Self-Stabilizing Clock Synchronization with 3-bit messages

Lucas Boczkowski ∗ Amos Korman † Emanuele Natale ‡

Abstract

This paper studies the complexities of basic distributed computing tasks while operating under severe fault-tolerant contexts and strong communication constraints. We consider the self-stabilizing context, in which internal states of processors (agents) are initially chosen by an adversary. Furthermore, we assume that agents are passively mobile in the sense that they have no control over who they interact with. More specifically, we consider a population of $n$ agents. Communication is restricted to the synchronous PULL model, where in each round, each agent can pull information from few other agents, chosen uniformly at random. We are primarily interested in the design of algorithms that use messages as small as possible. Indeed, in such models, restricting the message-size may have a profound impact on both the solvability of the problem and its running time. We concentrate on variants of two fundamental problems, namely, clock-synchronization, where agents aim to synchronize their clocks modulo some $T$, and Zealot-consensus, in which agents need to agree on a common output bit, despite the presence of at most one agent whose output bit is fixed throughout the execution.

Our main technical contribution is the construction of a clock-synchronization protocol that converges in $\tilde{O}(\log^2 T \log n)$ rounds with high probability and uses only 3 bits per message. In addition to being self-stabilizing, this protocol is robust to the presence of $O(n^{1/2-\varepsilon})$ Byzantine agents, for any positive constant $\varepsilon$. Using this clock-synchronization protocol, we solve the self-stabilizing Zealot-consensus problem in time $\tilde{O}(\log n)$ using only 4-bit messages.

Our technique for obtaining the clock synchronization protocol is based on a compact simulation of the stabilizing consensus protocol in the PULL model proposed by Doerr et al. (SPAA ’11). In fact, our method is rather general and can be applied to a large family of models where agents are passively mobile, provided that there exists a self-stabilizing symmetric-consensus protocol defined for a single bit. In this case, our transformer would produce a self-stabilizing clock-synchronization algorithm without increasing the asymptotic message size of the consensus algorithm, and paying only a small penalty in the running time.

∗IRIF, CNRS and University Paris Diderot, Paris, 75013, France. E-mail: Lucas.Boczkowski@irif.fr.
†IRIF, CNRS and University Paris Diderot, Paris, 75013, France. E-mail: Amos.Korman@irif.fr.
‡Department of computer science, Sapienza University of Rome. E-mail: natale@di.uniroma1.it.

This work has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 648032).
1 Introduction

1.1 Background

A recent trend in distributed computing is to investigate the construction of protocols that must operate under strong communication, computation, and fault-tolerant constraints \[5, 28, 29\]. Besides the purely theoretical interest and relevance to ad-hoc networks, this line of research is also motivated by possible connections to biological contexts \[1, 16, 34, 39\] and to models of interacting particles \[46\]. Following this framework, this paper studies the complexities of fundamental distributed computing problems while operating under both severe fault-tolerant contexts and strong communication limitations.

Concerning faults, although we may assume the presence of several Byzantine agents \[42\], we are primarily concerned with transient faults such as those occurring in self-stabilization contexts \[19\]. In such contexts, the system must converge to a correct configuration from any initial configuration, or in other words, the protocol must cope with the situation in which agents start their execution with arbitrary internal states. In particular, ID’s of agents can be initially corrupted, making the network essentially anonymous.

In terms of communication, agents are assumed to be passively mobile in the sense that they cannot control who they interact with \[5\]. The canonical example is a flock of birds with a sensor attached to each bird, but one can imagine many other scenarios, both in the natural world, and in engineering contexts, that may also apply to such a context. Passively mobile agents must organize their operations to fit meeting patterns which may only be partially known to them. Similarly to many works on passively mobile systems (see, e.g., \[3, 8, 10\]), we model such communication patterns by assuming that agents interact uniformly at random. Specifically, we consider the synchronous PULL model of communication, in which in each round, each agent can “observe” few other agents, chosen uniformly at random. One interesting aspect of such random communication models is their inherent sensitivity to congestion issues. Indeed, in such models it is not clear how to simulate a protocol that uses large messages while using only small size messages. For example, the straightforward strategy of breaking a large message into small pieces and sequentially sending them one after another does not work, since one typically cannot make sure that the small messages reach the same destination. Hence, reducing the message size may have a profound impact on the running time, and perhaps even on the solvability of the problem at hand.

To the best of our knowledge, there are only very few known protocols that operate under stochastic interaction patterns using small size messages and are both time efficient and self-stabilizing. A rare example is the work of Doerr et al. on stabilizing consensus \[22\] in the synchronous PULL model of communication. Essentially, aiming to agree on an output bit, each agent samples two other arbitrary agents in each round, and updates its bit according to the majority bit among the output bits of the two sampled agents and its own output bit. This simple protocol uses a single bit per interaction, namely, the output bit, and converges in \(O(\log n)\) time with high probability. Moreover, in addition to being self-stabilizing it can tolerate the interference of up to \(O(\sqrt{n})\) Byzantine agents.

In this paper, we are interested in the design of self-stabilizing algorithms for several basic problems in the same model of computation as assumed in \[22\]. Specifically, we consider the following problems.

- **Clock synchronization** \((T)\): Agents have clock counters that synchronously tick at each round, but may not agree on their values. The goal is to synchronize the clock counters modulo some prescribed integer \(T\).

- **Zealot-consensus**: Each agent \(u\) has an output bit. At most one agent, called Zealot\(^1\), has its output bit \(b\) frozen, i.e., it remains fixed throughout the execution. Its internal state is controlled by an adversary throughout the execution of the protocol. All other agents aim to reach a consensus on all output bits, including the output bit of the Zealot agent. In other words, in a legal configuration, the output bit of every agent should equal to \(b\).

\(^1\) Zealot is a term commonly used in the physics literature on the Voter model to denote an agent whose opinion is fixed (see e.g., \[48, 49\]).
In both problems, output bits, as well as inner states, including clock counters, are all assumed to be set initially by an adversary. For the Zealot-consensus problem, we assume that each agent knows the precise number of agents \( n \), and that this information is not corrupted.

Solving any of these problems using logarithmic size messages is not difficult. Our main challenge is to establish self-stabilizing protocols that use only \( O(1) \) number of bits per interaction.

### 1.2 On the difficulties of the problems

To get some intuition, consider first the Zealot-consensus problem. Here, difficulties may arise from both the self-stabilizing constraint, as well as from the fact that the output bit of the Zealot is fixed and its internal state is controlled by an adversary. To highlight the difficulties caused by self-stabilization, let us assume, for the purpose of this subsection, that the Zealot agent is in fact “cooperative”. That is, let us assume that although its output bit is frozen, the Zealot can run a separate protocol and use its internal state to actually help other agents converge on its output value. This weaker version of the problem can be thought of as a broadcast problem. To avoid confusion, in the context of this broadcast variant, we use the terminology source to denote the Zealot agent.

In the fault-free (non-self-stabilizing) context, this broadcast problem can trivially be solved in \( O(\log n) \) rounds as follows. In addition to its option bit \( b_u \), each agent \( u \) stores a certainty bit \( c_u \). Informally, having a certainty bit equal to 1 indicates that the agent is certain of the correctness of its output bit. Initially, only the source sets its certainty bit to 1, and all other agents set their corresponding bit to 0. Whenever an agent \( v \) observes \( u \) and sees the tuple \((b_u, c_u)\), where \( c_u = 1 \), it copies the output bit of \( u \) (i.e., sets \( b_v = b_u \)) and sets its certainty bit to 1. This way, the opinion bit of the source propagates to all agents within \( O(\log n) \) rounds, w.h.p., see e.g., [37]. Note that this strategy fails in the self-stabilizing context since agents may initially be fooled by an adversary to “think” that they are in the midst of the execution and already “informed” with the wrong bit. These agents will then propagate wrong information which may prevent the correct convergence of the system.

A first idea to handle the aforementioned difficulty is to let each agent store a timestamp \( t_u \), that is an integer between 1 and \( O(\log n) \) (the running time of the aforementioned propagation protocol). Informally, this timestamp indicates how “fresh” is the opinion at \( u \). The source agent always sets its timestamp to 1. In each round, each (non-source) agent increases its stamp by 1. Whenever an agent \( u \) sees another agent \( v \) with a smaller timestamp it copies both \( v \)’s opinion and it’s timestamp. For any initial configuration of (possibly corrupted) opinions and timestamps, this process guarantees that w.h.p., within \( O(\log n) \) rounds, the only opinion that will survive is that of the source. The downside of this simple protocol is that in order to encode a timestamp, \( \Omega(\log \log n) \) bit messages must be used, while we are restricted to use constant size messages only.

A second idea is to use certainty bits as before, but let all non-source agents count rounds internally, and reset their certainty bit to 0 every \( O(\log n) \) rounds. Note however, that for such an approach to work, agents must have a consistent notion of time (at least modulo \( O(\log n) \)) so that they can reset their certainty bit to zero at the same time. Indeed, if this reset is not done simultaneously, then it is not clear how to prevent agents that just reset their certainty bit to 0 from being “infected” by “misleading” agents, namely, those that have the wrong opinion and certainty bit equal to 1. In other words, it is not clear how to get rid of misleading agents.

To synchronize clocks modulo \( T \), one could use the protocol in [22], by displaying the bits of the clocks in each message, and synchronizing on each of them separately and in parallel while incrementing the clocks (see Section 3.1 for more details). Unfortunately, this approach is wasteful in terms of message size, as it requires to reveal \( \log T \) bits per interaction. As another approach, one could aim at sequentially synchronizing clocks bit after bit. That is, first display and synchronize the first bit; then, once agents “know” that the first bit has been synchronized, display and synchronize the second bit, etc. This approach is problematic in the context of self-stabilization, since, first, it requires agents to “know” when a bit is synchronized, and second, it requires agents to agree on the bit index that they currently aim to synchronize. Both of these require clocks to be synchronized to begin with.

Finally, we investigated some classes of protocols that, according to simulations, seem to effectively solve
the Zealot-consensus problem using just one bit per message. However, these candidate protocols rely on certain mechanisms whose analysis seems to be far from the reach of the current techniques for analyzing similar stochastic processes. We briefly present and discuss such protocols in Appendix A.5.

1.3 Our Results

We focus on the randomized PULL model of communication as assumed in [22]. Our main technical contribution is the construction of a time efficient self-stabilizing clock synchronization protocol that employs only 3 bits of communication in each message. Relying on this clock synchronization algorithm, and using an additional bit in each message, we then solve the self-stabilizing Zealot-consensus problem. Formally, we prove the following.

**Theorem 1.1.** Consider the synchronous randomized PULL model in which in each round, each agent contacts two agents chosen uniformly at random among all agents (including itself, with repetition). We assume that agents have clock counters that tick synchronously at each round but may not agree on their actual value. Let $T$ be a power of 2.

- There exists a self-stabilizing clock synchronization algorithm that synchronizes clocks modulo $T$ and uses only 3 bits per interaction. The protocol converges in $\tilde{O}(\log^2 T \log n)$ rounds with high probability.\(^2\)

- If agents know $n$ (and this information is not corrupted by an adversary) then the running time can be improved, while keeping the message size constant. Specifically, there exists a self-stabilizing clock synchronization algorithm that synchronizes clocks modulo $T$ and uses only 4 bits per interaction. The protocol converges in $\tilde{O}(\log T \log n)$ rounds with high probability.

When $T \leq \text{poly}(n)$, both protocols can tolerate the presence of up to $\mathcal{O}(n^{1/2-\epsilon})$ Byzantine agents. Without Byzantine interferences, the corresponding legal synchronized configuration will remain forever and with such interference, legal configurations will remain for polynomially many rounds, with high probability.

**Theorem 1.2.** Consider the synchronous randomized PULL model, and assume that agents know $n$. There exists a self-stabilizing Zealot-consensus algorithm that uses 4 bits per interaction. The protocol converges in $\tilde{O}(\log n)$ rounds with high probability, and afterwards remains in a legal configuration indefinitely.

Our technique for obtaining the clock synchronization protocol is based on a compact simulation of the stabilizing consensus protocol in the PULL model proposed by Doerr et al. in [22]. In fact, our method is rather general and could be applied to a large family of models and corresponding consensus protocols (see more details in Section 5). In these cases, our transformer uses the consensus algorithm as a “black-box” and produces a self-stabilizing clock synchronization algorithm without increasing the asymptotic message size of the consensus algorithm, and paying only a small penalty in the running time. Since this transformer is not the main focus of this paper, we only sketch the main ideas behind it.

1.4 Related work

Clock synchronization is one of the most fundamental building blocks of distributed systems [11, 41, 44, 45, 47]. Indeed, the vast majority of distributed tasks require some sort of synchronization, making the design of robust clock synchronization protocols highly applicable. We consider a synchronous system in which clocks tick at the same pace but may not share the same value. This version of clock synchronization has earlier been studied in [9, 15, 23, 24, 26, 35, 38, 40, 47] under different names, including “digital clock synchronization” and “synchronization of phase-clocks”. Here, we will simply use the term “clock synchronization”.

\(^2\)The notation $\tilde{O}(f(n))$ refers to $f(n) \cdot \log^{O(1)}(f(n))$. We also write $\tilde{O}(f(n)g(T))$ to denote $f(n)g(T) \log^{O(1)}(f(n)) \log^{O(1)}(g(T))$. As accustomed, the term high probability refers to a probability of at least $1 - n^{-\Omega(1)}$. 

3
Consensus is another fundamental problem of distributed computing and multi-agent systems \cite{30,33,43}. In this problem agents need to agree on a value proposed by one of them, despite the presence of different types of faults, including Byzantine or transient faults. Recently, there has been considerable interest in consensus algorithms that operate in models that severely restrict both communication and computation \cite{3,8,22,32}. Due to the nature of Byzantine processors (which in particular, can change their opinions during the execution), the typical requirement is that consensus is achieved by non-Byzantine agents only. In contrast, the Zealot-consensus problem studied in this paper requires all agents to agree on their output bit, including the Zealot agent. This holds even if the state of the Zealot is controlled by an adversary (as long as its output bit is fixed). A weaker version of this problem, as mentioned in Section 1.2, considers the Zealot as being more cooperative, in the sense that although its opinion bit is fixed, its internal state can be used by the Zealot to help other agents reach consensus. This weaker version of the problem bares similarities with the task of broadcast \cite{17,18,20,21,29,31,37}.

In recent years, there has been a lot of interest in the community of distributed computing in models of computation that restrict both memory and communication. One important example concerns Population protocols \cite{5,10}, which model systems composed of a large number of individuals equipped with constant memory size that interact with each other, but have little or no control over who they interact with. The typical communication models are either adversarial or uniformly random (similar to the model we assume). Works in this area focused on identifying the set of solvable tasks \cite{5,6}, and on understanding the convergence bounds achievable for basic tasks, most notably majority-consensus \cite{8} and leader-election \cite{4,25}. Another line of research concerns the Beeping model \cite{1,2} and the Stone age model \cite{28}, which focus on the ability to solve distributed tasks fast while using very limited communication. The main problem investigated in the Beeping model is the Maximal Independent Set problem. Unfortunately, despite considerable amount of interesting results obtained in all the aforementioned models, the understanding of their fault-tolerant aspects is still rather unsatisfactory \cite{7,13,14}.

Another (not necessarily disjoint) large body of work aims at analyzing the dynamics of simple protocols involving randomly interacting agents. Among the most well-studied interaction patterns are the synchronous PUSH and PULL models of communication on complete graphs \cite{37}. Under these models of interaction, a lot of attention has been devoted to studying the round and message complexities of Broadcast \cite{21,37} and Consensus \cite{12,22}. The robustness of PUSH / PULL based protocols to weak types of faults, such as crashes of messages and/or agents, or to the presence of relatively few (typically, polynomially small fraction of) Byzantine agents, has been known for quite a while \cite{27,37}. Recently, it has been shown that under such interaction patterns, there exists an efficient Broadcast protocol that uses a single bit per message and can overcome flips in messages (noise) \cite{29}. The protocol therein, however, heavily relies on the assumption that agents know when the protocol has started. Observe that in a self-stabilizing context such an assumption would be difficult to remove.

2 Preliminaries

2.1 Model

The PULL model of communication. We consider the same version of the synchronous randomized PULL model as assumed by Doerr et al. \cite{22}. Specifically, each agent designates one part of its memory, called the visible part (see below). Communication proceeds in discrete rounds. In each round, each agent $u$ “observes” two arbitrary other agents, chosen uniformly at random among all agents, and can peek into the visible part of their memory. If several agents observe $v$ at the same round then they all see the same visible part of $v$’s memory. This part of the memory is also called “message”. In contrast to most previous works on self-stabilization, we primarily aim at designing protocols that use messages as small as possible, while keeping the running time poly-logarithmic. Indeed, all our protocols operate in the PULL model.

Message size. As mentioned, the visible part of the memory is used by agents to encode the information they wish to reveal to others. In the case of Zealot-consensus, we assume that the opinion bit always appears in the visible parts of agents, at a designated location (e.g., at the least significant bit). The message size denotes the
maximal number of bits stored in the visible part of an agent.

Convergence. In any of the aforementioned problems (see Section 1.1), we do not require that each agent irrevocably commits to a final value but that eventually agents arrive at a legal configuration without necessarily being aware of that. Specifically, in the context of clock synchronization \((T)\), a legal configuration is reached when all clocks (of non-Byzantine agents) show the same time modulo \(T\), and in the Zealot-consensus and broadcast problems, a legal configuration is reached when all agents have the same output bit. The system is said to reach a **stable legal configuration** if it reaches a legal configuration that remains legal for at least some polynomial time. In fact, in the Zealot-consensus problem, we guarantee that once a legal configuration is reached, it remains legal indefinitely.

### 2.2 A majority based, self-stabilizing protocol for consensus on one bit

Recall the **CONSENSUS** protocol in [22]. Each agent holds an opinion bit. In each round each agent looks at the opinion bits of two other random agents and updates its opinion bit taking the majority among the bits of the observed agents and its own. Note that the protocol uses only a single bit per interaction, namely, the output bit. The usefulness of **CONSENSUS** comes from its extremely fast and fault-tolerant convergence toward an agreement among agents, as given by the following result.

**Theorem 2.1** (Doerr et al. [22]). From any initial configuration, **CONSENSUS** converges to a state in which all agents agree on the same output bit in \(O(\log n)\) rounds, w.h.p. Moreover, if there are at most \(\kappa \leq n^{1/2-\varepsilon}\) Byzantine agents, then after \(O(\log n)\) rounds all non-Byzantine agents have converged and consensus is maintained for \(n^{\Omega(1)}\) rounds w.h.p.\(^3\)

### 2.3 Notations

**Definition 2.2.** For a given integer \(x \in \mathbb{N}\), we denote by \(\text{BINARY}(x)\) the binary expansion of \(x\). On some occasions, if it is clear from context whether we are referring to the binary expansion or to the value of the number, we might omit the notation \(\text{BINARY}\). The reverse operation to \(\text{BINARY}\) will be denoted \(\text{INTEGER}\).

**Definition 2.3** (clocks). Let \(m \in \mathbb{N}\). We will often refer to a counter \(C\) modulo \(2^m\) as a clock. We may view it as written in binary \(C := \text{BINARY}(C) = (b_1, \ldots, b_m)\) and we will use the notation \(C(j)\) to denote \(b_j = \text{BINARY}(C)(j)\). We refer to \(m\) as the size of \(C\).

For an odd number \(n \in 2\mathbb{N} + 1\), and a string \(x \in \{0, 1\}^n\), let \(\text{MAJ}(x)\) denote the most frequent bit in \(x\). Given a multiset of \(k\)-strings of length \(n\), we denote by \(\text{BITWISE-MAJ}\) the \(k\)-string that gives the bitwise majority.

### 3 Self-stabilizing clock synchronization

We will describe and analyze two protocols for clock synchronization. First a simple one, denoted **SYNC-SIMPLE** that uses \(O(\log T)\) bits per interaction. Then we will present our main protocol denoted **SYNC-3BIT**, that uses only 3 bits of communication.

---

\(^3\) The original statement of [22] says that if at most \(\kappa \leq n\) agents can be corrupted at any round, then convergence happens for all but at most \(O(\kappa)\) agents. Let us explain how this implies the statement we gave, namely that we can replace \(O(\kappa)\) by \(\kappa\), if \(\kappa \leq n^{1/2-\varepsilon}\). Assume that we are in the regime \(\kappa \leq n^{1/2-\varepsilon}\). It follows from [22] that all but a set of \(O(\kappa)\) agents reach consensus after \(O(\log n)\) round. This set of size \(O(\kappa)\) contains both Byzantine and non Byzantine agents. However, if the number of agents holding the minority opinion is \(O(\kappa) = O(n^{1/2-\varepsilon})\), then the expected number of non Byzantine agents that disagree with the majority at the next round is in expectation \(O(\kappa^2/n) = O(n^{-\varepsilon})\) agents w.h.p. Thus, by Markov inequality, this implies, that at the next round consensus is reached among all non-Byzantine agents w.h.p. Note also that, for the same reasons, the Byzantine agents do not affect any other non-Byzantine agent for \(n^\varepsilon\) rounds w.h.p.
3.1 Description of Protocol \textsc{sync-simple}

Each agent maintains a clock $C_u \in [1, T]$. At each round, each agent $u$ displays the value of her clock $C_u$, pulls 2 uniform other such clock values, and updates her clock as the bitwise majority of the two clocks it pulled, and her own. Then the clock $C_u$ is incremented. Below we present the pseudo code for one round of algorithm \textsc{sync-simple}. It is to be executed by every agent $u$ synchronously:

<table>
<thead>
<tr>
<th>Algorithm 1: One round of \textsc{sync-simple}, executed by each agent $u$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $u_1, u_2 \leftarrow \text{SAMPLE}(2)$ # sample two agents uniformly from the population</td>
</tr>
<tr>
<td>2. $C_u \leftarrow \text{INTEGER} \left( \text{BITWISE_MAJ(BINARY}(C_u), \text{BINARY}(C_{u_1}), \text{BINARY}(C_{u_2})) \right)$</td>
</tr>
<tr>
<td>3. $C_u \leftarrow C_u + 1$</td>
</tr>
</tbody>
</table>

**Theorem 3.1.** Let $T$ be a power of 2. The protocol \textsc{sync-simple} is a self-stabilizing protocol that synchronizes clocks modulo $T$ in $O(\log T \log n)$ rounds w.h.p.

*Proof.* Let us look at the least significant bit. One round of \textsc{sync-simple} is equivalent to one round of \textsc{consensus} with an extra flipping of the opinion bit due to the increment of the clock. The crucial point is that all agents jointly flip their bit on every round. Because the function agents apply, MAJ, is symmetric, it commutes with the flipping operation. More formally, let $\vec{b}_i$ be the vector of all the first bits, of all clocks at round $i$ under an execution of \textsc{sync-simple}. We also denote by $\vec{c}_i$ all the first bits, of all messages sent at round $i$ under an execution of \textsc{sync-simple}, removing the increment of time (line 3). We couple $\vec{b}$ and $\vec{c}$ by running the two versions on the same interaction pattern. Then, $\vec{b}_i$ is equal to $\vec{c}_i$ on even rounds, and is equal to $\vec{c}_i$ flipped, on odd rounds. Moreover, we know from Theorem 2.1 that $\vec{c}_i$ converge to a stable value in a self-stabilizing manner. It follows that, from any initial configuration of states (i.e. clocks), w.h.p, after $O(\log n)$ rounds of executing \textsc{sync-simple}, all agents share the same value for their first bit, and jointly flip it in each round. Once agents agree on the first bit, because $T$ is a power of 2, the increment of time makes them flip the second bit jointly once every 2 rounds.

More generally, assuming agents agree on the first $\ell$ bits of their clocks, they jointly flip the $\ell + 1$st bit once every $2^\ell$ rounds, on top of doing the \textsc{consensus} protocol on that bit. The same reasoning shows that the flipping doesn’t affect the convergence on bit $\ell + 1$. Thus w.h.p., $O(\log n)$ rounds after the first $\ell$ bits are synchronized, the $\ell + 1$st bit is synchronized as well. \hfill $\square$

3.2 The Protocol \textsc{sync-3bit}

Here we describe the 3 bit protocol \textsc{sync-3bit}. See also the pseudo code below.

We want to apply a similar strategy to \textsc{sync-simple}, while using only $O(1)$ many bits per interaction. At a high level, agents keep a clock internally which they write using $\log T$ bits. This time, instead of displaying the full clock, we would like them to display the first bit of their clock all at the same time, then the second bit etc... They could do that, if they had a clock, which is precisely what we are trying to achieve.

We use a recursive approach, by which agents maintain several clocks, of larger and larger sizes, and use the already synchronized clocks to decide which bit of the next (bigger) clock to display (and synchronize on). More precisely, the recursion is initialized by synchronizing a small clock $C^{(1)}$ of 2 bits (thus running modulo 4). This is done in the same way as \textsc{sync-simple}. The small clock in fact corresponds to the first two bits of the clock $C$ to be synchronized. The third bit displayed gives the value at some index of the (binary expansion) of $C$. This third bit traverses all indices. Initially, the traversals are not synchronized, and agents are not displaying the same index in each round necessarily. The traversals are designed in a self improving way, making them more and more synchronized, as the protocol progresses.
Let us now give more details. Agents want to synchronize a big clock \( C \). They divide the bits used to write \( C \) into blocks of increasing sizes \( C^{(1)}, \ldots, C^{(k)} \) (see Figure 1 in the appendix), so the clock \( C \) can be written as the concatenation of all the \( C^{(i)} \)'s, that is: \( \text{BINARY}(C) = \left( \text{BINARY}(C^{(k)}), \ldots, \text{BINARY}(C^{(1)}) \right) \). We will henceforth drop the notation \( \text{BINARY} \) because it should be clear from context if we are talking of the integer value of \( C \) or its binary expansion. So we write

\[
C = \left( C^{(k)}, \ldots, C^{(1)} \right).
\]

We refer to these blocks as subclocks. In fact, when \( C^{(i)} \neq 0 \) we view \( C^{(i)} \) as an index of \( \text{BINARY}(C^{(i+1)}) \), namely the binary description. Otherwise, if \( C^{(i)} = 0 \), we interpret \( C^{(i)} \) differently.

Let \( m_i \) denote the size of \( C^{(i)} \). Subclock \( C^{(i)} \) thus represents an integer between 0 and \( 2^{m_i} - 1 \). The sequence \( m \) is chosen as follows

\[
\begin{align*}
m_1 &= 2, \\
\forall i \leq k - 2, \ m_{i+1} &= 2^{m_i} - 1. \\
m_k &= \log T - m_1 - \ldots - m_{k-1}.
\end{align*}
\]

Each agent displays 3 bits. The first two correspond to \( C^{(1)} \), a clock modulo 4. We call the third one the rabbit-hole bit. It corresponds to the value of some bit of some clock \( C^{(i)} \) for some \( i \leq k \). The index of this bit is given by a procedure called FETCH, which we define next (see also Figure 1). The procedure takes as input a clock \( C \in [T] \) and returns an index in \([\log T]\). To decide what is the fetched index, consider the minimal \( j \) such that \( C^{(j)} \neq 0 \), which we denote \( j^* \). The fetched index is the \( C^{(j^*)} \)th bit of the subclock \( C^{(j^*+1)} \).

**Algorithm 2:** The FETCH \( (C) \) procedure

1. Write \( C = \left( C^{(k)}, \ldots, C^{(2)}, C^{(1)} \right) \) with the size of \( C^{(i)} \) equal to \( m_i \).
2. \( j^* \gets \min\{k - 1 \geq j \geq 1 \mid C^{(j)} \neq 0\} \) # with the convention \( \min\emptyset = 1 \).
3. RETURN \( m_1 + m_2 + \ldots + m_{j^*} + C^{(j^*)} \)

Now that we have specified what agents store and display, we can describe how they use the 2 × 3 bits they collect at each round, to update their clock. Suppose agent \( u \) sampled \( u_1, u_2 \) and collected \( M_{u_1} = (C^{(1)}_{u_1}, b_{u_1}) \) and \( M_{u_2} = (C^{(1)}_{u_2}, b_{u_2}) \). First, the clock \( C^{(1)}_{u_1} \) is updated by applying \( \text{BITWISE\_MAJ} \) to \( (C^{(1)}_{u_1}, C^{(1)}_{u_2}, C^{(1)}) \), as in SYNC-SIMPLE (line 6).

The location where the third bit “comes from” is updated by replacing it with the MAJ of the three corresponding bits \( \text{MAJ}(b_u, b_{u_2}, b_{u_3}) \) (line 7). The last step is incrementing the clock \( C \) by one (line 8). This step potentially affects all clocks \( C^{(k)}, \ldots, C^{(1)} \).

### 3.3 Analysis of Protocol SYNC-3BIT

First, note that Protocol SYNC-3BIT uses 3 bits of communication per interaction, as promised. We now aim to prove the first part of Theorem 1.1, namely that the protocol synchronizes clocks modulo \( T \) within \( \tilde{O}(\log^2 T \log n) \) rounds.

**Proof of first part of Theorem 1.1.** For now, we assume there are no Byzantine agents. Recall the sequence \( m_1, \ldots, m_k \in \mathbb{N} \) satisfies \( m_1 = 2 \), for all \( i \leq k - 2, m_{i+1} = 2^{m_i} - 1 \), and \( m_k = \log T - m_1 - \ldots - m_{k-1} \). We define also \( N_i := \sum_{j=1}^i m_j \). Given such a sequence, we subdivide the binary expansion of \( C \) (viewed as a string) into blocks of length \( m_1, \ldots, m_k \). In particular the length of \( \text{BINARY}(C) \) is \( N_k \) and the clock \( C \) can count up to \( 2^{N_k} = T \). Recall from 1 that we write \( C = \left( C^{(k)}, \ldots, C^{(1)} \right) \).
The next lemma gathers a few observations concerning FETCH. Informally, it guarantees that, if clocks are already synchronized on the first \(i-1\) blocks, then whenever an index from these blocks is fetched, it will be fetched by all agents (point (1)). Moreover, whenever a clock \(C\) is incremented \(2^{N_i-1}\) times in a row, the FETCH function will take all values in \([N_i-1 + 1, N_i]\) (points (2 - 3)). The proof is deferred to Appendix A.2

**Lemma 3.2** (About FETCH). Let \(C, C'\) be clocks modulo \(T\).

1. If \(C\) and \(C'\) agree on the first \(i-1\) subclocks and \(\text{FETCH}(C) \neq \text{FETCH}(C')\) then \(\text{FETCH}(C), \text{FETCH}(C') \geq N_{i+1}\). In other words, the fetched indices in \(C\) and \(C'\) are from a block strictly greater than \(i\).

2. For any \(x \in [1, m_i]\), if \(\text{BINARY}(C)\) is of the form \((\ldots, \text{BINARY}(x), 0^{N_i-2})\) then \(\text{FETCH}(C) = N_{i-1} + x\).

3. Let \(C'_1, \ldots, C'_{2^{N_i-1}} \in [0, T - 1]\) be a sequence of values, such that \(C'_j = j + c\mod 2^{N_i-1}\) (for some fixed \(c \in \mathbb{N}\)). For instance we would get such a sequence \(C'\) by considering \(C, C+1, \ldots, C+2^{N_i-1}\). Then, all values in the interval \([N_{i-1} + 1, N_i]\) are fetched when \(\text{FETCH}\) runs over the sequence \(C'\). In symbols, \([N_{i-1} + 1, N_i] \subseteq \{\text{FETCH}(C'_1), \text{FETCH}(C'_2), \ldots, \text{FETCH}(C'_{2^{N_i-1}})\}\).

**Lemma 3.3** (Running time). Let \(i \leq k\). The protocol \textsc{SYNC-3BIT} is a self-stabilizing protocol that synchronizes the first \(i\) subclocks in time \(O(\log n \cdot \sum_{j \leq i} m_j \cdot 2^{N_j-1})\).

**Proof of Lemma 3.3**. Recall that the first 2 bits (corresponding to the small clock \(C^{(1)}\)) are being synchronized from any initial configuration, according to \textsc{SYNC-SIMPLE}, so the lemma holds if \(i = 1\) by Section 3.1.

Consider any configuration such that the first \(i-1\) subclocks are synchronized. In other words, the counters \(C\) of different agents agree on the first \(N_{i-1}\) bits. We want to bound the time it takes to reach a configuration in which the first \(i\) clocks are synchronized.

First we can observe that once the first \(i-1\) clocks are synchronized, they can never be unsynchronized. Indeed, for an agent to unsynchronize on a bit from the first \(i-1\) blocks, it has to be that at some point she is displaying this bit at the rabbit-hole, while at least one other agent is displaying a bit with a different index. But point (1) in Lemma 3.2 asserts that this will not happen since we assume that agents clocks’ already coincide on the first \(i-1\) subclocks.

Consider the first time when agents agree on the first \(N_{i-1}\) bits. Let \(c\) be the value of their clocks modulo \(2^{N_i-1}\) at that moment. During the next \(2^{N_i-1}\) rounds the clocks of all agents modulo \(2^{N_i-1}\), jointly take values \(c, c+1, \ldots, c+2^{N_i-1} - 1\) and so we know from Lemma 3.2 points (2 - 3) that agents will display (at the rabbit-hole) the bit with index \(N_i + x\) at least once (in fact exactly once), for each \(x \in [1, m_i]\). Moreover, they do so at the same time (it also follows from Lemma 3.2 point (1)). Thus, by assumption, after \(O(\log n)2^{N_i-1}\) rounds, the first bit is synchronized. By the same argument as in the proof of \textsc{SYNC-SIMPLE} in Section 3.1 after \(O(x \cdot \log n \cdot 2^{N_i-1})\) rounds, the first \(x\) bits of \(C^{(i)}\) are synchronized. Altogether, the time to synchronize the \(i\)th subclock after synchronizing the first \(i-1\) is \(O(log n \cdot m_i \cdot 2^{N_i-1})\). The result follows.
For a given $T$ and corresponding sequence $m_1 = 2, \ldots m_k$, we just need to estimate the term $\sum_{i=1}^{k} m_i 2^{N_i - 1}$, given that $\log T = N_k = m_1 + \ldots + m_k$. We want to show that it is of order $O(\log^2 T)$ and this will conclude the proof by applying Lemma 3.3. The following corollary establishes the required running time of the protocol, by expressing $\sum_{j=k}^{m} m_j \cdot 2^{N_j - 1}$ and $k$ as a function of $T$. Its simple proof is deferred to Appendix A.3.

**Corollary 3.4.** With the previous definitions, the sum $\sum_{j=k}^{m} m_j \cdot 2^{N_j - 1}$ is of order $O(\log^2 T)$, and so the running time of SYNC-3BIT is $O\left(\log^2 T \log n\right)$.

To conclude the proof of the first part of Theorem 1.1, we now analyze the resilience of the protocol to Byzantine failures. In fact, this resilience follows from the CONSENSUS protocol and the analysis done in [22]. Indeed, if there are $\kappa \leq n^{1/2-\varepsilon}$ Byzantine agents, Theorem 2.1 asserts that they don’t harm the convergence of CONSENSUS, and don’t have any effect even up to $polylog(n)$ rounds after convergence, with probability $1 - n^{-\Omega(1)}$. Thus we can apply this result for the convergence along any coordinate and recover the same guarantee for our protocol SYNC-3BIT. We apply CONSENSUS once for each of the $\log T$ coordinates of the clock, so using a union bound, the result holds with probability $\geq 1 - \log T n^{-\Omega(1)} = 1 - n^{-\Omega(1)}$ (since we assume $T \leq n^{O(1)}$). This concludes the proof of the first part of Theorem 1.1.

We now turn to the second bullet of Theorem 1.1. Namely we show a simple way to reduce the running time, if we know an explicit bound on the running time of CONSENSUS, say $c' \log n$ for some explicit $c' \in \mathbb{R}$. Agents build a “small clock” $C_{\text{small}}$ using Protocol SYNC-3BIT. This small clock runs modulo $c \log T \log n$ (where $c \geq c'$ is chosen such that this value is a power of 2). This is done using 3 bits per interaction. This clock is viewed as an index for a clock modulo $T$ (which has $\log T$ bits). This latter clock is denoted $C_{\text{big}}$. In the same spirit as in SYNC-3BIT, the fourth bit displayed, corresponds to the first bit of $C_{\text{big}}$ whenever $C_{\text{small}} \in [1, c \log n]$, to its second bit whenever $C_{\text{small}} \in [c \log n + 1, 2c \log n]$ and more generally to its $i$th bit whenever $C_{\text{small}} \in [i \cdot c \log n + 1, (i+1) \cdot c \log n]$. Again CONSENSUS is run on the corresponding bit and the big clock is incremented on every round.

Correctness follows from the same reasons as in the proof of the first part of Theorem 1.1. Let us now evaluate the running time. It takes $O(\log n \cdot (\log(\log n \log T))^2) = \tilde{O}(\log n \log T^2)$ to synchronize $C_{\text{small}}$. Then it takes another $O(\log n) \cdot \log T$ to do a complete loop over the indices of $C_{\text{big}}$, that starts at 1 and ends at $\log T$. Convergence is now guaranteed by time $\tilde{O}(\log n (\log T)^2) + O(\log n \log T) = O(\log n \log T)$.

## 4 Self-stabilizing Zealot-consensus

In this section, we prove Theorem 1.2, stating that there exists a Zealot-consensus algorithm that uses only 4 bits in each message. The protocol guarantees that, w.h.p., all agents reach consensus in $\tilde{O}(\log n)$ rounds, even in the presence of at most one Zealot agent. (Recall that a Zealot is an agent whose output bit is frozen, yet, all other bits of its internal state can be controlled by an adversary.)

We now present our Zealot-consensus protocol. Agents internally hold a clock modulo $T$. We take $T$ to be a power of 2, and $T = c \log n$, where the constant $c \in \mathbb{R}$ is chosen to be big enough so that the process of rumour spreading in the randomized PULL model finishes in time $T/2$ w.h.p. (see [37]). To agree on the value of $T$, agents need to know $n$, and this is the only place where we use this assumption. We know from Theorem 1.1 that, by displaying only 3 bits in each message, all non-Zealot agents can synchronize these clocks in $\tilde{O}(\log n \log^2 \log n) = \tilde{O}(\log n)$ rounds in a self-stabilizing manner (the Zealot, if it exists, playing the role of a Byzantine agent). The fourth bit is the one agents want to agree on, namely their output bit. Agents divide time into phases of length $T$, each of them being further subdivided into 2 subphases of length $T/2$. In the first subphase of each phase, agents are sensitive to opinion 0. This means that whenever they see a 0 in the output bit of another agent (amongst those they sample), they turn their output bit to 0. Then, in the second phase of each phase, they do the opposite, namely they switch their output bit to 1 as soon as they see a 1 (see Figure 2 in the Appendix). Of course, the Zealot is not necessarily behaving like this.
Consider first the case that there is a Zealot in the population. Once the clocks are synchronized, if the opinion of the Zealot agent is 0 then within one complete subphase $[0, T/2 - 1]$, every opinion is 0 and remains there forever. Otherwise, if the opinion of the Zealot is 1, when all agents go over a subphase $[T/2, T - 1]$ all opinions are set to 1 and remain 1 forever. Consider now the case that there is no Zealot in the population. Let $t_0$ be the first time when all clocks are synchronized and showing the time 0 modulo $T$. If at time $t_0$ there is an agent whose output bit is 0, then within $T/2$ rounds, all agents set their output bit to 0, and remain like that forever. Otherwise, all agent are 1 at time $t_0$, and will remain in that output configuration forever.

5 Generalizations: From consensus to clock synchronization

In this section, we informally comment on the degree to which our clock synchronization results can be generalized to other models of communication, and other consensus protocols instead of CONSENSUS.

The technique we used in Section 3.2 transforming the protocol CONSENSUS in the randomized PULL model to a clock synchronization protocol SYNC-3BIT in the same model can be generalized. More specifically, consider a consensus protocol $\mathcal{A}$ operating in a synchronous communication model $\mathcal{M}$. Under some conditions, our technique can be used to transform $\mathcal{A}$ into a clock synchronization protocol $\text{SYNC-3BIT}(\mathcal{A})$. The resulting protocol maintains the same asymptotic message size as $\mathcal{A}$ while paying only a small penalty in the running time. Moreover, if $\mathcal{A}$ is self-stabilizing then so is the clock synchronization algorithm $\text{SYNC-3BIT}(\mathcal{A})$. The properties we need from $(\mathcal{M}, \mathcal{A})$ are the following:

- **Passive and time invariant communication:** We assume that the meeting pattern is independent of the algorithm and time invariant, that is, the distribution of who meets who at each round does not depend on the round number nor on the agents’ states.

- **Symmetric:** An execution of $\mathcal{A}$ in $\mathcal{M}$ commutes with respect to flipping: Fix a meeting pattern and fix the random coins of the agents. If we flip all output bits from round $i$ and run the protocol for one more round, output bits as well as internal states at round $i + 1$ are the same as when running the protocol in round $i$ without flipping, and then flipping the output bits at the end of round $i + 1$.

Indeed, the only place where we use protocol CONSENSUS in the proof of Theorem 1.1 is in Lemma 3.3, where we use only the aforementioned properties of the PULL model and CONSENSUS. Recall that the clock synchronization algorithm aims to synchronize the $i$’th bit of the clock in the rounds where the rabbit-hole bit displays the $i$’th bit of the clock. The first property guarantees that, no matter on which sequence of rounds $t_1 < t_2 < \ldots < t_r$ the agents aim to synchronize their $i$’th bit, if no flipping of bits occur, then, when running $\mathcal{A}$, the output distribution would be the same as if the algorithm ran in rounds $1, 2, \ldots, r$. The second property guarantees that the convergence of the consensus algorithm, as simulated by the clock synchronization protocol, is not affected by the flipping of output bits, as long as they are all flipped at the same time.

Acknowledgments: The problem of self-stabilizing Zealot-consensus as considered in this paper was introduced through discussions with Ofer Feinerman. The authors are also thankful for Omer Angel, Bernhard Haeupler, Parag Chordia, Iordanis Kerenidis, Fabian Kuhn, Uri Feige, and Uri Zwick for helpful discussions regarding that problem. The authors also thank Michele Borassi for his helpful suggestions regarding the clock synchronization problem.
References


Appendix

A.1 Figure 1

Figure 1: This figure describes a clock subdivision (left) and the FETCH mechanism. a) \( j^* = 1 \) If \( C^{(1)} \neq 0 \), the third public bit is the \( C^{(1)} \)-th bit of subclock 2. b) \( j^* = 2 \) If the value of subclock 1 is \( C^{(1)} = 0 \) and \( C^{(2)} \neq 0 \), the third bit is the \( C^{(2)} \)-th bit of subclock 3. c) \( j^* = 3 \) If \( C^{(1)} = C^{(2)} = 0 \) and \( C^{(3)} \neq 0 \), the third bit is the \( C^{(3)} \)-th bit of subclock 4.

A.2 Proof of Lemma 3.2

Proof of point (1) Consider \( C \in [T] \) and \( C' \in [T] \) that agree on the first \( i \) blocks. If

\[
j^*(C) := \min\{j \mid C^{(j)} \neq 0\} \leq i - 1,
\]

then \( j^*(C) = j^*(C') \), and \( \text{FETCH}(C) = \text{FETCH}(C') \) because \( \text{FETCH}(C) \) and \( \text{FETCH}(C') \) are determined by block \( j^* + 1 \leq i \). Therefore, if \( \text{FETCH}(C) \neq \text{FETCH}(C') \), it has to be that \( j^*(C), j^*(C') \geq i \) and so the fetched indices are from some block \( \geq i + 1 \), which implies the result.

Proof of point (2) Let \( x \in [1, m_i] \), and assume \( C \) is of the form \((\ldots \text{BINARY}(x), 0^{N_{i-2}})\) then

\[
j^*(C) = \min\{j \mid C^{(j)} \neq 0\} = i,
\]

and the index to be fetched is thus given by the \( i \)th subclock, which has value \( x \).

Proof of point (3) This follows from the previous point. The length of \( \text{BINARY}(x) \) is \( m_{i-1} \), thus the length of \((\text{BINARY}(x), 0^{N_{i-2}})\) is \( N_{i-2} + m_{i-1} = N_{i-1} \) and by assumption all patterns \((\ldots \text{BINARY}(y), 0^{N_{i-2}})\) for all \( y \in [1, m_i] \) appear in \( C' \). We get the conclusion using the fact that \( N_i = N_{i-1} + m_i \).
A.3 Proof of Corollary 3.4

Recall the definition of the sequence \( m_i \), Equation 2 in Section 3.2. Starting with \( m_k \leq \log T \), we know that \( m_{k-1} = \lceil \log (1 + \log m_k) \rceil \). Thus \( m_{k-1} \) is at most \( \lceil \log (1 + \log T) \rceil \). In fact we can bound the sequence \( m_i \) by iterating \( i \) times \( \lceil \log (1 + X) \rceil \). It follows that the integer \( k \) is smaller than \( g(T) \), the number of times one needs to iterate \( \lceil \log (1 + X) \rceil \) before reaching 1.

Claim A.1. Let \( f, g : \mathbb{R}_+ \to \mathbb{R} \) be functions defined by \( f(x) = \lceil \log (1 + x) \rceil \) and

\[
g(x) = \inf \left\{ k \in \mathbb{N} \mid f^{\otimes k}(x) \leq 1 \right\},
\]

where we denote by \( f^{\otimes k} \) the \( k \)-fold iteration of \( f \). It holds that

\[
g(T) \leq \log^{\otimes 4} T + O(1).
\]

Proof. We can notice that

\[
f(T) \leq T - 1,
\]

if \( T \) is bigger than some constant \( c \). Moreover, when \( f(x) \leq c \), the number of iterations before reaching 1 is \( O(1) \). This implies that

\[
g(T) \leq T + O(1).
\]

But in fact, by definition, \( g(T) = g \left( f^{\otimes 4}(T) \right) + 4 \) (provided \( f^{\otimes 4}(T) > 1 \), which holds if \( T \) is big enough). Hence

\[
g(T) \leq g \left( f^{\otimes 4}(T) \right) + 4 \leq f^{\otimes 4}(T) + O(1) \leq \log^{\otimes 4} T + O(1).
\]

\[\square\]

From this lemma we get that

\[
m_k 2^{N_{k-1}} \leq \log T \cdot 2^{f^{\otimes 2}(T)} 2^{f^{\otimes 3}(T)} 2^{f^{\otimes 4}g(T)}
\]

\[
= \tilde{O} \left( \log T \cdot 2^{\log \log T + \log \log \log T + \log^{\otimes 4}(T) g(T)} \right)
\]

\[
= \tilde{O} \left( \log T \cdot 2^{\log \log T + (\log^{\otimes 4}(T))^2} \right)
\]

\[
= \tilde{O} \left( \log^2 T \cdot 2^{\log^{\otimes 3}(T)} \right) = \tilde{O}(\log^2 T).
\]

In particular it follows from Lemma 3.3 that the overhead is at most \( \sum_{i=1}^{k} m_i 2^{N_{i-1}} \leq km_k 2^{N_{k-1}} \), and since \( k = g(T) \), we know from Lemma A.1 that \( km_k 2^{N_{k-1}} = \tilde{O}(g(T) \log^2 T) = O(\log^2 T) \).
A.4 Figure: The wheel of time

![Diagram of the wheel of time]

Figure 2: The wheel of time used for the stable broadcast. During the first half, between times 0 and $T/2 - 1$ agents are sensitive to 0. Then they are sensitive to 1.

A.5 Experimental results, the power of one bit

Initially, part of the motivation for this work was trying to solve problems like ZEALOT-CONSENSUS, in the extremely restricted randomized PULL model of communication, using only 1 bit of communication, namely, the output bit. The main difficulty arises from the self stabilization requirement.

In fact, we were able to find a few heuristics that work well in simulations. These algorithms however don’t seem amenable to analysis (they are not memory-less, and seem to lack structure). Moreover a proof of self stabilization should consider any possible starting conditions.

A.5.1 A first candidate, the BSF Protocol

Perhaps the simplest example of such a protocol is the following\footnote{A similar protocol was suggested during discussions with Bernhard Haeupler.}. It has 2 parameters $\ell, k \in \mathbb{N}$. Agents can be in 3 states: boosting, sensitive, or frozen.

1. **Boosting** agents behave as in the CONSENSUS protocol. They apply the majority rule $\text{MAJ}$ to the 2 values they see in a given round and make it into their opinion for the next round. They also keep a counter $T$. If they have seen only agents of a given color $b$ for $\ell$ rounds, they become sensitive to the opposite value.

2. **Sensitive** to $b$ agents turn into frozen-$b$ agents if they see value $b$.

3. **Frozen-$b$** agents keep the value $b$ for $k$ rounds before becoming boosters again.

*Heuristic discussion*: A self stabilization proof has to consider all possible initial states (which includes counter values). Heuristically, what we expect is that, from every configuration, at some point almost all nodes would be in boosting state. Then, the boosting behavior would lead the agents to converge to a value $b$ (which depends on the initial conditions). Most agents should then become sensitive to $1 - b$. If the Zealot has opinion $1 - b$ then there should be a “switch” from $b$ to $1 - b$. The “frozen” period is meant to allow for some delay in the times at which agents become sensitive, and then flip their opinion. The parameters probably need to scale like $k, \ell = O(\log n)$.
Figure 3: A few simulations of the BSF protocol with $n = 2400$, $k = 30$, $\ell = 10$. The $x$-axis is the number of rounds and the $y$-axis the number of agents disagreeing with the source. Different curves correspond to different initializations.

### A.5.2 A second candidate, the Minority Protocol

In this paragraph, we present another candidate protocol to solve ZEALOT-CONSENSUS in the PULL model. The protocol has a parameter $r$. Each agent $u$ has a counter $C_u$ and three bits: its opinion $C(u)$, its susceptibility $S(u)$ and its next-susceptibility $W(u)$. At each round, each agent $u$ looks at the opinion of three neighbors $b_1, b_2$ and $b_3$, and performs the following actions.

1. If either $b_1, b_2$ or $b_3$, is equal to $S(u)$, then set $C(u) = S(u)$.
2. Decrease $C_u$ by 1.
3. If $C_u = 0$ then set $C_u = r + X_u$ where $X_u$ is a uniform random variable over the integers $\{1, \ldots, r\}$; then set $S(u) = W(u)$ and finally

$$W(u) = 1 - \text{MAJ \{b_1, b_2, b_3\}}.$$  

(3)

**Heuristic discussion:** As for the BSF protocol, the core idea of the Minority protocol is to produce an ever-growing oscillating behaviour toward the two “unanimous” configurations in which all nodes agree on an opinion, in such a way that eventually the system “gets stuck” in the configuration that agrees with the Zealot’s opinion. To produce such behaviour, a natural candidate is the complement of the majority rule of $[22]$. In fact, the majority rule has a strong drift toward the most frequent opinion in the system; thus, by complementing its output at each round, we get a mechanism that exhibits a strong drift toward the opposite opinion to the most frequent one in the system. This mechanism alone, then, would produce an oscillation between the two unanimous configurations. To allow the system to get stuck on the unanimous configuration that corresponds to the Zealot’s opinion, we then need to provide the nodes an additional mechanism that makes them cautious in changing their opinion. A natural idea is then to make them wait before starting to pick the opinion suggested by the complement of the majority mechanism. Thanks to this delay, if the system has hit the unanimous configuration that agrees with the Zealot’s opinion, then when the nodes finally start to be susceptible to the opposite opinion there would be no one to trigger such spreading process. On the contrary, if the configuration disagrees with the Zealot’s opinion, at some point the spreading process will be triggered and the process will tend to the opposite configuration. Note that, since the
rumor spreading takes $\Theta(\log n)$ to converge to an unanimous configuration (see [37]), the parameter $r$ here should scale like $\log n$.

Figure 4: A few simulations of the Minority protocol with $n = 600$, $r = 6$. The $x$-axis is the number of rounds and the $y$-axis the number of agents disagreeing with the source. Different curves correspond to different initializations.