was adopted in [3] to design a synchronization protocol able to deliver good synchronization accuracy in very large scale systems where every nodes either behaves correctly or crashes .

The filtering mechanism based on the notion of *α-trimmed mean*, is used to block the spread of false information injected in the system by byzantine processes. Specifically, every process drops some of the values read from other processes with the aim of eliminating all the faulty values provided by byzantine processes.

In order to evaluate our solution in a worst case scenario we modeled the testing environment assuming the presence of an adversary that can control the behavior of some processes. These corrupted processes can then cooperate with correct ones with the aim of disrupting the protocol behavior (i.e., break the synchronization). We define two goals in terms of predicates for the adversary: the first one (TP1) states that the adversary wins if one correct process has is clock value influenced by a corrupted process. The second goal (TP2) states that the adversary wins if each correct process has its local clock influenced by at least a corrupted process.

More specifically, we firstly show how our algorithm is able to maintain good convergence speed and resilience to error induced by network perturbation in absence of an adversary. Then we introduce two performance metrics, namely Error Persistence (i.e., number of synchronization rounds in which at least one node experienced an error in its local clock induced by a corrupted process with respect to the system lifetime), and Infection Index (i.e., percentage of correct processes that at each synchronization round experienced an error in its local clock induced by a corrupted process), that allow to practically asses when TP1 and TP2 become true as a function of the number of corrupted processes and the size of the view associated to each process. Doing so, our algorithm is actually highly configurable in order to obtain the desired level of resilience (according to the application needs) to attacks of the adversary, in terms of amount of time in which an error caused by corrupted processes is present in the system and the number of processes infected by the corrupted clock values.

The rest of the paper is organized as follows: Section 2 presents the system under which our algorithm works, while Section 2.1 presents the clock coupling model along with the algorithm, definition of precision achievable by a clock synchronization algorithm and of a Byzantine Resilient system. Section 3 presents the clock synchronization algorithm and provides a description of the behavior of a corrupted process while the experimental evaluation is presented in Section 4. Finally Section 5 discusses related works and Section 6 concludes the paper.

2. System Model

We consider a system constituted at any time t by a finite but unknown set of uniquely identified processes $\mathbb{P}_t$. This set can change over time. Process can either crash or be corrupted by an adversary at any time during their computation. A process that either does not crash or does not get corrupted during the entire system lifetime is considered correct. At any time t the set of correct processes will be denoted as P_t, the one of corrupted processes C_t and $\mathbb{P}_t = P_t \cup C_t$.

Each process $p_i \in \mathbb{P}_t$ has a finite set of neighbors. In the following we refer to this set as the *local view* (lv_i) of process p_i. Neighbors processes communicate by exchanging messages through point-to-point communication. Communication between correct processes is authenticated and reliable even though message transmission delays are unpredictable.

The local view of a process $p_i \in \mathbb{P}_t$ is obtained through a Peer Sampling Service[14]. We assume the processes belonging to the p_i's view provided by the Peer Sampling service, to be a uniform random sample of the system population at time t. If a process p_i crashes, the Peer Sampling Service will not include anymore p_i in any view.

We also assume that every process is equipped with a hardware clock. Depending on its quality and the operating environment, its frequency may drift. Manufacturers typically provide a characterization for ρ – the maximum absolute value for clock drift. Ignoring, for the time being, the resolution due to limited pulsing frequency of the clock, the hardware clock can be described by:

$$C(t) = ft + C_0;$$

where: $(1 - \rho) \leq f \leq (1 + \rho)$.

2.1. Adversary-free Internal Clock Synchronization

In an Internal Clock Synchronization processes belonging to the system have to cooperate in order to reach a common clock value despite the lack of an external time reference and the presence of failures, an adversary and communication uncertainty. This global clock is then instantiated at each process through a local clock value that depends on the local hardware clock and that can be adjusted thanks to the message exchanged among the processes. The goal of the Internal Clock Synchronization is to minimize the differences between local clocks of processes and to bound it to an error that will be a function of the communication delays and the view of the system of each process.

In order to define the correct behavior of our system in absence of an adversary we introduce the *Synchronization Error (SE)* that describes the precision of synchronization at time t and of *adversary-free Synchronization Range* that, otherwise, describes the precision of synchronization that a clock synchronization algorithm run by correct processes can achieve. More in particular the Synchronization Error $SE(t)$ at time t is the standard deviation, computed at time t, of clock value of node belonging to the system and a definition of Synchronization Range is the following:

Definition 1 ***Adversary-free Synchronization Range*** *Let consider I be a time interval $[t_1, t_2]$ (with $t_1 < t_2$) where $C_t = \emptyset$ for each $t \in I$. Moreover let R_I be a synchronization range dependent from a clock synchronization algorithm A and an error distribution in $[t_1, t_2]$ E. The system is synchronized in I iff:*

$$|SE_{P_t}(t)| < R_I, \forall t \in I$$

2.2 Model of the Adversary

At any time t a subset of corrupted processes C_t can get into the system under the control of an adversary. When a process is corrupted, the adversary can impose any value to the local clock of the process. The adversary obeys the following assumptions:

1. The adversary knows the Synchronization Range R_I for any possible time interval I;

2. The adversary is able to coordinate behavior of corrupted processes using out-of-band communications. Therefore the adversary is able to coordinate a common *strategy* including a common starting point for the attack and let corrupted processes impose a common local clock value.

The adversary has two goals to achieve expressed by the following *target predicates*:

(TP1) $\exists t \in [t_1, t_2] : \forall t' \in [t, t_2] \Rightarrow (\exists p \in P_{t'} : |SE_{P_{t'}}(t')| \geq R_I$ and $|SE_{P_{t'}-\{p\}}(t')| < R_I)$
This predicate states that the adversary wins (i.e., TP1 becomes true) when, starting from a certain time instant t there will always be at least a correct process in the system whose clock value is infected by the adversary and the synchronization error fails to remain within the Synchronization Range.

(TP2) $\exists t \in [t_1, t_2] : \forall t' \in [t, t_2] \Rightarrow$
$(\nexists k \in (0, |P_{t'}|] : \forall \{p_{i_1}, \ldots, p_{i_k}\} \in P_{t'} \Rightarrow (|SE_{P_{t'}}(t')| \geq R_I$ and $|SE_{P_{t'}-\{p_{i_1},\ldots,p_{i_k}\}}(t')| < R_I))$
This predicate states that the adversary wins (i.e., TP2 becomes true) when, starting from a certain time instant t, the local clock of every correct process in the system is infected by the adversary. In this last case there will be no two correct processes in the system with synchronized clocks.

Let us remark that TP2 implies TP1 and the viceversa is not necessarily true.

Finally the assumption of correctness of the Peer Sampling Service follows that also Eclipse attack are not possible for the adversary. In an Eclipse attack [6, 20], a modest number of corrupted processes conspire to fill the local view of a correct process with corrupted ones, with the goal of dominating the neighbor set of each correct process.

3 The Filter-Based Algorithm

We present a solution inspired by the algorithm in [3] where it is proposed a fully decentralized algorithm able to maintain Internal Synchronization in large scale systems. We develop a new clock synchronization algorithm through the introduction of a static filtering based on well-know theory of *α-trimmed mean*. Using this filtering mechanism, each node discards from its view, of size n, $m = (U_i - L_i)$ elements, where $L_i = \lfloor \alpha_m n \rfloor$, $U_i = n - L_i$, and $\alpha_m \in [0, 0.5)$. In the following we assume that the difference between neighboring clocks are estimated as NTP does [18]. Under this assumption, the real offset is such as the error is bounded by $\pm RTT/2$, where RTT is the round trip time but, as it is showed in [3], the error strictly depends from channel asimmetry.

3.1 Behavior of a correct process

Using this solution the filter-based clock synchronization algorithm performs at each round ℓ the following steps in a correct process p_i:

1. Asks to Peer Sampling Service a random list of neighbours.

2. Evaluates the difference with every neighboring clock, using the Remote Clock Reading Procedure.

3. Sorts the offset respect its neighbours in a ordered list O_i

4. Computes new clock by mean of the equation

$$C_i((\ell+1)\Delta T) = C_i(\ell\Delta T) + f_i\Delta T + \\ + \frac{1}{U_i - L_i}\sum_{k=L_i}^{U_i}[O_i(k)] \quad (1)$$

5. Update the value of C_i.

Where ΔT is the time interval between two consecutive synchronization rounds.

3.2 Behavior of a corrupted process

At any time t, a process belonging to C_t executes the same algorithm of a correct one but it adds/subtracts a value $E_B(t)$ at the local clock computed at step 4 of the filter-based algorithm. In the following we assume that all corrupted processes add/subtract the same value to the their clock which results in the worst attack from an adversary point of view as different added or subtracted values could inherently reduce the strength of the attack. Considering that the adversary knows R_I (see Section 2.2), $E_B(t)$ is computed by the adversary in order that if a correct processes computes its new clock value including the value of a corrupted process, the synchronization error fails to remain within the Synchronization Range. Finally we say that a correct process is *infected* when a correct process either reads a value from at least one corrupted process (i.e., the filter is not able to discard all corrupted values) or it reads a value from at least one infected process.

4 Evaluation

In this section we evaluate the behavior of the proposed filter-based algorithm. We first discuss its behavior in an adversary-free scenario where its performance can only be affected by transmission delays. Through this first step we show how, in this setting, the algorithm is able to quickly synchronize clocks of a large set of processes with an error that strictly depends from the underlying communication network characteristics.

In a second phase we evaluate the algorithm behavior in a setting where an adversary tries to achieve its goals (i.e., TP1 and TP2) by coordinating of a certain number of corrupted processes. This part of the evaluation assesses how, under certain conditions, the filtering mechanism is able to rule-out the influence of corrupted processes and maintain all (or just a part of) correct processes synchronized.

The evaluation was realized through a simulation-based study conducted on the Peersim [1] simulator.

4.1 Evaluation Metrics

Throughout this section we consider to the following metrics to evaluate the algorithm behavior:

Synchronization Error. The synchronization error (SE) is the standard deviation of the various processes' clock values. In an ideal setting this value should be zero, i.e. all clocks are perfectly synchronized on a same value.

Convergence Time. The Convergence Time is the number of synchronization rounds (SR) needed for the system Synchronization Error to fall below a desired target value. If not differently stated, the Synchronization Error target value considered in out tests was $10\mu sec$.

Error Persistence. The Error Persistence is defined as the fraction of synchronization rounds in which at least a correct process is infected from a corrupted one. Intuitively, this metric measures the ability of the adversary to continuously maintain at least part of the system under its influence; when Error Persistence is 1, at each round at least one correct process in the system is infected and, therefore, the adversary succeed in reaching the goal defined by TP1.

Infection Index. The Infection Index at time t is defined as the ratio between the number of correct processes infected by corrupted ones and the total number of correct processes P_t. Intuitively, this metric measures the ability of the adversary to influence a large fraction of the system; when the Infection Index is 1, no two correct processes exist in the system with synchronized clocks and, therefore, the adversary succeed in reaching the goal defined by TP2.

4.2 Evaluation Scenarios

Metrics have been measured in two relevant scenarios. These scenarios were defined differently in order to isolate specific aspects relevant to the metrics under exam.

Adversary-free scenario - This scenario models a system in which the set of participating processes is static (no process is added/removed during the whole system lifetime), the network delay is bounded but unknown and there is no adversary trying to infect processes (i.e. $\forall t, \mathbb{P}_t = P_t$). Every process executes the filter-based algorithm every ΔT time units, but not in a lock step mode.

Network channel delays are modeled by means of a Gaussian distribution whose mean and standard deviation have been derived by fitting several round-trip data set measured over the Internet [4]. More specifically, in this scenario we consider *slow channels* characterized by an average round trip delay of $180msec$, and *fast channels* characterized by an average round trip delay of $30msec$. In order to evaluate the Convergence Time and the Synchronization Error we considered the communication channels symmetric and asymmetric respectively. Channel asymmetry defines how the round-trip time (RTT) experienced by processes is distributed between the two directions of a communication channel. Given a channel connecting process A to process B, channel asymmetry is defined as the ratio between the transfer delay of a message from A to B and the delay back from B to A. Limitedly to tests where asymmetric channels are assumed, the asymmetry is modeled by means of a Gaussian distribution with mean 0.5. The parameters of this distribution are used in order to explore the sensitivity of the algorithm to channel asymmetry, and thus to measure the corresponding Synchronization Error.

The algorithm round duration ΔT was defined, for the purposes of these tests, as the third quartile of the slow channels RTT Gaussian distribution.

Scenario with the adversary - This scenario models a system in which the set of participating processes is static, the network delay is bounded but unknown and an adversary is present that tries to influence the system through a set C_t of corrupted processes (i.e. $\forall t, \mathbb{P}_t = P_t \cup C_t$). In this scenario, in order to better characterize the dependency of the Synchronization Error from the presence of corrupted processes, we ignore the estimation errors on clock differences caused by the communication channels, i.e. we assume symmetric channels.

4.3 Results for the Adversary-Free Scenario

In this first evaluation phase, we want to investigate if a system running the filter-based algorithm is able to reach a target synchronization range R. This evaluation is conducted in the previously introduced adversary-free scenario. In order to evaluate the behaviour of the proposed algorithm we can refer to well-known results from statistics. Our algorithm, in fact implements a *α-trimmed mean*. Previous works on the trimmed mean [8] show that its variance depends from the distribution of samples in the considered population. When a normal distribution or similar lighter-tail distributions are considered the variance of the sample mean is a lower bound for the variance of the trimmed mean. Otherwise, when heavy-tail distributions are considered the variance of the sample mean is an upper bound for the variance of trimmed mean. In the latter case results obtained applying the trimmed mean improve by increasing the value of α_m (i.e., the percentage of discarded samples) with respect to those obtainable by means of a sample mean.

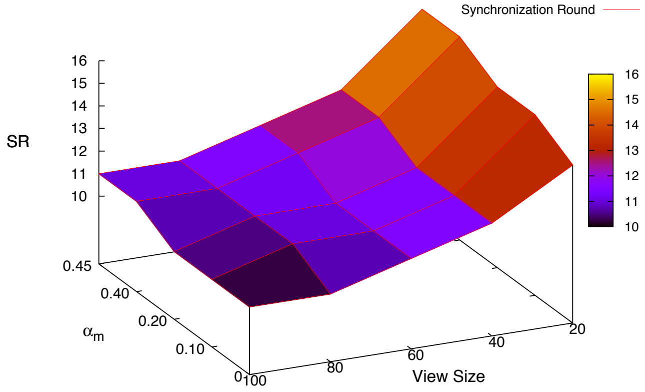


Figure 1. Convergence Time dependency from view size and α_m ($|\mathbb{P}_t| = 64000$)

Figure 1 reports the Convergence Time of the filter-based algorithms when varying both the percentage of discarded samples α_m and the local view size lv_i. These results refer to a system constituted by $\mathbb{P}_t = P_t = 64000$ processes running over a network characterized by perfectly symmetric channels. Clock values have been initially distributed uniformly on the range [0,60] The graph shows how Convergence Time increases with α_m and, on other hand, decreases as the view size lv_i increase. This is coherent with the previously described trimmed mean properties. For a rectangular distribution, in fact, the trimmed mean has a variance larger than the one of the corresponding sample mean [19]. Consequently, increasing α_m, also increases the Convergence Time needed to reach the predefined target synchronization range $R = 10^{-8}$ [1]. Despite this, differences in the Convergence Time are limited; consequently, large values for α_m can be considered acceptable.

Asymmetric channels Figure 2 reports the Convergence Error obtained by filter-based algorithms when varying both the percentage of discarded samples α_m and the local view size lv_i. These results refer to a system constituted by $|\mathbb{P}_t| = |P_t| = 64000$ processes running over a network characterized by asymmetric channels. Clock values have been initially distributed uniformly on the range [0,60]

[1] The synchronization range was set close to 0 because, when channels are perfectly symmetric communication delays have no influence on the Synchronization Error (see Equation 1).

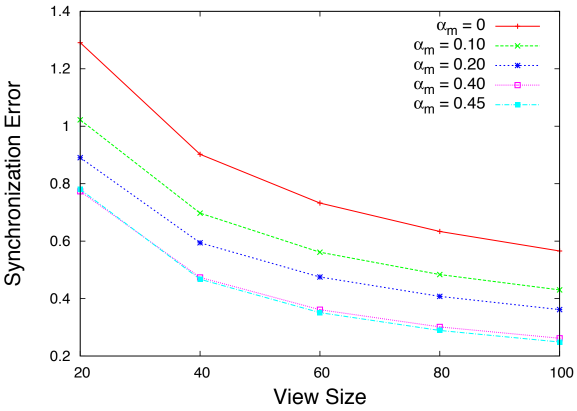


Figure 2. Synchronization Error dependency from view size and α_m ($|\mathbb{P}_t| = 64000$)

Due to the different communication channel model adopted in this test, errors in clock readings show a heavy-tail distribution. This distribution causes the trimmed mean to produce values with a variance smaller than the one of the corresponding sample mean. Consequently, increasing the value of α_m, the Synchronization Error decreases, coherently with the result showed in [19]. Our results, shown in Figure 2, confirm this behaviour and show that for this setting an upper bound on the Synchronization Error can be obtained with $\alpha_m = 0$.

From the tests conducted in the two considered adversary-free scenarios we can conclude that the filter-based algorithm is able to converge to a target synchronization range, as long as the local view size of every process and the percentage of discarded elements α_m are correctly configured. Moreover, large values of α_m are acceptable as they affect negatively only for a small factor the Convergence Time but considerably improve the resilience of system to channel errors (i.e. reduce the Synchronization Error).

4.4 Results for the Scenario with the Adversary

Let us first note that the filter-based algorithm is able to deterministically discard all values read from corrupted processes when each local view includes all processes in $\mathbb{P}_t$ and if $\alpha_m \cdot |\mathbb{P}_t| > |C_t|$ (i.e. the number of discarded elements is greater than the number of corrupted processes). In this ideal case the behavior of the proposed algorithm with respect to the resilience to corrupted processes resembles the behavior of Cristian and Fetzer's algorithm [7] with respect to the resilience to byzantine faults. Roughly speaking, this happen if we can configure our algorithm to always discard a number of read values greater than the maximum number of concurrent corrupted clocks.

In more realistic setting where local view size must be limited (and this is usually much smaller than the system size), or where we do not know the maximum number of processes corrupted by an adversary, the probability of discarding all values read from corrupted processes depends from the local view size, from the percentage of discarded elements α_m and, last but not least, from the ratio of corrupted processes currently present in the system.

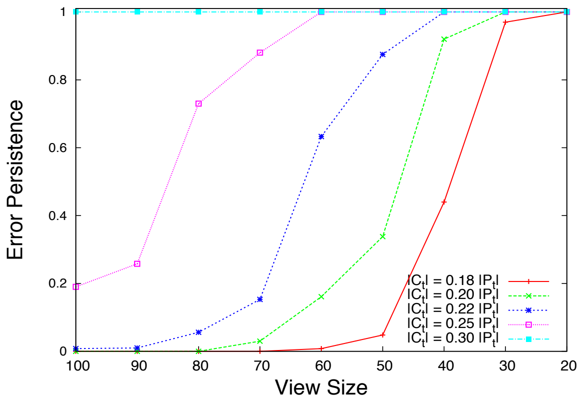
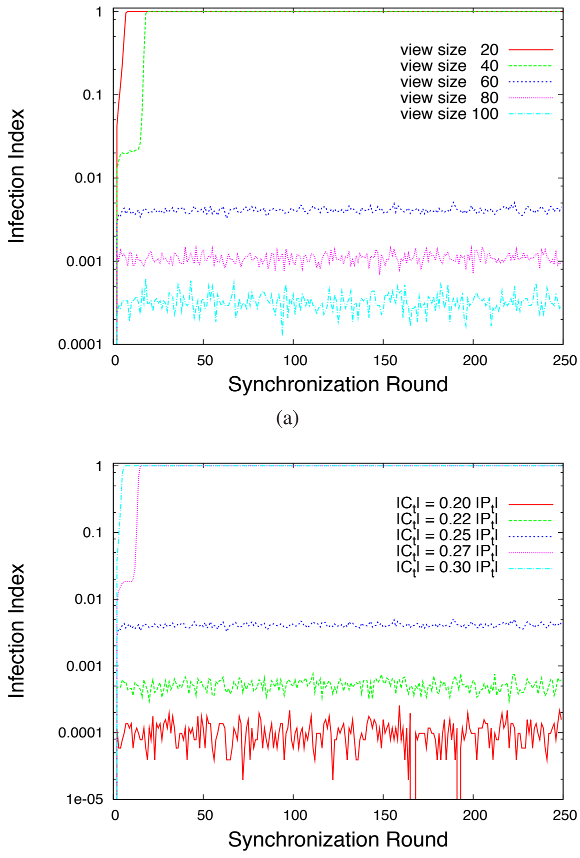


Figure 3. Error Persistence dependency from the view size and the amount of corrupted processes($|P_t| = 64K$, $\alpha_m = 0.45$)

Error Persistence. Figure 3 reports the Error Persistence of the system when varying both the local view size lv_i and the number of processes corrupted by the adversary. These results refer to a system constituted by $P_t = 64000$ correct processes running over a network characterized by perfectly symmetric channels. The curves show that, given a certain amount of corrupted processes, the algorithm can be conveniently configured in order to attain a desired Error Persistence level. If the number of corrupted processes is too large (e.g. 30% in the plot), even using large local views, the filter based values is not able to discard all values read from corrupted processes. When this happens the Error Persistence reach its maximum value (i.e. 1), and the adversary reach its first goal(i.e.,TP1 flips to true) as it is able to continuously influence the system Synchronization Error.

Maintaining the Error Persistence below a certain threshold can be exploited by applications that can tolerate that correct processes will be synchronized for at a certain amount of time. For this class of applications, our clock synchronization algorithm can be useful because Error Persistence can be configured through the view size: the larger the view is, the lower the error will be. This means that by increasing the view size, we enlarge the percentage of time in which all correct processes are synchronized.

Infection Index. The Error Persistence metric does not give any hint about the percentage of correct processes infected by values proposed by the adversary. From this point of view evaluating the Infection Index is very important, because it allows us to asses which fraction of correct processes is affected by the adversary when Error Persistence is equal to 1 (i.e., when TP1 is true). This assessment can be used by applications that can tolerate a clock synchronization service in which some fraction of correct processes is infected by the adversary.



(a)

(b)

Figure 4. Evolution of the Infection Index depending from (a) the view size and (b) the amount of corrupted processes ($|P_t| = 64000$, $\alpha_m = 0.4$)

Figure 4 reports the evolution during various synchronization rounds of the Infection Index when varying the local view size (Figure 4(a)) or the number of processes corrupted by the adversary (Figure 4(b)).

Note that these results are obtained with α_m smaller than in previous tests in order to work in a settings where the Error Persistence is always 1. Figure 4 shows a bimodal behavior. There exists a threshold for both view size and number of corrupted processes after which the whole system becomes infected. In this case the number of correct processes infected is sufficiently large to be comparable with the number of corrupted processes. In this scenario the probability of infection, for a non-infected correct process, grows in each round and rapidly the infection becomes incontrollable. When the system works below this threshold we can tune the local view size in order to limit the fraction of the system influenced by the adversary under a desired level. Let us remark that when all the correct processes becomes infected (infection index equal to one) the adversary reaches its second goal, i.e., TP2 flips to true.

Another important observation from the plots of Figure 4 is the following: if the number of corrupted processes is below the threshold that leads to the whole system infection, the percentage of the correct processes that read a value of a corrupted process at any round can be bounded by a constant number. For example, if we consider the plot depicted in Figure 4(b) with $C_t = 0.27 * P_t$, this constant can be 0.01, therefore at most only one process out of one hundred will be out of synchronization at any time. Therefore this bound can be exploited by applications that can tolerate such bound of non-synchronized correct processes. Finally, the plot in Figure 5 is a compact representation of Figure 4(a)) and Figure 4(b) where the dependency among the infection index and the view size and the number of corrupted processes is highlighted. In this plot we show the average infection index of the system computed with respect to a whole simulation after a transitory of 50 synchronization rounds, in which the system stabilizes. From these plots it is very easy to compute a bound of corrupted processes when view size and number of corrupted processes has been fixed.

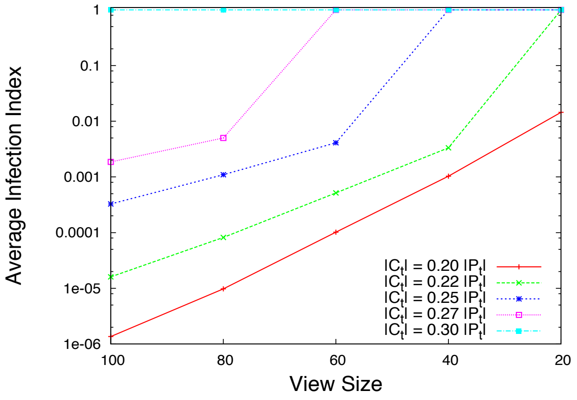


Figure 5. Infection Index dependency from the view size and the number of corrupted processes ($|P_t| = 64000$, $\alpha_m = 0.4$)

5 Related Work

There is very large body of work on clock synchronization. Below we focus on the most relevant works.

Since the seminal paper of Lamport [16], many papers addresses the problem of clock synchronization resilient to faulty processes. Differently from our approach, these protocols are based on the complete knowledge of the processes forming the system and on having a clique as underlying communication graph (e.g. [16, 17, 5, 7, 10]) where processes can either execute point-to-point or broadcast communication primitives (e.g. [12]). Moreover, these protocols require each process reads the local clock of every other node in the system and, when handling byzantine failures, there is a predefined bound on the number of correct processes in the system in order that the clock synchronization is correct (e.g., [9, 17, 7]. The interaction between one process and all the other processes in the system would limit severely the scalability of the algorithm in a large scale setting, this is why in our approach each process interacts only with the processes in its view. In this paper we do not show a formal proof of correctness of convergence of clock synchronization, rather we asses the impact of corrupted processes on the effectiveness of clock synchronization. In particular we estimate, considering a certain view size and a number of corrupted processes, (i) the percentage of time in which at least one correct process reads from corrupted processes and becomes infected (i.e., interval of time in which TP1 is true with respect to the system lifetime) and (ii) the percentage of correct processes that becomes infected (i.e., interval of time in which TP2 is true with respect to the system lifetime).

Differenly from our peer-to-peer dynamic approach, NTP uses a master-slave approach in a static and manually-configured hierarchical topology[18]. In this hierarchy, the primary time servers are directly connected to an external reference time source (GPS, atomic clock, etc.) and are the roots of the spanning tree used to diffuse clock values. Secondary time servers are represented by inner nodes in the hierarchy, and they synchronize their clock communicating with one or more primary time servers. In order to be byzantine-fault tolerant NTP utilizes certification authorities, authentication of primary and secondary time servers and secure cryptographed communications.

Peer-to-peer solutions to clock synchronization based on a gossip-based protocol are a recent research field. We know only three works that use this approach ([13, 2, 3]) and none of them do not present an analysis of their behaviour in a byzantine environment. All of them rely on a peer-sampling service and [13] is an example of external clock synchronization while [2] and [3] implements internal clock synchronization.

The model of the adversary used in [5] is closer to ours. They define an adversary that can corrupt processes and orchestrate attacks and the power of the adversary depens on the number of processes it controls. Differently from our model of adversary, their model assumes that corrupted processes can recover.

6 Concluding Remarks

Clock synchronization for distributed systems is a fundamental problem that has been widely treated in the literature. Today's large scale distributed applications, deployed on WANs like Internet, pose new issues in terms of security and scalability that are hardly addressed by existing solutions. These applications thus require the development of new clock synchronization approaches able to reach satisfying level of synchronization while providing the desired level of scalability and security also under attacks. In this paper we presented a peer-to-peer filter based solution to clock synchronization. The peer-to-peer aspect adds scalability to the solution while the filtering capability improve the resistance to attack of an adversary that try to degrade the level of synchronization reached by correct processes. We defined two goals for the adversary that capture two distinct levels of disruption of the system synchronization. The first goal, namely TP1, characterizes the percentage of time in which at least one correct process reads from corrupted processes and becomes infected and the second goal, namely TP2, characterizes when all correct processes become infected.

The simulation campaign pointed out that decreasing the view size associated to each process, it decreases the number of corrupted processes that can be tolerated by the set of correct processes to keep a desired level of synchronization for a certain percentage of time. Moreover, interestingly, the plots showed that if the number of corrupted processes is below a given threshold, the percentage of the correct processes that read a value of a corrupted process at any round can be bounded by a constant number. These bounds can be used by applications that can tolerate at any time such bound of non-synchronized correct processes.

References

[1] Peersim: A peer-to-peer simulator. http://peersim.sourceforge.net.

[2] O. Babaoglu, T. Binci, M. Jelasity, and A. Montresor. Firefly-inspired heartbeat synchronization in overlay networks. In *Proceedings of First IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 77–86, 2007.

[3] R. Baldoni, A. Corsaro, L. Querzoni, S. Scipioni, and S. Tucci-Piergiovanni. An adaptive coupling-based

algorithm for internal clock synchronization of large scale dynamic system. In *OTM Conferences*, pages 701–716, 2007.

[4] R. Baldoni, C. Marchetti, and A. Virgillito. Impact of wan channel behavior on end-to-end latency of replication protocols. In *Proceedings of European Dependable Computing Conference*, 2006.

[5] B. Barak, S. Halevi, A. Herzberg, and D. Naor. Clock synchronization with faults and recoveries. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, pages 133–142, 2000.

[6] M. Castro, P. Druschel, A. Ganesh, and A. Rowstron D. S. Wallach. Secure routing for structured peer-topeer overlay networks. In *Proceedings of the 5th symposium on Operating Systems Design and Implementation*, pages 299–314, 2002.

[7] F. Cristian and C. Fetzer. Integrating internal and external clock synchronization. *Journal of Real Time Systems*, 12(2):123–171, 1997.

[8] S. Csorgo, E. Haeusler, and D. M. Mason. The asymptotic distribution of trimmed sums. *The Annals of Probability*, 16(2):672–699, 1988.

[9] A. Daliot, D. Dolev, and H. Parnas. Linear time byzantine self-stabilizing clock synchronization. Technical Report TR2003-89, Schools of Engineering and Computer Science, The Hebrew University of Jerusalem, 2003.

[10] S. Dolev. Possible and impossible self-stabilizing digital clock synchronization in general graph. *Journal of Real-Time Systems*, 12(1):95–107, 1997.

[11] P. T. Eugster, R. Guerraoui, A.M. Kermarrec, and L. Massouliè. Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5):60–67, 2004.

[12] J. Halpern, B. Simons, R. Strong, and D. Dolev. Fault-tolerant clock synchronization. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 89–102, 1984.

[13] K. Iwanicki, M. van Steen, and S. Voulgaris. Gossip-based synchronization for large scale decentralized systems. In *Proceedings of the Second IEEE International Workshop on Self-Managed Networks, Systems and Services*, pages 28–42, 2006.

[14] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 79–98, 2004.

[15] Y. Kuramoto. *Chemical oscillations, waves and turbulence*, chapter 5. Springer-Verlag, 1984.

[16] L. Lamport. Time, clocks and ordering of events in a distributed system. *Commun ACM*, 21(7):558–565, 1978.

[17] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, 1985.

[18] D. L. Mills. Network time protocol version 4 reference and implementation guide. Technical Report 06-06-1, Electrical and Computer Engineering, University of Delaware, 2006.

[19] P. Prescott. A selection of trimming proportions for robust adaptive trimmed means. *Journal of the American Statistical Association*, 73(361):133–140, 1978.

[20] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against eclipse attacks on overlay networks. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, 2004.

[21] A. T. Winfree. Biological rhythms and the behavior of populations of coupled oscillators. *Journal of Theoretical Biology*, 28:327–374, 1967.