Byzantine Fireflies

Rachid Guerraoui, Alexandre Maurer

Byzantine Fireflies

Rachid Guerraoui and Alexandre Maurer

EPFL, Switzerland
rachid.guerraoui@epfl.ch
alexandre.maurer@epfl.ch

Abstract. This paper addresses the problem of synchronous beeping, as addressed by swarms of fireflies. We present Byzantine-resilient algorithms ensuring that the correct processes eventually beep synchronously despite a subset of nodes beeping asynchronously. We assume that $n > 2f$ (n is the number of processes and f is the number of Byzantine processes) and that the initial state of the processes can be arbitrary (self-stabilization). We distinguish the cases where the beeping period is known, unknown or approximately known. We also consider the situation where the processes can produce light continuously.

1 Introduction

Biologically inspired algorithms have become increasingly popular in the last decades [21]. This field is motivated by fascinating emerging phenomena in nature: swarms of simple individuals (cells [15], ants [3], cuckoos [7], bats [13], ...) that seem to achieve a very consistent and regular behavior without centralized control and with very limited communications. It is appealing to design distributed algorithms reproducing their behavior with minimal communication assumptions.

One of these phenomena is the synchronization of fireflies [18, 19]. Fireflies are insect that can produce flashes of light at night. They can do so synchronously and with a regular period. Our interest here is to recreate this phenomena in the field of distributed computing.

At first sight, this problem has similarities with the problem of clock synchronization [9, 4, 8], which can also be declined for wireless ad hoc networks [16] and simultaneous-action synchronization problems [20]. These problems are however typically studied in message passing systems: the processes can identify each other and send semantically rich messages with timestamps. Such strong communication assumptions do not seem to be available in a fireflies swarm. In this paper, we therefore consider minimal communication primitives.

The fireflies synchronization problem has first been studied from a mathematical point of view [12, 14]: the individuals are represented as dynamical oscillators, and the problem is modeled as a system of differential equations. Another model, more related to the field of distributed computing, proposed in [2, 11, 17], involves processes that can produce discrete *beeps* at arbitrary moments, and must eventually beep synchronously.

Solutions to these problems [2, 11, 17] use averaging methods to achieve synchronous beeping. These solutions are efficient, but are also very sensitive to incorrect processes, which can easily move forward the mean value. Therefore, if there is no limit on the frequency of malicious beeps, one single incorrect process is sufficient to prevent synchronization.

In this paper, we consider that some fireflies may have an incorrect behavior: they can be broken, dead, ill, or trying to eat each other [1]. Our motivation is the intuition that a biological system should be inherently resilient to such malfunctions. We thus consider the problem of synchronous beeping in the presence of malicious (*Byzantine*) processes. A first solution relaxes the requirement to "beeping in a bounded interval" [5]. However, we would like to preserve perfect synchronous beeping here. In order to tolerate malicious beeps, our strategy is not based on averaging methods, but on the number of simultaneous beeps and the delay between two groups of beeps.

Our contribution. We consider a system where each process can produce *discrete* and *anonymous* beeps (flashes of light). Each process can "see" all beeps, and count the number of beeps produced at a given instant. The processes can however not distinguish the authors of the beeps, and no other communication is allowed between the processes. We consider the most general failure model: *Byzantine* failures [10], where the failing processes have a totally arbitrary behavior. Among the n processes, at most f are Byzantine. We assume that $n > 2f$ (we show that this condition is necessary in Section 3).

We consider the context of *self-stabilization* [6]: the initial state is arbitrary – that is, each process has initially memorized an arbitrary sequence of beeps. This assumption encompasses any chaotic sequence of events occurring before the synchronization (for instance, some processes can join and leave the system, which is usually the case in swarms of insects). Then, we show how to achieve synchronous beeping: all correct processes beep simultaneously and with the same period, whatever the behavior of Byzantine processes may be.

We present the model and the problem in Section 2, and show that the condition $n > 2f$ is necessary in Section 3. Then, we present four algorithms for this problem:

- We first give two algorithms for the cases where the desired beeping period is known (Section 4) and unknown (Section 5).
- Then, we give an algorithm for the case where each process has an approximate knowledge of the desired period (Section 6), which may be the case of insects. Thus, the correct processes must agree on a same period, *and* this period must be in the range of the desired one. This problem is more difficult, as correct processes are disorganized while Byzantine processes are perfectly coordinated. In this part, we relax the self-stabilization property (we give an impossibility result) and assume that $n > 3f$.
- Finally, we consider an alternative model where processes can produce light continuously. We give an algorithm with approximate period knowledge that does not relax the two aforementioned properties (Section 7).

These solutions show that synchronous beeping is feasible even in the presence of adversaries with an unbounded power. They could be adapted for clock synchronization with minimal communication assumptions.

2 Model and Problem

In this section, we state the distributed system model and the problem.

Communication model. We consider a distributed system of n processes and a continuous time domain. A process can *beep* at any time t. A beep is discrete, and is entirely described by its time position t (we will revisit this assumption in Section 7). No other communication than beeping is available, and the beeps are anonymous.

For any time t, let $S(t)$ be a multiset containing the time of each previous beep ($\forall t' \in S(t)$, $t' < t$). If m processes beep simultaneously at time t', then t' appears m times in $S(t)$ (for instance, if $S(t) = \{t_1, t_2, t_2, t_2, t_3\}$ with $t_1 < t_2 < t_3$, 3 processes beep simultaneously at time t_2). For $S(t) = \{t_1, t_2, t_3, \dots\}$, let $S'(t) = \{t - t_1, t - t_2, t - t_3, \dots\}$.

For any time t, each process knows the set $S'(t)$. In other words, each process only knows the position of the previous beeps relatively to the current time: there is no common time origin (otherwise, the problem would be trivial and would not correspond to swarms of fireflies). The processes can count the beeps at a given instant, but cannot distinguish the authors of the beeps. We do not consider any memory restriction that would prevent the processes from knowing $S'(t)$ entirely. No other form of memory (such as internal variables) is available.

We denote by $m(t)$ the number of processes beeping at time t. The beeps are strictly discrete: there exists no time interval $[t_1, t_2]$ with $t_1 < t_2$ such that, $\forall t \in [t_1, t_2]$, $m(t) \neq 0$. Therefore, there must always be a time interval between two successive and non-simultaneous beeps.

Correct and Byzantine processes. At most f processes are *Byzantine*, and may exhibit an arbitrary behavior. The other processes are *correct*, and follow the algorithm assigned to them. We assume that $n > 2f$. All correct processes follow the same algorithm.

Self-stabilization. Our objective is to achieve synchronization despite any arbitrary initial state. Therefore, we assume the previous model encompasses such an arbitrary initial state.

Let t_0 be a given time, unkown to the correct processes. We assume that, before t_0, each correct process has memorized an arbitrary sequence of beeps (which may be different for each process). Then, starting from t_0, all correct processes register the same beeps.

More precisely, for any process p, let $A(p) = \{t_1, t_2, t_3, \dots\}$ be an arbitrary set with repetition such that, $\forall t \in A(p)$, $t < t_0$. $A(p)$ represents the arbitrary sequence of beeps memorized by p before t_0. For any time t, let $A'(p, t) =$

$\{t - t_1, t - t_2, t - t_3, \dots\}$. Then, we now assume that for any time t, p knows $S'(t) \cup A'(p,t)$.

An algorithm ensuring a given property in such a context (the initial state is arbitrary) is *self-stabilizing* [6]. In particular, it can represent the fact that some processes join and leave the system arbitrarily before t_0.

Problem. Let T be any time period. We say that the processes achieve *synchronous beeping* at time t if, starting from time t, all correct processes beep and only beep at time t, $t+T$, $t+2T$, $t+3T$, ...

3 Lower bound

Is this section, we show that it is necessary to have a strict majority of correct processes ($n > 2f$) to solve the problem.

Theorem 1. *An algorithm can only ensure synchronous beeping if $n > 2f$.*

Proof. Suppose the opposite: there exists an algorithm ensuring synchronous beeping with $n \leq 2f$. In particular, let us suppose that $n = 2f$.

We first show that for any initial state, there exists a time period T' and a time t_1 such that, $\forall t \geq t_1$, the behavior of the correct processes at time t (that is, their decision to beep or not) only depends of the time interval $]t - T', t[$. Suppose the opposite. Then, there exists an initial state such that $\forall t \geq t_0$, there exists $t' \geq t$ such that two correct processes do not have the same behavior at time t'. Therefore, at time t', at least one correct process beeps and at least one correct process does not beep. Thus, as synchronous beeping requires all correct processes to have the same behavior after a time $t \geq t_0$, we do not have synchronous beeping: contradiction. Thus, there exists such a time t_1 and such a time period T'.

Let $t, t+T, t+2T \dots$ be the times of synchronous beeping. Let us suppose that the Byzantine processes beep at times $t+T/2, t+3T/2, t+5T/2 \dots$ Let i be an integer such that $iT > T'$ and $t + iT \geq t_1$. As we have synchronous beeping, all correct processes must beep at time $t + (i+1)T$. As f processes beep a time $t, t+T/2, t+T, t+3T/2 \dots$, the interval $]t' - T', t'[$ contains exactly the same beeps for $t' = t + (i+1)T$ and for $t' = t + (i+1)T + T/2$. Thus, all correct processes also beep at time $t + (i+1)T + T/2$, and we do not have synchronous beeping: contradiction.

4 Known Beeping Period

In this section, we assume that all correct processes know the same time period T. We give an algorithm ensuring synchronous beeping in this setting.

4.1 Algorithm (Known Period Synchronous Beeping - KPSB)

A correct process beeps at time t if at least one of the two following conditions is true:

1. $\forall t' \in]t-T, t[,\ m(t') = 0$
2. $(m(t-T) \neq 0) \wedge (\forall t' \in]t-T, t[, m(t') \leq f)$

4.2 Informal description

The KPSB algorithm performs in three steps:

- If no process beeps, then eventually, some correct process beeps (condition 1 of the algorithm).
- Then, T time units after a beep, all correct processes beep (condition 2 of the algorithm).
- Finally, when at least $f+1$ processes beep at the same time, the correct processes wait T time units and beep (condition 2 of the algorithm). Then, we have synchronous beeping.

4.3 Correctness proof

Lemma 1. *There exists $t \geq t_0$ such that $m(t) \neq 0$.*

Proof. Suppose the opposite: $\forall t \geq t_0,\ m(t) = 0$. Then, according to condition 1 of the algorithm, a correct process eventually beeps: contradiction.

Lemma 2. *There exists $t \geq t_0$ such that $m(t) > f$.*

Proof. Suppose the opposite: $\forall t \geq t_0, m(t) \leq f$. According to Lemma 1, there exists a time $t' \geq t_0$ such that $m(t') \neq 0$. Then, according to condition 2 of the algorithm, all correct processes beep at time $t'+T$, and $m(t'+T) > f$: contradiction. Thus, the result.

Theorem 2. *Algorithm KPSB ensures synchronous beeping.*

Proof. According to Lemma 2, there exists a time t such that $m(t) > f$. Then, according to condition 2 of the algorithm, no correct process can beep in $]t+T[$.

Suppose that a correct process p does not beep at time $t+T$. Consider the point of view of p. Then, according to condition 2 of the algorithm, there exists $t' \in]t, t+T[$ such that $m(t') > f$. Thus, as no correct process beeps in $]t+T[$, at least $f+1$ Byzantine processes beep at time t': contradiction. Thus, all correct processes beep a time $t+T$, and $m(t+T) > f$.

Therefore, by induction, we achieve synchronous beeping at time $t+T$.

5 Unknown Beeping Period

We now assume that no common time period is initially known to the processes. We thus give an algorithm where the correct processes achieve synchronous beeping after agreeing on a same period. Note that, as the processes do no have any common time metric, this can represent the case where the processes have a different perception of time.

5.1 Preliminaries

The algorithm makes use of the following predicates.

Let t be any time, let $t_1 < t$, and let $t_2 < t_1$. We define the following predicates:

- $C_1(t, t_1, t_2) \equiv (m(t_1) > f) \wedge (m(t_2) > f) \wedge (\forall t' \in]t_2, t_1[\cup]t_1, t[, m(t') \leq f)$
- $C_2(t, t_1, t_2) \equiv (m(t_1) \neq 0) \wedge (m(t_2) \neq 0)$

For any time t and $\forall i \in \{1, 2\}$, we also define the following predicates:

- $now_i(t)$: there exists $t_1 < t$ and $t_2 < t_1$ such that $C_i(t, t_1, t_2)$ is true and $t - t_1 = t_1 - t_2$
- $wait_i(t)$: there exists $t_1 < t$ and $t_2 < t_1$ such that $C_i(t, t_1, t_2)$ is true and $t - t_1 < t_1 - t_2$

5.2 Algorithm (Unknown Period Synchronous Beeping - UPSB)

Let p be a correct process. Let $T(p)$ be a totally arbitrary time period known by p.

Process p beeps if one of the following conditions is satisfied:

1. $now_1(t)$
2. $\neg wait_1(t) \wedge now_2(t)$
3. $\neg wait_1(t) \wedge \neg wait_2(t) \wedge (\forall t' \in]t - T(p), t[, m(t') = 0)$

5.3 Informal description

The UPSB algorithm performs in three steps:

- If no process beeps, then eventually, some correct process beeps (condition 3 of the algorithm).
- If two processes beep with a time interval T, then all correct processes beep T time units after the second beep (condition 2 of the algorithm). This ensures that at least $f + 1$ processes beep at the same time.
- If at least $f + 1$ processes beep at two different times, then all correct processes beep with the same time interval (condition 1 of the algorithm). Thus, we have synchronous beeping.

To avoid collisions between the conditions, we define the algorithm such that condition 1 has priority over condition 2, and condition 2 has priority over condition 3. This is ensured by the conditions $wait_i$ and now_i.

5.4 Correctness proof

Lemma 3. *Let $t \geq t_0$. There exists $t' \geq t$ such that $m(t) \neq 0$.*

Proof. Suppose the opposite: $\forall t' > t, m(t) = 0$.

Consider the point of view of a given process p.

- If there exists $t_1 < t$ and $t_2 < t_1$ such that $C_1(t, t_1, t_2)$ is true and $t - t_1 \leq t_1 - t_2$, then according to condition 1 of the algorithm, p beeps at time $t' = 2t_1 - t_2 \geq t$.
- Otherwise, if there exists $t_1 < t$ and $t_2 < t_1$ such that $C_2(t, t_1, t_2)$ is true and $t - t_1 \leq t_1 - t_2$, then according to condition 2 of the algorithm, p beeps at time $t' = 2t_1 - t_2 \geq t$.
- Otherwise, according to condition 3 of the algorithm, p beeps at time $t' \in [t, t + T(p)]$.

Therefore, in all cases, p beeps at a time $t' \geq t$: contradiction. Thus, the result.

Lemma 4. *Let $t \geq t_0$. There exists $t' \geq t$ such that $m(t) > f$.*

Proof. Suppose the opposite: $\forall t' \geq t, m(t) \leq f$.

Consider the point of view of a given correct process p. If there exists $t_1 < t$ and $t_2 < t_1$ such that $C_1(t, t_1, t_2)$ is true, then $\forall t' > 2t_1 - t_2$, $now_1(t')$ is false. Otherwise, $\forall t' \geq t$, $now_1(t')$ is false. Thus, there exists a date $t_3(p) \geq t$ such that, $\forall t' \geq t_3(p)$, $now_1(t')$ and $now_2(t')$ are false. Let t_3 be such that, for any correct process p, $t_3 \geq t_3(p)$.

According to Lemma 3, there exists $t_4 \geq t_3$ and $t_5 > t_4$ such that $m(t_4) \neq 0$ and $m(t_5) \neq 0$. Thus, at time $t' = 2t_5 - t_4$, $now_3(t')$ is true. As $t_5 > t_4 \geq t_0$, for all correct processes, condition 2 of the algorithm is satisfied at time t'. Thus, all correct processes beep at time t', and $m(t') > f$: contradiction. Thus, the result.

Theorem 3. *Algorithm UPSB ensures synchronous beeping.*

Proof. According to Lemma 4, there exists $t \geq t_0$ and $t_1 > t$ such that $m(t) > f$ and $m(t_1) > f$. Let t_2 be the earliest time such that $t_2 > t$ and $m(t_2) > f$. Let $T = t_2 - t$.

According to the algorithm, no correct process can beep a time $t' \in]t, t + T[$. Thus, as there is at most f Byzantine processes, $\forall t' \in]t, t + T[$, $m(t) \leq f$. Thus, as $t \geq 0$, for all correct processes, condition 1 of the algorithm is satisfied at time $t + T$. Thus, all correct process beep at time t, and $m(t) > f$.

Therefore, by induction, we achieve synchronous beeping at date t, with a period T.

6 Average Beeping Period

In the two previous sections, we gave an algorithm for the case where a same period T is initially known to all correct processes, and then one for the case where this period is unknown, and where the correct processes must agree on the same period. However, this can be any period.

We now consider the case where the correct processes have an approximate knowledge of a desired period T_0, and must agree on a period close to T_0. This is a more difficult problem, as correct processes are disorganized while Byzantine processes keep their perfect coordination capabilities.

For these reasons, we consider a more restrictive setting than the previous section. We now assume that $n > 3f$, and that no process beeps before a time t_0. This second assumption is justified in Section 6.1.

We assume that each correct process knows a time period $T(p)$ which is in a certain interval around T_0: $T(p) \in [T_0, (1+\epsilon)T_0]$, with $\epsilon \in]0, 1[$. The parameter ϵ can be as small as we want, and represents the precision of the knowledge of the period. We give an algorithm that ensures synchronous beeping with a period $T \in [(1-\epsilon)T_0, (1+\epsilon)T_0]$.

6.1 Lower bound

First, let us justify the removal of the self-stabilizing property for this part. We show that, if we require self-stabilization, no algorithm can ensure that the beeping period is in the desired interval.

Theorem 4. *There is no self-stabilizing algorithm ensuring synchronous beeping with a period $T \in [(1-\epsilon)T_0, (1+\epsilon)T_0]$.*

Proof. Suppose the opposite. Then, for a given initial state, there exists T' and t_1 such as described in Theorem 1.

Let t be a time where we have synchronous beeping with a period T. Let i be such that $iT \geq T'$. Then, all correct processes beep at time $t_2 = t + (i+1)T$. Let p be a correct process and let $T_1 = T(p)$.

Now, let q be a correct process, and suppose that the content of the interval $]t_2 - T', t_2[$ is the initial state for q. Then, if $T(q) = T_1$, q beeps at times t_0, $t_0 + T_1$, $t_0 + 2T_1 \ldots$

Let us show that there exists $T_2 > T_1$ such that, if $T(q) = T_2$, q does not beep at all times $t_0, t_0 + T_1, t_0 + 2T_1 \ldots$ Suppose the opposite. Then, the beeping period T_1 is independent of $T(q)$, and it is impossible to ensure that $T_1 \in [(1-\epsilon)T_0, (1+\epsilon)T_0]$: contradiction. Let T_2 be the smallest period having this property, and let $T_3 = T_2/(1+\epsilon)$.

Now, let us consider the two following situations:

1. For one correct process q, $T(q) = T_3$. For each other correct process p, $T(p) = T_2$. One Byzantine process b acts like a correct process with $T(b) = T_2$.
2. For each correct process p, $T(p) = T_2$. One Byzantine process b acts like a correct process with $T(b) = T_3$.

In situation 2, as the algorithm ensures synchronous beeping, all correct processes beep at date t_0, $t_0 + T_1, t_0 + 2T_1 \ldots$ Thus, as the two situations are indistinguishable for the correct processes, each correct process p such that $T(p) = T_2$ beeps at the same times. However, there exists a time t' such that all correct processes but q beep. As the behavior of q at a given time t only depends of the time interval $]t - T', t[$, the same situation repeats each T' time units, and q never beeps synchronously with other correct processes. Thus, we do not have synchronous beeping: contradiction.

6.2 Preliminaries

Our algorithm uses the following function g as well as several predicates.

Let T and T' be two time periods. Let $k \in \mathbb{Z}$ be the largest rational integer such that $T(1+\epsilon)^k \leq T'$. Then, let $g(T, T') = T(1+\epsilon)^k$.

Let t be any time, let $t_1 < t$, and let $t_2 < t_1$. We define the following predicates:

- $C_1(t, t_1, t_2) \equiv (m(t_1) > f) \wedge (m(t_2) > f) \wedge (\forall t' \in]t_2, t_1[\cup]t_1, t[, m(t') \leq f) \wedge (\exists t' < t_2, m(t')$
- $C_2(t, t_1, t_2) \equiv (m(t_1) > f) \wedge (m(t_2) \neq 0) \wedge (\forall t' \in]t_2, t_1[, m(t') = 0) \wedge (\forall t' \in]t_1, t[, m(t') \leq f)$
- $C_3(t, t_1, t_2) \equiv (m(t_1) \neq 0) \wedge (m(t_2) \neq 0)$

Then, for any time t and $\forall i \in \{1, 3\}$, we consider the following predicates:

- $now_i(t)$: there exists $t_1 < t$ and $t_2 < t_1$ such that $C_i(t, t_1, t_2)$ is true and $t - t_1 = t_1 - t_2$
- $wait_i(t)$: there exists $t_1 < t$ and $t_2 < t_1$ such that $C_i(t, t_1, t_2)$ is true and $t - t_1 < t_1 - t_2$

At last, we add the two following predicates:

- $now(t, T)$: there exists $t_1 < t$ and $t_2 < t_1$ such that $C_2(t, t_1, t_2)$ is true and $t - t_1 = g(t_1 - t_2, T)$.
- $wait(t, T)$: there exists $t_1 < t$ and $t_2 < t_1$ such that $C_2(t, t_1, t_2)$ is true and $t - t_1 < g(t_1 - t_2, T)$.

6.3 Algorithm (Average Period Synchronous Beeping - APSB)

A correct process p beeps if one of the following conditions is satisfied:

1. $now'_1(t)$
2. $\neg wait'_1(t) \wedge now(t, T(p))$
3. $\neg wait'_1(t) \wedge \neg wait(t, T(p)) \wedge now_3(t)$
4. $\neg wait'_1(t) \wedge \neg wait(t, T(p)) \wedge \neg wait_3(t) \wedge (\forall t' \in]t - T(p), t[, m(t') = 0)$

6.4 Informal description

The main difficulty is that the correct processes are disorganized. For instance, if each process p beeps $T(p)$ time units after a given beep, the correct processes may never beep at the same time.

To overcome this difficulty, we use a function $g(T, T')$ that uses any time measure T to split the periods $T(p)$ in two groups (see Lemma 5). The two possible output periods are in the interval $[(1-\epsilon)T_0, (1+\epsilon)T_0]$ (see Lemma 6). Thus, as $n > 3f$, a majority of correct processes beep with the same period, which is in the desired interval.

The formalism is similar to the previous algorithm. The principle is as follows:

- Condition 4 of the algorithm ensures that a process always eventually beeps.
- Condition 3 ensures that all correct processes eventually beep at the same time.
- Condition 2 computes the aforementioned principle.
- Condition 1 reproduces the same time period and ensures synchronous beeping.

6.5 Correctness proof

Lemma 5. *For a given $T > 0$, let $G(T) = \bigcup_{T' \in [T_0, (1+\epsilon)T_0]} g(T, T')$. Then, $|G(T)| \leq 2$.*

Proof. Let $k \in \mathbb{Z}$ be the largest relative integer such that $T(1+\epsilon)^k \leq T_0$. Thus, $g(T, T_0) = T(1+\epsilon)^k$ and $T_0 < T(1+\epsilon)^{k+1}$. Therefore, $(1+\epsilon)T_0 < T(1+\epsilon)^{k+2}$ and $g(T, (1+\epsilon)T_0) \leq T(1+\epsilon)^{k+1}$. Then, either $G(T) = \{T(1+\epsilon)^k\}$ or $G(T) = \{T(1+\epsilon)^k, T(1+\epsilon)^{k+1})\}$, and $|G(T)| \leq 2$.

Lemma 6. *$\forall T > 0$ and $\forall T' \in [T_0, (1+\epsilon)T_0]$, $g(T, T') \in [(1-\epsilon)T_0, (1+\epsilon)T_0]$.*

Proof. Let k be largest relative integer such that $T(1+\epsilon)^k \leq T_0$. Then, $T_0 < T(1+\epsilon)^{k+1} = g(T, T_0)(1+\epsilon)$. As $(1-\epsilon)(1+\epsilon) = 1-\epsilon^2 < 1$, $1-\epsilon < 1/(1+\epsilon)$, and $g(T, T_0) > T_0/(1+\epsilon) > (1-\epsilon)T_0$. Thus, $g(T, T') \geq (1-\epsilon)T_0$. Besides, as $g(T, T') \leq T'$, $g(T, T') \leq (1+\epsilon)T_0$. Thus, the result.

Lemma 7. *There exists $t \geq t_0$ such that $m(t) \neq 0$.*

Proof. Suppose the opposite: $\forall t \geq t_0$, $m(t) = 0$. Let p be a correct process. Then, according to condition 4 of the algorithm, p beeps at time $t_0 + T(p)$: contradiction. Thus, the result.

Lemma 8. *There exists $t \geq t_0$ and $t' < t$ such that $m(t) > f$ and $m(t') \neq 0$.*

Proof. Suppose the opposite: there exists no such t and t'. According to Lemma 7, there exists a time t_1 such that $m(t_1) \neq 0$. Let t_1 be the earliest date such that $m(t_1) \neq 0$. According to Lemma 7, there exists $t' \geq t_0$ such that $t' > t_1$ and $m(t') \neq 0$. Then, according to condition 3 of the algorithm, all correct processes beep at time $2t' - t_1$, and $m(2t' - t_1) > f$: contradiction. Thus, the result.

Theorem 5. *Algorithm APSB ensures synchronous beeping with a period* $T \in [(1-\epsilon)T_0, (1+\epsilon)T_0]$.

Proof. Let t be the earliest time such as described in Lemma 8. Let t' be the latest time such that $t' < t$ and $m(t') \neq 0$. According to condition 2 of the algorithm, each correct process p beeps at time $t + g(t - t', T(p))$.

According to Lemma 5 and Lemma 6, there are only two possible value T_1 and T_2 of $g(t-t', T(p))$, and $\{T_1, T_2\} \in [(1-\epsilon)T_0, (1+\epsilon)T_0]$. Let P_1 (resp. P_2) be the set of correct processes beeping at time $t+T_1(p)$ (resp. $t+T_2(p)$). As $n > 3f$, then either $|P_1| > f$ or $|P_2| > f$. Thus, there exists $t_1 \in \{t + T_1(p), t + T_2(p)\}$ such that $m(t_1) > f$. Let t_1 be the earliest time such that $t_1 > t$ and $m(t_1) > f$.

Let us show that $T = t_1 - t \in [(1-\epsilon)T_0, (1+\epsilon)T_0]$. Suppose the opposite. Then, according to condition 2 of the algorithm, no correct process beeps at time t_1. Thus, at least $f + 1$ Byzantine nodes beep at time t_1: contradiction.

Then, according to condition 1 of the algorithm, all correct processes beep at time $t_1 + T$, and no correct process beeps in the time interval $]t_1, t_1 + T[$. Therefore, by induction, we have synchronous beeping at time t_1 with a period $T \in [(1-\epsilon)T_0, (1+\epsilon)T_0]$.

7 Synchronous Lighting

In the previous sections, we assumed that the processes could produce discrete beeps. In this section, we assume that a process p can continuously increase and decrease a luminosity variable $l(p)$. We define an alternative but similar problem (*synchronous lighting*) and give an algorithm for the case where the desired period is approximately known. In this section, $n > 2f$.

Each correct process p knows a time period $T(p) \in [T_0, (1+\epsilon)T_0]$, and holds a variable $l(p) \in [0, 1]$. $l(p)$ is a continuous function of time. We assume that the time to increase (resp. decrease) $l(p)$ from 0 to 1 (resp. 1 to 0) is at most ϵT_0. Let P be the set of processes. Let $L(t)$ be the value of $\Sigma_{p \in P} l(p)$ at time t.

7.1 Problem

We say that the processes achieve *synchronous lighting* at time t_1 if there exists t_2, t_3, $t_4 \ldots$ such that:

1. Each time t_i corresponds to a peak of luminosity: $\forall i \in \{1, 2, 3, \ldots\}$, $L(t_i) \geq n - f$.
2. The delay between two consecutive times t_i is approximately equal to T_0: $\forall i \in \{1, 2, 3, \ldots\}$, $t_{i+1} - t_i \in [T_0, (1 + 2\epsilon)T_0]$.
3. The correct processes only produce light around times t_i: for a given correct process p, if $l(p) \neq 0$ at time t, then there exists $i \in \{1, 2, 3, \ldots\}$ such that $|t - t_i| \leq 2\epsilon T_0$.

7.2 Algorithm (Average Period Synchronous Lighting - APSL)

Each correct process p has the following behavior:

- If there exists $t' \in]t - T(p), t[$ such that $L(t') \geq n - f$, decrease $l(p)$.
- Otherwise, increase $l(p)$.

7.3 Correctness Proof

Lemma 9. *There exists $t \geq t_0$ such that $L(t) \geq n - f$.*

Proof. Suppose the opposite: $\forall t \geq t_0$, $L(t) < n - f$. Let p be a correct process. Then, according to the algorithm, starting from time $t_0 + T(p)$, $l(p)$ increases. Therefore, at time $t_0 + (1 + 2\epsilon)T_0$, for each correct process p, $l(p) = 1$. Thus, $L(t_0 + (1 + 2\epsilon)T_0) \geq n - f$: contradiction. Thus, the result.

Theorem 6. *Algorithm APSL ensures synchronous lighting.*

Proof. According to Lemma 9, there exists $t \geq t_0$ such that $L(t) \geq n - f$. Therefore, according to the algorithm, starting from time t, each correct process p decreases $l(p)$. Therefore, at time $t + \epsilon T_0$, for each correct process p, $l(p) = 0$. Then, as there are at most f Byzantine processes, $\forall t' \in]t + \epsilon T_0, t + T_0[$, $L(t') \leq f \leq n - f$.

Now, let us show that there exists $t_1 \in [t + T_0, t + (1 + 2\epsilon)T_0]$ such that $L(t_1) \geq n - f$. Suppose the opposite. Let p be a correct process. Then, according to the algorithm, starting from time $t + T(p)$, $l(p)$ increases. Therefore, at time $t + (1 + 2\epsilon)T_0$, for each correct process p, $l(p) = 1$. Thus, $L(t + (1 + 2\epsilon)T_0) \geq n - f$: contradiction.

Then, $L(t_1) \geq n - f$, $t_1 - t \in [T_0, (1 + 2\epsilon)T_0]$ and for each correct process p, if $t' \in]t + \epsilon T_0, t + T_0[$, $l(p) = 0$ at time t'. Thus, if $l(p) \neq 0$ at time $t' \in [t, t_1]$, then either $|t' - t| \leq 2\epsilon T_0$ or $|t' - t_1| \leq 2\epsilon T_0$.

Therefore, by induction, we have synchronous lighting at time t.

8 Conclusion

We considered the problem of synchronous beeping. We assumed the presence of Byzantine processes that can beep as often as they want. We gave synchronization algorithms for the cases where the period is known, unknown and approximately known. We also considered an alternative continuous model.

An open question is the tightness of the condition $n > 3f$ for the average beeping knowledge. Also, many extensions could be made on the communication graph and the communication delays.

Acknowledgement

This work has been supported in part by the European ERC Grant 339539 - AOC.

References

1. Facts about fireflies (http://www.firefly.org/facts-about-fireflies.html).
2. Dan Alistarh, Alejandro Cornejo, Mohsen Ghaffari, and Nancy Lynch. Firefly synchronization with asynchronous wake-up. *Workshop on Biological Distributed Algorithms (BDA 2014)*.
3. Alejandro Cornejo, Anna Dornhaus, Nancy Lynch, and Radhika Nagpal. Task allocation in ant colonies. *Proceedings of the 28th International Symposium on Distributed Computing (DISC 2014)*.
4. Flaviu Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3:146–158, 1989.
5. A. Daliot, D. Dolev, and H. Parnas. Self-stabilizing pulse synchronization inspired by biological pacemaker networks. *Symposium on Stabilization, Safety and Security of Distributed Systems (SSS 2003)*.
6. Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.
7. Amir Hossein Gandomi, Xin-She Yang, and Amir Hossein Alavi. Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Engineering with Computers*, 29:17–35, 2013.
8. Hermann Kopetz. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers*, C-36:933–940, 1987.
9. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21:558–565, 1978.
10. Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
11. Dennis Lucarelli and I-Jeng Wang. Decentralized synchronization protocols with nearest neighbor communication. *SenSys*, pages 62–68, 2004.
12. Renato E. Mirollo, Steven, and H. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM J. Appl. Math*, 50:1645–1662, 1990.
13. R.Y.M. Nakamura, L.A.M. Pereira, K.A. Costa, D. Rodrigues, J.P. Papa, and X.-S. Yang. BBA: A binary bat algorithm for feature selection. *25th Conference on Graphics, Patterns and Images (SIBGRAPI 2012)*.
14. Charles S. Peskin. Mathematical aspects of heart physiology. 1973.
15. Chris Reid, Hannelore MacDonald, Tanya Latty, Richard Mann, and Simon Garnier. Cellular decision-making: How an amoeboid organism solves the two-armed bandit problem. *Workshop on Biological Distributed Algorithms (BDA 2014)*.
16. Kay Römer. Time synchronization in ad hoc networks. *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking and computing*, pages 173–182, 2001.
17. Osvaldo Simeone, Umberto Spagnolini, Yeheskel Bar-Ness, and S. Strogatz. Distributed synchronization in wireless networks. *IEEE Signal Processing Magazine*, 25(5):81–97, 2008.
18. Hugh M. Smith. Synchronous flashing of fireflies. *Science*, 82(2120):151–152, 1935.
19. Steven H. Strogatz. Sync: The emerging science of spontaneous order. *Hyperion, 1st edition*.
20. Danny Weyns and Tom Holvoet. Regional synchronization for simultaneous actions in situated multi-agent systems. *Multi-Agent Systems and Applications III, Lecture Notes in Computer Science*, 2691:497–510, 2003.
21. Xin-She Yang, Zhihua Cui, Renbin Xiao, Amir Hossein Gandomi, and Mehmet Karamanoglu. *Swarm Intelligence and Bio-Inspired Computation, Theory and Applications*. Elsevier Insights, 2013.