Stone Age Distributed Computing

Yuval Emek Jasmin Smula Roger Wattenhofer

Computer Engineering and Networks Laboratory (TIK)
ETH Zurich, Switzerland

1 Introduction

Networks are at the core of many scientific areas, be it social sciences (where networks for instance model human relations), logistics (e.g. traffic), or electrical engineering (e.g. circuits). *Distributed computing* is the area that studies the power and limitations of distributed algorithms and computation in networks. Due to the major role that the Internet plays today, models targeted at understanding the fundamental properties of networks focus mainly on "Internet-capable" devices. The standard model in distributed computing is the so called *message passing* model, where nodes may exchange large messages with their neighbors, and perform arbitrary local computations.

Some networks though, are not truthfully represented by the classical message passing model. For example, *wireless* networks such as ad hoc or sensor networks, whose research has blossomed in the last decade, require some adaptations of the message passing model so that it meets the limited capabilities of the underlying wireless devices more precisely. More recently, there is a trend to apply distributed computing methods, and in particular, the message passing model, to networks of sub-microprocessor devices, for instance networks of biological cells, or even nano-scale mechanical devices. However, the suitability of the message passing model to these types of networks is far from being certain: do tiny bio/nano nodes "compute" and/or "communicate" essentially the same as a computer? Since such nodes will be fundamentally more limited than silicon-based devices, we believe that there is a need for a network model, where nodes are by design below the computation and communication capabilities of Turing machines.

Networked finite state machines. In this paper, we take a radically different approach: Instead of imposing additional restrictions on the existing models for networks of computer-like devices, we introduce an entirely new model, referred to as *networked finite state machines (nFSM)*, that depicts a network of randomized finite state machines progressing in asynchronous steps (refer to Section 2 for a formal description). Under the nFSM model, nodes communicate by transmitting messages belonging to some finite alphabet Σ such that a message $\sigma \in \Sigma$ transmitted by node u is

delivered to its neighbors (the same σ to all neighbors) in an asynchronous fashion; each neighbor v of u has a port corresponding to u in which the last message delivered from u is stored.

The access of node v to its ports is limited: each state q in the state set Q of the FSM is associated with some letter $\sigma = \sigma(q) \in \Sigma$; if node v resides in state q at some step of the execution, then the next state and the message transmitted by v at this step are determined by q and by the number $\sharp_\sigma$ of occurrences of σ in v's ports. The crux of the model is that $\sharp_\sigma$ is calculated according to the *one-two-many*[1] principle: the node can only count up to some predetermined bounding parameter $b \in \mathbb{Z}_{>0}$ and any value of $\sharp_\sigma$ larger than b cannot be distinguished from b.

In particular, the nFSM model satisfies the following requirements, that we believe, make it more applicable to the study of networks consisting of weaker devices such as those mentioned above.

(R1) The model is applicable to arbitrary network topologies.

(R2) All nodes run the same protocol executed by an FSM.

(R3) The network operates within an asynchronous environment, with node activation patterns independent of the message delivery pattern.

(R4) All features of the FSM (specifically, the state set Q, message alphabet Σ, and bounding parameter b) are of constant size independent of any parameter of the network (including the degree of the node executing the FSM).

The last requirement is perhaps the most interesting one as it implies that a node cannot count beyond some predetermined constant. This comes in contrast to many distributed algorithms operating under the message passing model that strongly rely on the ability of a node to count up to some parameter of the network (or a function thereof).

Our results. Our investigation of the new model begins by designing an nFSM synchronizer that practically allows the protocol designer to assume a fully synchronous environment. Subsequently, we show that the computational power of a network operating under the nFSM model is essentially equivalent to that of a randomized Turing machine with linear space bound (cf. linear bounded automata). In comparison, the computational power of a network operating under the message passing model is trivially equivalent to that of a (general) Turing machine, therefore there exists distributed problems that can be solved (theoretically) under the message passing model in constant time and cannot be solved under the nFSM model at all.

Nevertheless, we show that some of the most important and extensively studied problems in distributed computing admit efficient — namely, with run-time polylogarithmic in the number n of nodes — protocols operating under the nFSM model. Specifically, a maximal independent set (MIS) can be computed in arbitrary graphs in time $O(\log^2 n)$; trees can be 3-colored in time

[1] The one-two-many theory states that some small isolated cultures (e.g., the Piraha tribe of the Amazon [9]) did not develop a counting system that goes beyond 2. This is reflected in their languages that include words for "1", "2", and "many" that stands for any number larger than 2.

$O(\log n)$ (without any predetermined edge orientation); and a maximal matching can be computed in arbitrary graphs in time $O(\log^2 n)$ (this requires a small unavoidable modification of the nFSM model).

Related work. As mentioned above, the message passing model is the gold standard when it comes to understanding distributed algorithms. Several variants exist for this model, differing mainly in the bounds imposed on the message size and the level of synchronization. Perhaps the most popular message passing variants are the fully synchronous *local* and *congest* models [10, 13], assuming that in each round, a node can send messages to its neighbors (different messages to different neighbors), receive and interpret the messages sent to it from its neighbors, and perform arbitrary local computation[2] determining, in particular, the messages sent in the next round. The difference between the two variants is cast in the size of the communicated messages: the local model does not impose any restrictions on the message size, hence it can be used for the purpose of establishing general lower bounds, whereas the congest model is more information-theoretic, with a (typically logarithmic) bound on the message size. Indeed, most theoretical literature dealing with distributed algorithms relies on one of these two models.

As the congest model still allows for sending different messages to different neighbors in each round, it was too powerful for many settings. Instead, with the proliferation of wireless networks, new more restrictive message passing models appeared such as the *radio network* model [5]. In radio networks, nodes still operate in synchronous rounds, where in each round a node may choose to transmit a message or stay silent. A transmitted message is received by all neighbors in the network if the neighbors do not experience interference by concurrently transmitting nodes in their own neighborhood. There are several variants, e.g. whether nodes do have collision detection, or not.

Since the radio network model is still too powerful for some wireless settings, more restrictive models were suggested. One such example is the *beeping* model [7, 6], where in each round a node can either beep or stay silent, and a silent node can only distinguish between the case in which no node in its neighborhood beeps and the case in which at least one node beeps. Inspired by a biological process that occurs during the development of the nervous system of a fly, [2] developed a distributed MIS algorithm that essentially works under the beeping model; this was later improved [1]. Note that the beeping model resembles our nFSM model in the sense that the "beeping rule" can be viewed as counting under the one-two-many principle with bounding parameter $b = 1$. However, it is much stronger in other perspectives: (i) the beeping model assumes synchronous communication and does not seem to have a natural asynchronous variant, thus it does not satisfy requirement (R3); and (ii) the local computation is performed by a Turing machine whose memory

[2] It is important to point out that even though the local and congest models allow for arbitrary local computation, the proposed algorithms hardly ever assume anything that cannot be computed in time polynomial in the size of the information received thus far; the rare exceptions are typically clearly mentioned in the text.

is allowed to grow with the network (this is crucial for the algorithms of [2, 1]), thus it does not satisfy requirements (R2) and (R4). The assumption that each node executes a FSM is especially relevant in the biological setting as demonstrated by [3].

Our nFSM model is partially inspired by the extensively studied *cellular automaton* model [12, 8, 14] that captures a network of FSMs, arranged in a grid topology (or other highly regular topology), where the transition of each node depends on its current state and the states of its neighbors. Still, the nFSM model differs from the cellular automaton model in many aspects; in particular, the latter model is not applicable for non-regular network topologies, in contrast to requirement (R1), and to the most part, it also does not support asynchronous environments (at least not as asynchrony is grasped in the current paper), in contrast to requirement (R3).

Another model that resembles the nFSM model that of *communicating automata* [4]. This model also assumes that each node in the network operates a FSM in an asynchronous manner, however the steps of the FSMs are message driven: for each state q of node v and for each message m that node v may receive from an adjacent node u while residing in state q, the transition function of v should have an entry characterized by the 3-tuple (q, u, m) that determines its next move. As such, different nodes would typically operate different FSMs, hence the model does not satisfy requirement (R2), and the size of the FSM operated by node v inherently depends on the degree of v, hence it does not satisfy requirement (R4). Moreover, the node activation pattern is driven by the incoming messages, so it also does not satisfy requirement (R3).

Structure of the paper. A formal exposition of the nFSM model is provided in Section 2. In Section 3, we design a synchronizer for the nFSM model and then show how multiple letter queries can be implemented. The characterization of the computational power of a network under the nFSM model and the nFSM algorithms for MIS, tree coloring, and maximal matching are deferred to the full version of the paper.

2 Model

Throughout, we assume a network represented by a finite undirected graph $G = (V, E)$. Under the *networked finite state machines (nFSM)* model, each node $v \in V$ runs a protocol depicted by the 8-tuple

$$\Pi = \langle Q, Q_I, Q_O, \Sigma, \sigma_0, b, \lambda, \delta \rangle ,$$

where

- Q is a finite set of *states*;
- $Q_I \subseteq Q$ is the subset of *input states*;

- $Q_O \subseteq Q$ is the subset of *output states*;
- Σ is a finite *communication alphabet*;
- $\sigma_0 \in \Sigma$ is the *initial letter*;
- $b \in \mathbb{Z}_{>0}$ is a *bounding parameter*; let $B = \{0, 1, \ldots, b-1, b^{\geq}\}$ be a set of $b+1$ distinguishable symbols;
- $\lambda : Q \to \Sigma$ assigns a *query letter* $\sigma \in \Sigma$ to every state $q \in Q$; and
- $\delta : Q \times B \to 2^{Q \times (\Sigma \cup \{\varepsilon\})}$ is the *transition function*.

It is important to point out that protocol Π is oblivious to the graph G. In fact, the number of states in Q, the size of the alphabet Σ, and the bounding parameter b are all assumed to be universal constants, independent of any parameter of the graph G. In particular, the protocol executed by node $v \in V$ does not depend on the degree of v in G. We now turn to describe the semantics of the nFSM model.

Communication. Node v communicates with its adjacent nodes in G by *transmitting* messages. A transmitted message consists of a single letter $\sigma \in \Sigma$ and it is assumed that this letter is delivered to all neighbors u of v. Each neighbor u has a *port* $\psi_u(v)$ (a different port for every adjacent node v) in which the last message σ received from v is stored. At the beginning of the execution, all ports store the initial letter σ_0. It will be convenient to consider the case in which v does not transmit any message (and hence does not affect the corresponding ports of the adjacent nodes) as a transmission of the special symbol ε.

Execution. The execution of node v progresses in discrete *steps* indexed by the positive integers. At each step $t \in \mathbb{Z}_{>0}$, v resides in some state $q \in Q$. Let $\lambda(q) = \sigma \in \Sigma$ be the query letter that λ assigns to state q and let $\sharp_\sigma$ be the number of occurrences of σ in v's ports at step t. Then, the pair (q', σ') of state $q' \in Q$ in which v resides at step $t+1$ and message $\sigma' \in \Sigma \cup \{\varepsilon\}$ transmitted by v at step t (recall that ε indicates that no message is transmitted) is chosen *uniformly at random* among the pairs in

$$\delta(q, f_b(\sharp_\sigma)) \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \ ,$$

where $f_b : \mathbb{Z}_{\geq 0} \to B$ is defined as

$$f_b(\sharp_\sigma) = \begin{cases} \sharp_\sigma & \text{if } 0 \leq \sharp_\sigma \leq b-1 \ ; \\ b^{\geq} & \text{otherwise} \ . \end{cases}$$

Informally, this can be thought of as if v queries its ports for occurrences of σ and "learns" the exact value of $\sharp_\sigma$ as long as it is smaller than the bounding parameter b; otherwise, v merely "learns" that $\sharp_\sigma \geq b$ which is indicated by the symbol $b^{\geq}$.

Input and output. Initially (at step 1), each node resides in some of the input states in Q_I. The choice of the initial state of node $v \in V$ reflects the input passed to v at the beginning of the execution. This allows our model to cope with distributed problems in which different nodes get different input symbols. When dealing with problems in which the nodes do not get any initial input (such as the graph theoretic problems addressed in this paper), we shall assume that Q_I contains a single *initial* state.

We say that the (global) execution of the protocol is in a *ready* configuration if all nodes reside in output states of Q_O. If this is the case, then the output of node $v \in V$ is determined by the output state $q \in Q_O$ in which v resides.

Asynchrony. The nodes are assumed to operate in an *asynchronous* environment. This asynchrony has two facets: First, for the sake of convenience, we assume that the actual application of the transition function in each step $t \in \mathbb{Z}_{>0}$ of node $v \in V$ is instantaneous (namely, lasts zero time) and occurs at the end of the step;[3] the length of step t of node v, denoted $L_{v,t}$, is defined as the time difference between the application of the transition function at step $t-1$ and that of step t. It is assumed that $L_{v,t}$ is finite, but apart from that, we do not make any further assumptions on this length, that is, the step length $L_{v,t}$ is determined by the adversary independently of all other step lengths $L_{v',t'}$. In particular, we do not assume any synchronization between the steps of different nodes whatsoever.

Another facet of the asynchronous environment is that a message transmitted by node v at step t (if such a message is transmitted) is assumed to reach the port $\psi_u(v)$ of an adjacent node u after a finite time delay, denoted $D_{v,t,u}$. We assume that if v transmits message $\sigma_1 \in \Sigma$ at step t_1 and message $\sigma_2 \in \Sigma$ at step $t_2 > t_1$, then σ_1 reaches u before σ_2 does. Apart from this "FIFO" assumption, we do not make any other assumptions in the context of the delays $D_{v,t,u}$. In particular, this means that under certain circumstances, the adversary may overwrite message σ_1 with message σ_2 in port $\psi_u(v)$ of u so that u will never "know" that message σ_1 was transmitted.[4]

Consequently, a *policy* of the adversary is captured by: (1) the length $L_{v,t}$ of step t of node v for every $v \in V$ and $t \in \mathbb{Z}_{>0}$; and (2) the delay $D_{v,t,u}$ of the delivery of the transmission of node v at step t to an adjacent node u for every $v \in V$, $t \in \mathbb{Z}_{>0}$, and $u \in N(v)$.[5] Assuming that the adversary is oblivious to the random coin tosses of the nodes, an adversarial policy is depicted by infinite sequences of $L_{v,t}$ and $D_{v,t,u}$ parameters.

[3] This assumption can be lifted at the cost of a more complicated definition of the adversarial policy described soon.

[4] Often, much stronger assumptions are made in the literature. For example, a common assumption for asynchronous environments is that the port of node u corresponding to the adjacent node v is implemented by a buffer so that messages cannot be "lost". We do not make any such assumption for our nFSM model.

[5] We use the standard notation $N(v)$ for the *neighborhood* of node v in G, namely, the subset of nodes adjacent to v.

For further information on asynchronous environments, we point the reader to one of the standard textbooks [13, 11].

Correctness and run-time measures. A protocol Π for problem P is said to be *correct* under the nFSM model if for every instance of P and for every adversarial policy, Π reaches a ready configuration within finite time with probability 1, and for every ready configuration reached by Π with positive probability, the output of the nodes is a valid solution to P. Given a correct protocol Π, the complexity measure that interests us in the current paper is the *run-time* of Π defined as follows.

Consider some instance $\mathcal{I}$ of problem P. Given an adversarial policy $\mathcal{A}$ and a sequence (actually an n-tuple of sequences) $\mathcal{R}$ of random coin tosses that lead to a ready configuration within finite time, the run-time $T_\Pi(\mathcal{I}, \mathcal{R}, \mathcal{A})$ of Π on $\mathcal{I}$ with respect to $\mathcal{R}$ and $\mathcal{A}$ is defined as the (possibly fractional) number of *time units*[6] that pass from the beginning of the execution until the first time the protocol reaches a ready configuration, where a time unit is defined to be the maximum among all step length parameters $L_{v,t}$ and delivery delay parameters $D_{v,t,u}$ appearing in $\mathcal{A}$ before the ready configuration is reached. The run-time $T_\Pi(\mathcal{I}, \mathcal{R})$ of Π on $\mathcal{I}$ with respect to $\mathcal{R}$ is then defined to be $T_\Pi(\mathcal{I}, \mathcal{R}) = \sup_{\mathcal{A}} T_\Pi(\mathcal{I}, \mathcal{R}, \mathcal{A})$. Following the standard procedure in this regard, we say that the run-time of a correct protocol Π for problem P is $f(n)$ if the run-time of Π on any n-node instance of P is at most $f(n)$ on expectation and with high probability.

3 More convenient models

3.1 Implementing a synchronizer

As described in Section 2, the nFSM model assumes an asynchronous environment. Nevertheless, it will be convenient to extend the nFSM model to *synchronous* environments. One natural such extension augments the model described in Section 2 with the following two *synchronization properties* for every two adjacent nodes $u, v \in V$ and for every $t \in \mathbb{Z}_{>0}$:
(S1) when node u is at step t, node v is at step $t-1$, t, or $t+1$; and
(S2) at the end of step $t+1$ of u, port $\psi_u(v)$ stores the message transmitted by v at step t of v's execution (or the last message transmitted by v prior to step t if v does not transmit any message at step t).
Note that properties (S1) and (S2) are sufficient to guarantee a valid execution of any protocol designed to operate in a *fully* synchronous environment, namely, in an environment in which the steps of all nodes are synchronized with the ticks — a.k.a. *rounds* — of some global clock. Therefore, for protocols operating under a synchronous environment, we shall often refer to the steps of

[6] Note that time units are defined solely for the purpose of the analysis. Under an asynchronous environment, the nodes have no notion of time and in particular, they cannot measure a single time unit.

the execution as rounds.

Our goal in this section is to show that every nFSM protocol $\Pi = \langle Q, Q_I, Q_O, \Sigma, \sigma_0, b, \lambda, \delta \rangle$ designed to operate in a synchronous environment can be simulated in an asynchronous environment by a protocol $\widehat{\Pi}$ with a constant multiplicative run-time overhead. The procedure in charge of this simulation is referred to as a *synchronizer*.

Overview. Round $t \in \mathbb{Z}_{>0}$ of node $v \in V$ under Π is simulated by $O(1)$ contiguous steps under $\widehat{\Pi}$; the collection of these steps is referred to as v's *simulation phase* of round t. Protocol $\widehat{\Pi}$ is designed so that v maintains the value of $t \pmod 3$, referred to as the *trinarity* of round t, which is also encoded in the message transmitted by v at the end of round t.[7] The main principle behind our synchronizer is that node v will not move to the simulation phase of round $t+1$ while its ports still contain messages sent in a round whose trinarity is $t-1 \pmod 3$.

Under Π, the decisions made by node v at round t should be based on the messages transmitted by all neighbors u of v at round $t-1$. However, during v's simulation phase of round t, port $\psi_v(u)$ may contain messages transmitted at round $t-1$ or at round t under Π. The latter case is problematic since the message transmitted by u in the simulation phase of round $t-1$ is overwritten by that transmitted in the simulation phase of round t. To avoid this obstacle, a message transmitted by node u under $\widehat{\Pi}$ at the end of the simulation phase of round t also encodes the message that u transmitted under Π at round $t-1$.

So, if v resides in a state whose query letter is $\sigma \in \Sigma$ at round t under Π, then under $\widehat{\Pi}$, v should query for all Σ'-letters encoding a transmission of σ at round $t-1$. Since there are several such letters, a carefully designed feature should be used so that $\widehat{\Pi}$ accounts for their combined number.

Protocol $\widehat{\Pi}$. Let

$$\widehat{\Pi} \;=\; \left\langle \widehat{Q}, \widehat{Q}_I, \widehat{Q}_O, \widehat{\Sigma}, \widehat{\sigma}_0, b, \widehat{\lambda}, \widehat{\delta} \right\rangle \;.$$

Consider node $v \in V$ and round $t \in \mathbb{Z}_{>0}$. As the name implies, node v's *simulation phase* of round t under $\widehat{\Pi}$, denoted $\phi_v(t)$, corresponds to round t of Π. Protocol $\widehat{\Pi}$ is designed so that at every step in $\phi_v(t)$ other than the last one, v does not transmit any message (indicated by transmitting ε), and at the last step of the simulation phase, v always transmits some message $\widehat{\sigma} \in \widehat{\Sigma}$, denoted $M_v(t)$.

The alphabet $\widehat{\Sigma}$ is defined to be

$$\Sigma' \;=\; (\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \times \{0,1,2\} \;.$$

The semantics of the message $M_v(t) = (\sigma, \sigma', j)$ sent by node v at the last step of the simulation phase $\phi_v(t)$ is that: v transmits $\sigma \in \Sigma \cup \{\varepsilon\}$ at round $t-1$ under Π; v transmits $\sigma' \in \Sigma \cup \{\varepsilon\}$ at round t under Π; and $j = t \pmod 3$. Following that logic, we set $\widehat{\sigma}_0 = (\varepsilon, \sigma_0, 0)$.

[7] Note that maintaining the value of $t \pmod 2$ is insufficient for the sake of reaching synchronization.

The state set $\widehat{Q}$ of $\widehat{\Pi}$ is defined to be

$$\widehat{Q} \;=\; \left(\bigcup_{q\in Q}(P_q \cup S_q)\right) \times \{0,1,2\}\ ,$$

where $P_q \times \{j\}$ and $S_q \times \{j\}$, $q \in Q$, $j \in \{0,1,2\}$, are referred to as the *pausing* and *simulating* features, respectively, whose role will be clarified soon. Suppose that v resides in state $q \in Q$ at step t under Π and that $j = t \pmod 3$. Then, throughout $\phi_v(t)$, node v resides in some state in $(P_q \cup S_q) \times \{j\}$. In particular, in the first steps of the simulation phase, v resides in states of the pausing feature $P_q \times \{j\}$, and then at some stage it switches to the simulating feature $S_q \times \{j\}$ and remains in its states until the end of the simulation phase.

The pausing feature. For the simulation phase of round t, we denote the letters in $(\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \times \{j-2\}$ as *dirty* and the letters in $(\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \times \{j-1, j\}$ as *clean*.[8] The purpose of the pausing feature $P_q \times \{j\}$ is to pause the execution of v until its ports do not contain any dirty letter. This is carried out by including in $P_q \times \{j\}$ a state $p_{\sigma,\sigma'}$ for every $\sigma, \sigma' \in \Sigma \cup \{\varepsilon\}$; the query letter of $p_{\sigma,\sigma'}$ is (the dirty letter) $\widehat{\lambda}(p_{\sigma,\sigma'}) = (\sigma, \sigma', j-2)$ and the transition function $\widehat{\delta}$ is designed so that v moves to the next (according to some fixed order) state in the feature $P_q \times \{j\}$ if and only if there are no ports storing the query letter.

We argue that the pausing feature guarantees synchronization property (S1). For the sake of the analysis, it is convenient to assume the existence of a fully synchronous simulation phase of a virtual round 0; upon completion of this simulation phase (at the beginning of the execution), every node $v \in V$ transmits the message $M_v(0) = \widehat{\sigma}_0$. We are now ready to establish the following lemma.

Lemma 3.1. *For every $t \in \mathbb{Z}_{>0}$, $v \in V$, and $u \in N(v)$, when v completes the pausing feature of $\phi_v(t)$, port $\psi_v(u)$ stores either $M_u(t-1)$ or $M_u(t)$.*

Proof. By induction on t. The base case of round $t = 0$ holds by our assumption that $\phi_v(0)$ and $\phi_u(0)$ are fully synchronous. Assume by induction that the assertion holds for round $t-1$. Applying the inductive hypothesis to both u and v, we conclude that (1) when v completes the pausing feature of $\phi_v(t-1)$, port $\psi_v(u)$ stores either $M_u(t-2)$ or $M_u(t-1)$; and (2) when u completes the pausing feature of $\phi_u(t-1)$, port $\psi_u(v)$ stores either $M_v(t-2)$ or $M_v(t-1)$.

Let τ_u and τ_v denote the times at which u and v complete the pausing feature of $\phi_u(t)$ and $\phi_v(t)$, respectively. Since v cannot complete the pausing feature of $\phi_v(t)$ while $M_u(t-2)$ is still stored in $\psi_v(u)$, it follows that at time τ_v, port $\psi_v(u)$ stores the message $M_u(t')$ for some $t' \geq t-1$. Our goal in the remainder of this proof is to show that $t' \leq t$. If $\tau_v < \tau_u$, then t' must be exactly $t-1$, which concludes the inductive step for that case.

[8] Throughout this section, arithmetic involving the parameter j is done modulo 3.

So, assume that $\tau_v > \tau_u$ and Suppose by contradiction that $t' \geq t+1$. Using the same line of arguments as in the previous paragraph, we conclude that at time τ_u, port $\psi_u(v)$ stores the message $M_v(t-1)$. Node u cannot complete the pausing feature of $\phi_u(t+1)$ while $M_v(t-1)$ is still stored in $\psi_u(v)$, hence v must have transmitted $M_v(t)$ before u completed the pausing feature of $\phi_u(t+1)$. But this means that v completed the pausing feature of $\phi_v(t)$ before u could have transmitted $M_u(t+1)$, in contradiction to the assumption that $\psi_v(u)$ stores $M_u(t')$ for some $t' \geq t+1$ at time τ_v. The assertion follows. □

Consider two adjacent nodes $u, v \in V$. If node u is at round $t-1$ when an adjacent node v is at round $t+1$, then v completed the pausing feature of $\phi_v(t)$ before u transmitted $M_u(t-1)$, in contradiction to Lemma 3.1. Therefore, our synchronizer satisfies synchronization property (S1). Furthermore, a similar argument shows that between the time v completed the pausing feature of $\phi_v(t)$ and the time v completed the simulation phase $\phi_v(t)$ itself, the content of $\psi_v(u)$ may change from $M_u(t-1)$ to $M_u(t)$ (if it was not already $M_u(t)$), but it will not store $M_u(t')$ for any $t' > t$. This fact is crucial for the implementation of the simulation feature.

The simulation feature. Upon completion of the pausing feature $P_q \times \{j\}$, v moves on to the simulation feature $S_q \times \{j\}$. The purpose of this feature is to perform the actual simulation of round t in v, namely, to determine the state (of Q) dominating the simulation phase of the next round and the message transmitted when moving from the simulation phase of the current round to that of the next round.

To see how this works out, suppose that $\lambda(q) = \sigma \in \Sigma$. We would have wanted node v to count (up to the bounding parameter b) the number of occurrences of $\widehat{\Sigma}$-letters in its ports that correspond to the transmission of σ at round $t-1$ under Π, that is, the number of occurrences of letters in $\Gamma_{t-1} \cup \Gamma_t$, where

$$\Gamma_{t-1} \;=\; \big\{\big(\sigma', \sigma, j-1\big) \mid \sigma' \in \Sigma \cup \{\varepsilon\}\big\} \qquad \text{and} \qquad \Gamma_t \;=\; \big\{\big(\sigma, \sigma', j\big) \mid \sigma' \in \Sigma \cup \{\varepsilon\}\big\} \;.$$

More formally, the application of the transition function $\widehat{\delta}$ at the end of the simulation phase $\phi_v(t)$ should be based on $f_b(\sum_{\gamma \in \Gamma_{t-1} \cup \Gamma_t} \sharp_\gamma)$, where $\sharp_\gamma$ stands for the number of occurrences of the letter γ in the ports of v at the end of $\phi_v(t)$.

Identifying the integer b with the symbol $b^{\geq}$, we observe that the function $f_b : \mathbb{Z}_{\geq 0} \to B$ satisfies

$$f_b(x+y) \;=\; \min\{f_b(x) + f_b(y), b\}$$

for every $x, y \in \mathbb{Z}_{\geq 0}$. A natural attempt to compute $f_b(\sum_{\gamma \in \Gamma_{t-1} \cup \Gamma_t} \sharp_\gamma)$ would include in the feature $S_q \times \{j\}$ a state $s_{\gamma,i}$ for every letter $\gamma \in \Gamma_{t-1} \cup \Gamma_t$ and integer $i \in \{0, \ldots, b\}$; the query letter of $s_{\gamma,i}$ would be $\widehat{\lambda}(s_\gamma) = \gamma$ and the transition function $\widehat{\delta}$ would be designed so that v moves from $s_{\gamma,i}$ to $s_{\gamma',i'}$, where γ' follows γ in some fixed order of the letters in $\Gamma_{t-1} \cup \Gamma_t$ and $i' = \min\{i + f_b(\sharp_\gamma), b\}$.

However, care must be taken with this approach since $\sharp_\gamma$ may decrease (respectively, increase) during $\phi_v(t)$ for $\gamma \in \Gamma_{t-1}$ (resp., for $\gamma \in \Gamma_t$) due to new incoming messages. To avoid this obstacle, we design the feature $S_q \times \{j\}$ so that first, it computes $\varphi_1 \leftarrow f_b(\sum_{\gamma \in \Gamma_{t-1}} \sharp_\gamma)$; next, it computes $\varphi_2 \leftarrow f_b(\sum_{\gamma \in \Gamma_t} \sharp_\gamma)$; and finally, it computes "again" $\varphi_3 \leftarrow f_b(\sum_{\gamma \in \Gamma_{t-1}} \sharp_\gamma)$. If $\phi_1 = \phi_3$, then the current simulation phase is over and $\widehat{\delta}$ is applied, simulating $\delta(q, f_b(\phi_1 + \phi_2))$; otherwise, the feature $S_q \times \{j\}$ is invoked from scratch. Since the value of $f_b(\sum_{\gamma \in \Gamma_{t-1}} \sharp_\gamma)$ cannot increase during the simulation phase, and since $\phi_1 \leq b$, the feature $S_q \times \{j\}$ is invoked at most b times throughout the execution of the simulation phase. By induction on t, we conclude that our synchronizer satisfies synchronization property (S2), which concludes the correctness proof of the simulation.

Accounting. It remains to show that all ingredients of protocol $\widehat{\Pi}$ are of constant size and that the run-time of protocol $\widehat{\Pi}$ incurs at most a constant multiplicative overhead on top of that of protocol Π. The former claim is established by following our synchronizer construction, observing that $|\widehat{\Sigma}| = O\left(|\Sigma|^2\right)$ and $|\widehat{Q}| = O\left(|Q| \cdot (|\Sigma|^2 + |\Sigma| \cdot b)\right)$ (recall that the bounding parameter b remains unchanged). For the latter claim, we need the following definition: given some node subset $U \in V$ and round $t \in \mathbb{Z}_{>0}$, let $\tau(U,t)$ denote the first time at which u completed simulation phase $\phi_u(t)$ for all nodes $u \in U$. The following proposition can now be established.

Proposition 3.2. *For every node $v \in V$ and round $t \in \mathbb{Z}_{>0}$, the time difference $\tau(\{v\}, t+1) - \tau(N(v) \cup \{v\}, t)$ is (up)bounded by a constant.*

Proof. Since each transmitted message has a delay of at most 1 unit of time, it follows that by time $\tau(N(v) \cup \{v\}, t) + 1$, message $M_u(t)$ must reach $\psi_v(u)$ for all $u \in N(v)$. The pausing and simulation features of $\phi_v(t+1)$ are then completed within $O(|\Sigma|^2)$ and $O(|\Sigma|b)$ steps, respectively. The assertion follows as each step lasts for at most 1 unit of time. □

Employing Proposition 3.2, we conclude by induction on t that $\tau(V,t) = O(t)$ for every $t \in \mathbb{Z}_{>0}$, hence if the execution of protocol Π requires T rounds, then the execution of protocol $\widehat{\Pi}$ is completed within $O(T)$ time units.

3.2 Multiple-letter queries

Recall that according to the model presented in Section 2, each state $q \in Q$ is associated with a query letter $\lambda(q)$ and the application of the transition function when node v resides in state q is determined by $f_b(\sharp_\sigma)$, where $\sharp_\sigma$ is the number of occurrences of the letter σ in the ports of v. From the perspective of the protocol designer, it is often more convenient to assume that the node queries on multiple letters — in fact, all letters — simultaneously, namely, that the application of the transition function is determined by the vector $(f_b(\sharp_\sigma))_{\sigma \in \Sigma}$.

Now that we may assume a synchronous environment, this stronger multiple-letter queries

assumption can easily be supported. Indeed, at the cost of increasing the number of states and the run-time by constant factors, one can subdivide each round into $|\Sigma|$ subrounds, dedicating each subround to a different letter in Σ, so that at the end of the round, the state of the v indicates $f_b(\sharp_\sigma)$ for every $\sigma \in \Sigma$.

References

[1] Y. Afek, N. Alon, Z. Bar-Joseph, A. Cornejo, B. Haeupler, and F. Kuhn. Beeping a maximal independent set. In *Proceedings of the 25th international conference on Distributed computing*, DISC'11, pages 32–50, Berlin, Heidelberg, 2011. Springer-Verlag.

[2] Y. Afek, N. Alon, O. Barad, E. Hornstein, N. Barkai, and Z. Bar-Joseph. A Biological Solution to a Fundamental Distributed Computing Problem. *Science*, 331(6014):183–185, Jan. 2011.

[3] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414(6862):430–434, Nov. 2001.

[4] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30:323–342, April 1983.

[5] I. Chlamtac and S. Kutten. On Broadcasting in Radio Networks–Problem Analysis and Protocol Design. *Communications, IEEE Transactions on [legacy, pre - 1988]*, 33(12):1240–1246, 1985.

[6] A. Cornejo and F. Kuhn. Deploying wireless networks with beeps. In *Proceedings of the 24th international conference on Distributed computing*, DISC'10, pages 148–162, Berlin, Heidelberg, 2010. Springer-Verlag.

[7] R. Flury and R. Wattenhofer. Slotted Programming for Sensor Networks. In *International Conference on Information Processing in Sensor Networks (IPSN), Stockholm, Sweden*, April 2010.

[8] M. Gardner. Mathematical games the fantastic combinations of john conway s new solitaire game life . *Scientific American*, 223(4):120–123, 1970.

[9] P. Gordon. Numerical Cognition Without Words: Evidence from Amazonia. *Science*, 306(5695):496–499, Oct. 2004.

[10] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21:193–201, Feb. 1992.

[11] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1st edition, 1996.

[12] J. V. Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[13] D. Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[14] S. Wolfram. *A new kind of science*. Wolfram Media, Champaign, Ill, 2002.