On the Fairness of Probabilistic Schedulers for Population Protocols

Ioannis Chatzigiannakis[1,2], Shlomi Dolev[3], Sándor P. Fekete[4], Othon Michail[1,2] and Paul G. Spirakis[1,2]

[1] Research Academic Computer Technology Institute (CTI), Patras, Greece
{ichatz, michailo, spirakis}@cti.gr
[2] Computer Engineering and Informatics Department, University of Patras, Greece
[3] Department of Computer Science, Ben-Gurion University of the Negev, Israel 84105
dolev@cs.bgu.ac.il
[4] Department of Computer Science, Braunschweig University of Technology, Braunschweig, Germany
s.fekete@tu-bs.de

Abstract. We propose a novel, generic definition of *probabilistic schedulers* for population protocols. We design two new schedulers, the *State Scheduler* and the *Transition Function Scheduler*. Both possess the significant capability of being *protocol-aware*, i.e. they can assign transition probabilities based on information concerning the underlying protocol. We prove that the proposed schedulers, and also the *Random Scheduler* that was defined by Angluin et al. [1], are all fair with probability 1. We also define and study *equivalence* between schedulers w.r.t. *performance* and *correctness* and prove that there exist fair probabilistic schedulers that are not equivalent w.r.t. to performance and others that are not equivalent w.r.t. correctness. We implement our schedulers using a new tool for simulating population protocols and evaluate their performance from the viewpoint of experimental analysis and verification. We study three representative protocols to verify stability, and compare the experimental time to convergence with the known complexity bounds. We run our experiments from very small to extremely large populations (of up to 10^8 agents). We get very promising results both of theoretical and practical interest.

Keywords. Population Protocols, Fairness, Probabilistic Schedulers, Communicating Automata, Sensor Networks, Experimental Evaluation

1 Introduction

Recently, Angluin et al. [1,2] introduced the notion of a computation by a population protocol to model distributed systems in which individual agents are extremely limited and can be represented as finite-state machines. In their model, complex behavior of the system as a whole emerges from the rules governing pairwise interaction of the agents. The computation is carried out by a collection of agents, each of which receives a piece of the input. These agents move around and information can be exchanged between two agents whenever they come into contact with (or sufficiently close to) each other. The goal is to ensure that every agent can eventually output the value that is to be computed.

An execution of a protocol proceeds from the initial configuration by interactions between pairs of agents. In a real distributed execution, interactions could take place simultaneously, but when writing down an execution simultaneous interactions can be ordered arbitrarily. Angluin et al. think of the order in which pairs of agents come into contact and interact as being chosen by an adversary. From a particular system state, the adversary decides which of the possible different interactions will be selected; essentially it decides the computation sequence (i.e. schedule of interactions). So, the designer's goals is to make protocols work correctly under any schedule the adversary may choose.

In such models there may exist diverging (infinite) schedules of interactions such that during their execution some *event becomes possible infinitely often* but it has not an infinite number of occurences. If the adversary selects such a sequence, it will lead the system to unfair situations, where although an event is realizable infinitely often, it never occurs because conflicts are resolved in a non equitable manner. To deal with these issues, a **fairness** restriction is imposed on the adversarial scheduler: the scheduler is not allowed to avoid a possible step forever. The fairness constraint allows the scheduler to behave arbitrarily for an arbitrarily long period of time, but does require that it behave nicely eventually. Therefore correctness is a property that can be satisfied eventually.

Fairness relative to a set of state is important since most of the "interesting" system properties express reachability relations of some set of states [3]. In other words, fairness becomes crucial when a property is to be proven in formal systems based on non-deterministic models. In this work we try to apprehend the concept of fairness in the basic population protocols model. To do so, we focus on the class of *probabilistic schedulers* proposed in [1,2], in which the scheduler selects randomly the next pair to interact. We define two new adversaries that are bound by the fairness constraint of [1]. The "reasonable" scheduling policies that they introduce lead to significantly different performance characterizations for some protocols well studied in the relevant literature. We show that the current notion of fairness gives rise to many difficulaties in studying not only to performance but also protocol correctness.

Although in the area initiated by the proposal of the population protocol model there exists already a large amount of work, this work has almost excluded experimental evaluation through simulations, testbed development and testing in real sensor populations. Mainly, the foundational approach has been followed, but as the gap between theoretical results and their practical verification grows we believe that now is the right time for approaches that deal with the latter. Moreover, any progress in the development of tools specialized on experimental evaluation of protocols could provide a powerful alternative option in cases where analysis fails or turns out to be extremely hard. To expermental study the fairness property of our schedulers, we've developed a specialized simulation tool, dedicated for the simulation of population protocols. Following [4], we replace low-level effects with abstract and exchangeable models. This fact, together with the use of data structures that support efficient lookup and updates in constant time, allows us to simulate for the first time extremely large populations (i.e. of up to 10^8 agents) in a reasonable time.

We choose three characteristic protocols appearing in the relevant literature, for which known bounds exist in the case of the random scheduler, that are of great significance in a wide range of applications; these are the *Or* protocol, the *Leader Election* protocol, and the *Majority* protocol. These protocols are then tested intensively on a wide range of possible real scenarios. We compare their performance based mainly on three metrics that we define, namely, the *Protocol Stability*, the *Approximate Stability*, and the *Population Dynamics*. Our results indicate that no scheduler can be thought of as being universally worst/best in terms of performance. For the three protocols under consideration, our simulations validate the existing time complexity bounds in the case of the random scheduler and yield new time complexity results for the proposed schedulers (taking into account a possible amount of statistical error).

The *Population Protocol* (PP) model was inspired in part by work by Diamadi and Fischer [5] on trust propagation in a social network. The motivation given for the model was the study of sensor networks in which passive agents were carried along by other entities. Much work has been devoted to the, now, well-known fact that the set of computable predicates of the basic (complete interaction graph) population protocol model and most of its variants is exactly equal or closely related to the set of *semilinear predicates*. Moreover, in [1,2], the *Probabilistic Population Protocol* model was proposed, in which the scheduler selects randomly and uniformly the next pair to interact. More recent work has concentrated on performance, supported by this random scheduling assumption. Additionally, several extensions of the basic model have been proposed in order to more accurately reflect the requirements of practical systems. In [6], Angluin et al. studied what properties of restricted communication graphs are stably computable, gave protocols for some of them, and proposed a model extension with *stabilizing inputs*. In [7] the *Mediated Population Protocol* (MPP) model was proposed that extends the population protocol model with communication links that are able to store states. The MPP model was proved to be computationally stronger than the PP model and it was observed that it is especially capable of deciding graph properties, concerning the communication graph on which the protocol runs. In [8] the decidable graph properties by MPPs where studied for the first time and it was proven that connectivity cannot be decided by the new model. Unfortunatelly, the class of decidable graph languages by MPPs remains open. Finally, some works incorporated agent failures and gave to the agents slightly increased memory capacity. For an excellent introduction to the subject of population protocols see [9] and for some recent advances mainly concerning mediated population protocols see [10].

2 Population Protocols

A *population protocol* (PP) is a 6-tuple (X, Y, Q, I, O, δ), where X, Y, and Q are all finite sets, and X is the *input alphabet*, Y is the *output alphabet*, Q is the set of *states*, $I : X \to Q$ is the *input function*, $O : Q \to Y$ is the *output function*, and $\delta : Q \times Q \to Q \times Q$ is the *transition function*. If $\delta(a, b) = (a', b')$, we call $(a, b) \to (a', b')$ a *transition* and we define $\delta_1(a, b) = a'$ and $\delta_2(a, b) = b'$.

A population protocol $\mathcal{A} = (X, Y, Q, I, O, \delta)$ runs on a *communication graph* (also known as *interaction graph*) $G = (V, E)$ (G is here assumed to be directed and without multiple edges or self-loops). From now on we will use the letter n to denote the cardinality of V (size of the population). Initially, all agents (i.e. the elements of V) receive a global start signal, sense their environment and each one receives an input symbol from X. After receiving their input symbol, all agents apply the input function to it and go to their initial state (e.g. all agents that received $\sigma \in X$ begin with initial state $I(\sigma) \in Q$). An adversary scheduler selects in each step a directed pair of agents $(u, \upsilon) \in E$, where $u, \upsilon \in V$ and $u \neq \upsilon$, to interact. Assume that the scheduler selects the pair (u, υ), that the current states of u and υ are $a, b \in Q$, respectively, and that $\delta(a, b) = (a', b')$. Agent u plays the role of the *initiator* in the interaction (u, υ) and υ that of the *responder*. When interacting, u and υ apply the transition function to their directed pair of states, and, as a result, u goes to a' and υ to b' (both update their states according to δ, and specifically, the initiator applies δ_1 while the responder δ_2).

A *configuration* is a snapshot of the population states. Formally, a configuration is a mapping $C : V \to Q$ specifying the state of each agent in the population. C_0 is the initial configuration (for simplicity we assume that all agents apply the input function at the same time) and, for all $u \in V$, $C_0(u) = I(x(u))$, where $x(u)$ is the input symbol sensed by agent u. Let C and C' be configurations, and let u, υ be distinct agents. We say that C goes to C' via encounter $e = (u, \upsilon)$, denoted $C \xrightarrow{e} C'$, if $C'(u) = \delta_1(C(u), C(\upsilon))$, $C'(\upsilon) = \delta_2(C(u), C(\upsilon))$, and $C'(w) = C(w)$ for all $w \in V - \{u, \upsilon\}$, that is, C' is the result of the interaction of the pair (u, υ) under configuration C and is the same as C except for the fact that the states of u, υ have been updated according to δ_1 and δ_2, respectively. We say that C can go to C' in one step, denoted $C \to C'$, if $C \xrightarrow{e} C'$ for some encounter $e \in E$. We write $C \xrightarrow{*} C'$ if there is a sequence of configurations $C = C_0, C_1, \ldots, C_t = C'$, such that $C_i \to C_{i+1}$ for all i, $0 \leq i < t$, in which case we say that C' is *reachable* from C.

3 Schedulers

3.1 Fair Probabilistic Schedulers

As defined in [2], the *transition graph* $T(\mathcal{A}, G)$ of a protocol $\mathcal{A}$ running on a communication graph G (or just T when no confusion arises about the protocol and the communication graph) is a directed graph whose nodes are all possible configurations and whose edges are all possible one-step transitions between those configurations.

Definition 1. *A* probabilistic scheduler, *w.r.t. a transition graph $T(\mathcal{A}, G)$, defines for each configuration $C \in V(T)$ an infinite sequence of probability distributions of the form $(d_1^C, d_2^C, \ldots)$, over the set $\Gamma^+(C) = \{C' \mid C \to C''\}$ (the possibly closed out-neighbourhood of C), where $d_t^C : \Gamma^+(C) \to [0,1]$ and such that $\sum_{C' \in \Gamma^+(C)} d_t^C(C') = 1$ holds, for all t and C.*

The initial configuration C_0 depends only on the values sensed by the population and, in particular, it is formed by their images under the input function. So, for the time being, we can assume that C_0 is selected in a deterministic manner. Let C_t denote the configuration selected by the scheduler at step t (the configuration of the system after t selections of the scheduler and applications of the transition function). Assume that it is the lth time that C_t is encountered during the execution so far; then a probabilistic scheduler selects C_{t+1} randomly, according to the distribution $d_l^{C_t}$. In other words, d_l^C denotes the probability distribution over $\Gamma^+(C)$ when C is encountered for the lth time.

Definition 2. *We call a probabilistic scheduler* consistent, *w.r.t. a transition graph $T(\mathcal{A}, G)$, if for all configurations $C \in V(T)$, it holds that $d^C = d_1^C = d_2^C = \ldots$, which, in words, means that any time the scheduler encounters configuration C it chooses the next configuration with the same probability distribution d^C over $\Gamma^+(C)$, and this holds for all C (each with its own distribution).*

From now on, and when no confusion arises, we shall use the letters i and j not only to denote configuration indices but also to denote configurations themselves. Note that a consistent probabilistic scheduler for $T(\mathcal{A}, G)$ is simply a labeling $P : E(T) \to [0,1]$ on the arcs of T, such that for any $i \in V(T)$, $\sum_{j \in \Gamma^+(i)} P(i,j) = 1$. So, any time a consistent scheduler encounters a configuration i, it selects the next configuration j according to the probability distribution defined by the labels of the arcs leaving from i. Note that in the latter case, if we remove from T all $e \in E(T)$ where $P(e) = 0$ then the resulting graph D is the underlying graph of a finite Markov chain where the state space is $\mathcal{C} = Q^V$ (all possible configurations) and for all $i, j \in \mathcal{C}$, if $i \to j$ then $\mathbb{P}_{ij} = P(i,j)$, i.e. equal to the label of arc (i,j), otherwise $\mathbb{P}_{ij} = 0$.

A strongly connected component of a directed graph is *final* iff no arc leads from a node in the component to a node outside. A *configuration* is final iff it belongs to a final strongly connected component of the transition graph.

An *execution* is a finite or infinite sequence of configurations $C_0, C_1, C_2, \ldots$, where C_0 is an initial configuration and $C_i \to C_{i+1}$, for all $i \geq 0$. An infinite execution is *fair* if for every possible transition $C \to C'$, if C occurs infinitely often in the execution then C' also occurs infinitely often. A *computation* is an infinite fair execution.

Let y_C, where $y_C(u) = O(C(u))$ for all $u \in V$, denote the output (assignment) of configuration C. We say that a computation of a population protocol $\mathcal{A}$ *stabilizes* to output y_C if it contains a configuration C such that for all C' reachable from C we have that $y_{C'} = y_C$.

Theorem 1. *Let $\Xi = C_0, C_1, \ldots$ be an infinite execution of $\mathcal{A}$ on G, $\mathcal{F}_{\Xi}$ be the set of configurations that occur infinitely often in Ξ, and $T_{\mathcal{F}_{\Xi}}$ be the subgraph of $T(\mathcal{A}, G)$ induced by $\mathcal{F}_{\Xi}$. Ξ is a computation (i.e. it is additionally fair) iff $T_{\mathcal{F}_{\Xi}}$ is a final strongly connected component of $T(\mathcal{A}, G)$.*

Proof. The "only if" part was proven in [2]. We prove here the "if" part. Assume that $T_{\mathcal{F}_{\Xi}}$ is a final strongly connected component of $T(\mathcal{A}, G)$ and that Ξ is not fair (i.e. that the statement of the "if" part does not hold). Then there exists some configuration $C \in \mathcal{F}_{\Xi}$ (i.e. appearing infinitely often) and a $C' \notin \mathcal{F}_{\Xi}$ such that $C \rightarrow C'$. But this contradicts the fact that $T_{\mathcal{F}_{\Xi}}$ is final. $\square$

We now keep the preceding definitions of Ξ, $T(\mathcal{A}, G)$, $\mathcal{F}_{\Xi}$, $T_{\mathcal{F}_{\Xi}}$, but additionally assume a consistent scheduler.

Theorem 2. *If for all $i \in \mathcal{F}_{\Xi}$ and all configurations j s.t. $i \rightarrow j$ it holds that $\mathbb{P}_{ij} > 0$, then Ξ is a computation with probability 1.*

Proof. Because i is persistent and all its successor configurations j may occur in one step from i with non-zero probability it follows that those j are also persistent with probability 1, i.e. they also occur infinitely often in Ξ, thus Ξ is a computation with probability 1, by definition. $\square$

Definition 3. *A scheduler S is fair if for any protocol $\mathcal{A}$, any communication graph G, and any infinite execution Ξ of $\mathcal{A}$ on G caused by S, Ξ is also a computation (i.e. additionally fair).*

Intuitively a scheduler is fair if it always leads to computations.

Theorem 3. *Any consistent scheduler, for which it holds that $\mathbb{P}_{ij} > 0$, for any protocol $\mathcal{A}$, any communication graph G, and all configurations $i, j \in V(T(\mathcal{A}, G))$ where $i \rightarrow j$ and $i \neq j$, is fair with probability 1.*

Proof. First of all, note that the underlying Markov chain graph of such a scheduler is the transition graph without possibly some self-loops. Assume that the statement does not hold. Then the probability that a specific infinite execution Ξ of some protocol $\mathcal{A}$ on some graph G caused by the scheduler is not a computation is non-zero. This means that a Ξ may occur, for which there exists some configuration $i \in \mathcal{F}_{\Xi}$ (appearing infinitely often in Ξ) and $j \notin \mathcal{F}_{\Xi}$ such that $i \rightarrow j$. Now there are two cases:

1. $i = j$. In this case the contradiction is trivial, because it follows that $i \in \mathcal{F}_{\Xi}$ while at the same time $i \notin \mathcal{F}_{\Xi}$.
2. $i \neq j$. However, by assumption $\mathbb{P}_{ij} > 0$, and because i is persistent j must also be with probability 1.

$\square$

3.2 Proposed Schedulers

In [1] a probabilistic scheduler that selects the next ordered pair to interact at random, independently and uniformly from all ordered pairs corresponding to arcs of the communication graph (i.e. elements of E) was defined. Here we call this scheduler Random Scheduler, and define two new probabilistic schedulers, namely the State Scheduler and the Transition Function Scheduler.

The **Random Scheduler**. To generate C_{i+1} the Random Scheduler selects an ordered pair $(u, v) \in E$ at random, independently and uniformly (each with probability $1/|E|$), and applies the transition function to $(C_i(u), C_i(v))$.

The **State Scheduler**. Consider a population protocol for k-mutual exclusion, in which only k agents are in state 1 and the rest of the population is in state 0. When an agent that holds a token interacts with another agent, it passes the token. Now consider an execution where $n \gg k$ and we use the Random Scheduler. In the case in which the communication graph is complete, the probability of selecting a pair with states $(1, 0)$ is much smaller than selecting a pair with states $(0, 0)$. Essentially, this means that the scheduler may initiate a large number of interactions that do not help the protocol in making progress. This observation is the motivation for the *State Scheduler*. Instead of selecting a pair of processes independently and uniformly, the scheduler selects a pair based on the states of the processes. It first selects a pair of *states* and in the sequel it selects one process from each state. Thus it allows the "meaningful" transitions to be selected more often and may avoid selecting a large number of interactions that delay the protocol's progress.

More formally, an ordered pair of states (q, q') is said to be an *interaction candidate* under configuration C if $\exists(u, v) \in E$ such that $C(u) = q$ and $C(v) = q'$. Then a configuration C_{i+1} is generated from C_i as follows: (i) by drawing an order pair (q, q') of states at random, independently and uniformly from all ordered pairs of states that are interaction candidates under C_i, (ii) drawing an ordered pair (u, v) such that $C_i(u) = q$ and $C_i(v) = q'$ from all such pairs at random, independently and uniformly, (iii) applying the transition function δ to $(C_i(u), C_i(v))$ and updating the states of u and v accordingly to obtain C_{i+1}.

The **Transition Function Scheduler**. Continuing the same argument, we define one more scheduler that assumes knowledge of the protocol executed. It examines the transition function δ and selects pairs of agents based on the defined transitions. In the case in which function δ defines transitions that do not change the state, neither of the initiator nor of the responder agent (e.g., $(\alpha, \beta) \to (\alpha, \beta)$), these transitions are ignored by the scheduler. This scheduler guarantees that all interactions will lead to a state change of either the initiator or the responder or both.

More formally, suppose $\to$ is a binary relation over Q^2 which is the relation analogue of the corresponding transition function δ. The reflexive reduction of $\to$, denoted by $\dot{\to}$, is simply $\to$ without members related to themselves by $\to$. A configuration C_{i+1} is generated from C_i as follows: (i) by drawing a pair $((q_1, q_2), (q'_1, q'_2))$ at random, independently and uniformly from all such pairs belonging to $\dot{\to}$ for which (q_1, q_2) is an interaction candidate under C_i, (ii) drawing an ordered pair (u, v) such that $C_i(u) = q_1$ and $C_i(v) = q_2$ from all such pairs at random, independently and uniformly, (iii) applying the transition function δ to $(C_i(u), C_i(v))$ and updating the states of u and v accordingly to obtain C_{i+1} (if in step (i) there exists no such interaction candidate, then the Transition Function Scheduler becomes a Random Scheduler, and remains in the same configuration for an infinite number of steps).

Given the above schedulers we can classify any scheduler for population protocols based on whether it assumes any knowledge on the actual protocol executed or not.

Definition 4. *We call a scheduler* protocol-oblivious *(or* agnostic*) if it constructs the interaction pattern without any knowledge on the protocol executed and* protocol-aware *if it takes into account information concerning the underlying protocol.*

Based on this classification, the Random Scheduler is a protocol-oblivious scheduler while the State and Transition Function Schedulers are protocol-aware.

Theorem 4. *The Random Scheduler, State Scheduler, and Transition Function Scheduler are all fair with probability 1.*

Proof. Let $T(\mathcal{A}, G)$ be any transition graph.

– *Random Scheduler.* Let i be any configuration in $V(T)$. Any time i is encountered, any j for which $i \rightarrow j$ is selected with probability $\mathbb{P}_{ij} = |K_{ij}|/|E|$, where $K_{ij} = \{e \mid e \in E(G) \text{ and } i \xrightarrow{e} j\}$, which is independent of the number of times i has been encountered. Thus the Random Scheduler is consistent. Moreover, $|K_{ij}| > 0$, because from definition of $i \rightarrow j$ we have that $\exists e \in E$ (where E is used instead of $E(G)$) such that $i \xrightarrow{e} j$. Thus $\mathbb{P}_{ij} > 0$ and Theorem 3 applies implying that the Random Scheduler is fair with probability 1.
– *State Scheduler.* Let i, j be distinct configurations in $V(T)$ such that $i \rightarrow j$. When the State Scheduler has chosen i to select the next configuration of the execution, it performs two experiments. First it selects a pair of states (q, q') from all interaction candidates. Then it selects an arc e from all $(u, v) \in E$ such that $i(u) = q$ and $i(v) = q'$. Let K_{ij} again denote the set of arcs (i.e. interactions) that convert i to j. Let also $M_{ij} = \{(q, q') \mid \exists (u, v) \in K_{ij}$ such that $i(u) = q$ and $i(v) = q'\}$ and IC_i denote the set of all interaction candidates under i (note that $M_{ij} \subseteq IC_i$). Now $1/|IC_i|$ is the probability that a specific interaction candidate is selected by the scheduler. Let $K_{ij}^{(q,q')} = \{(u, v) \mid (u, v) \in K_{ij}$ and $i(u) = q, i(v) = q'\}$ (the subset of K_{ij} containing all arcs (u, v) that convert i to j and where the state of u is q and the state of v is q') and $E_i^{(q,q')} = \{(u, v) \mid (u, v) \in E$ and $i(u) = q, i(v) = q'\}$. Now given a chosen interaction candidate $(q, q') \in M_{ij}$ the probability that j is selected is equal to $|K_{ij}^{(q,q')}|/|E_i^{(q,q')}|$. Thus we have

$$\mathbb{P}_{ij} = \sum_{(q,q') \in M_{ij}} \frac{|K_{ij}^{(q,q')}|}{|IC_i||E_i^{(q,q')}|}.$$

$|K_{ij}^{(q,q')}|$, $|IC_i|$ and $|E_i^{(q,q')}|$ for all $(q,q') \in M_{ij}$ only depend on the specific configurations i and j and are always the same w.r.t. different times at which i is encountered by the scheduler. Thus the State Scheduler is consistent. Moreover, since $(i \to j) \Rightarrow \exists e = (u,v) \in E$ such that $i \xrightarrow{e} j$. Let q and q' be the states of u and v under i, respectively. It follows that $(q,q') \in M_{ij}$ and that $|M_{ij}| > 0$. Finally, note that $e \in K_{ij}^{(q,q')}$, because $e \in K_{ij}$, $i(u) = q$, and $i(v) = q'$. Thus $\mathbb{P}_{ij} > 0$, Theorem 3 applies and as a consequence the State Scheduler is fair with probability 1.

– *Transition Function Scheduler*. In the case in which $i \neq j$, $\mathbb{P}_{ij}$ is defined as in the State Scheduler, by simply replacing the phrase "interaction candidate" with "interaction candidate that constitutes the lhs of some rule in the reflexive reduction of δ". So also this scheduler is consistent and fair with probability 1. Note that when $i = j$ and i has at least one out-neighbor in T different from i, then $\mathbb{P}_{ij} = 0$, since this scheduler does not select transitions that leave the states of the participating agents unaffected. Moreover, if i has a unique out-going arc (in T) pointing to itself, then the scheduler selects i for an infinite number of steps with probability 1 (in this case becomes a Random Scheduler). In both cases no problem arises, because for Theorem 3 to apply we only require $\mathbb{P}_{ij} > 0$ for all $i \neq j$ such that $i \to j$.

□

4 Large Scale Simulation

A plethora of network simulation environments and tools with widely varying scopes have been proposed to the relevant research community, each with different application domains, focusing on different aspects of the system. Most incorporate details on the operation of the lower stacks of the networks to provide a more realistic simulation environment (e.g., *ns-2* [11], *OPNET Modeler* [12]). Others focus on particular network types in order to improve accuracy, performance, or scalability, most notable examples are *Shawn* [4], *AlgoSenSim* [13], *ADAPT* [14] and *WSNGE* [15].

The most central issue in simulating population protocols is the way in which agents interact; the scheduler of the population must be at the core of the simulator. The vast majority of existing simulation tools use a discrete-event processor as their engine that does not allow to directly dictate the way the events are processed. The only way to affect the way agents interact is to do it indirectly via controlling either the mobility of the nodes, or their power schedule or through a failure model. This is a very limiting factor for simulating population protocols. For this reason we decided to develop a specialized tool, dedicated for the simulation of population protocols.

We totally avoid the complete simulation of the physical environment and the lower-level networking protocols. We follow the approach of [4] and replace low-level effects with abstract and exchangeable models that allows us to simulate extremely large networks in reasonable time. In a 32-bit linux with 4GB of memory we can simulate populations of up to $n = 10^8$ while in a 64-bit system this can increase to even higher numbers. Our simulation tool (which we call **ppsim**) is available for download at `http://ru1.cti.gr/projects/ppsim`.

We developed the tool using generic programming in JAVA 1.6 where all the components are parameterized on the set of states Q of the population protocol, implemented by the component *State*. Figure 1 (see Appendix) shows the main components of the simulation tool; dashed borders represent *abstract* classes that are extended to implement specific protocols, schedulers and experiments. The *Agent* component represents a single node and only keeps track of the state of the agent, i.e., a local variable of the parameterized type *State*. The *Protocol* component only defines the transition function T as the corresponding transition relation $T \subseteq Q^4$ by a four column matrix where each row $(\alpha, \beta, \alpha', \beta')$ represents a transition $(\alpha, \beta) \rightarrow (\alpha', \beta')$. Thus protocols are implemented in a very easy and quick way.

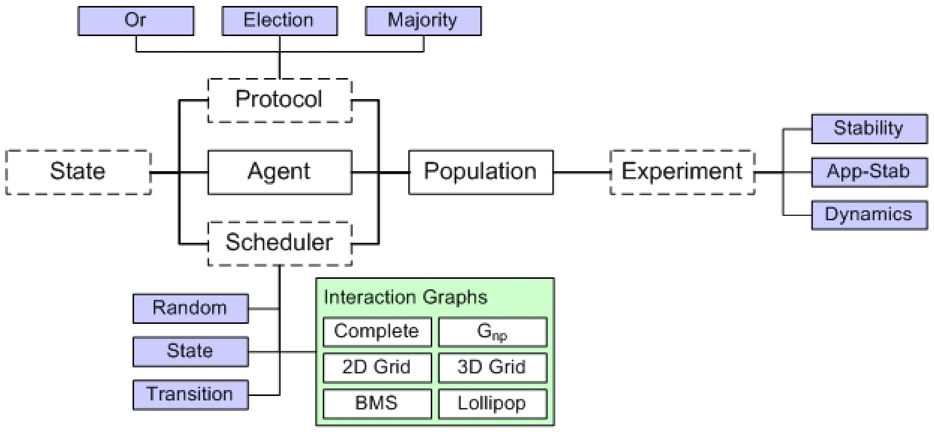


Fig. 1. Main components of the simulator and their interconnection.

The *Population* component holds all the *Agents* in a hash table (or `HashMap` in JAVA) and uses two additional data structures for monitoring the dynamics of the population and for grouping the agents based on their state.

The *Scheduler* component implements a specific policy of interactions. All the schedulers defined in Sec. 3.2 subclass this component. The restricted interactions schedulers are also responsible for storing the graph structure using a hash table that maps to each agent a set of neighbors.

The last component of our tool is the *Experiment* that defines the performance metrics, duration and operation parameters of the simulator in a parameterized way. We define a set of experiments for evaluating the performance of protocols based on the following metrics:

Protocol Stability. The most important goal of simulation is to verify the correctness of a protocol. The experiment lets the population interact and monitors the number of interactions required until the execution converges to a *stable state*.

Approximate Stability. In large sized populations it is common for a protocol to require a very large number of interactions until the execution converges to a stable state as in the final stages it requires a specific interactions to occur between very few agents. Therefore we wish to examine the number of interactions required until *a fraction of the population reaches a stable state*.

Population Dynamics. In order to understand the behavior of a protocol under different schedulers and interaction graphs we wish to monitor the dynamics of the population as the protocol execution progresses. After each interaction we count the number of agents that are at state $q \in Q$. Let n_q be the number of agents at a given state q, then $n = \sum_{i=1}^{|Q|} n_i$.

5 Logical OR Protocol (Propagation of an epidemic)

We start by considering a simple protocol based on an example of [9] where each agent with input 0 simply outputs 1 as soon as it discovers that another agent had input 1. Formally, we have $Q = X = Y = \{0, 1\}$ and the transitions defined by function T are the following:

$$
\begin{array}{ll}
(0,0) \to (0,0) & \\
(0,1) \to (1,1) & (1,0) \to (1,1) \\
(1,1) \to (1,1) &
\end{array}
$$

Essentially, if all agents have input 0, no agent will ever be in state 1. If some agent has input 1, given a fair scheduler, we expect that the number of agents with state 1 will increase and will eventually reach n. In both cases, all agents stabilize to the correct output value, though some important fundamental questions are "how fast?" and "how does the scheduler and the interaction graph affect the number of interactions required to stabilize?".

We start our experimentation by evaluating the *protocol stability*. We set the input bit of only one agent to 1 and set $n = \{10^1, \ldots, 10^7\}$. We count the number of interactions until all agents output 1. The experimental results have been used to compare the average number of interactions required when using different schedulers and interaction graphs. For the case of a complete interaction graph, based on Figure 2, we observe that the *Random scheduler* seems to require $\mathcal{O}(n \log n)$ interactions. The experimental results with the random scheduler verify the theoretical results of [16] that characterize the behavior of the Or protocol in complete graphs as a **one-way epidemic**. They show that the number of interactions for the epidemic to finish is $\Theta(n \log n)$ with high probability by using arguments of the well known coupon collector problem, in which balls are thrown uniformly at random into bins until every bin contains at least one ball. Interestingly, the *State scheduler* and the *Transition Function scheduler* seem to require only $\mathcal{O}(n)$ interactions.

In the second set of experiments we evaluate the population dynamics. Figure 3 depicts the number of agents that have changed to state 1 as the protocol

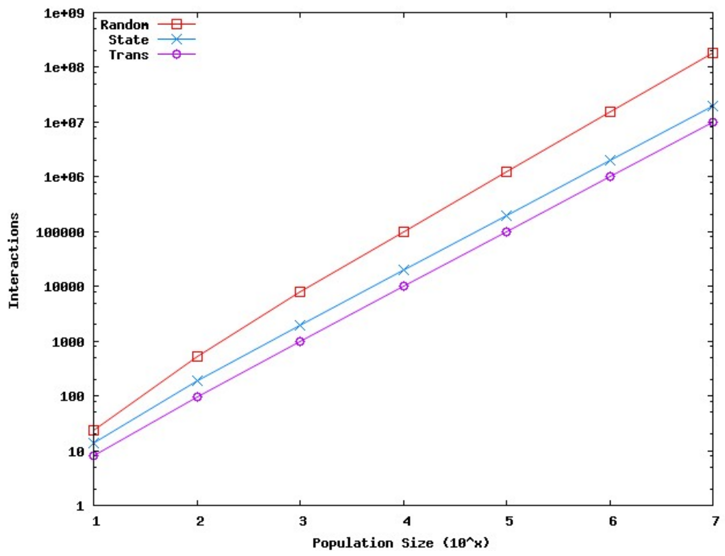


Fig. 2. Total number of interactions for Logical OR protocol to reach a stable state under different schedulers.

evolves. Interestingly, for the case of the Random scheduler, initially the rate of increase of n_1 is small, then as a critical mass of agents is infected, n_1 increases very fast until it slows down again when it comes to infecting the last remaining agents. This experiment shows clearly that although the Or protocol is correct since the execution reaches a stable state, it requires a very large number of interactions to cover all the population.

6 Leader Election Protocol

We now study the fundamental problem of leader election in such constrained yet extremely large scale systems. We use the deterministic leader election protocol stated in [6] where an agent at state 1 is a *leader* and at state 0 it is a *non-leader*. Initially all agents start with a status leader and the protocol stabilizes when only one agent is a leader and all other agents have a status follower. Formally, $Q = X = Y = \{0, 1\}$ and transition function T is defined as follows:

$$\begin{array}{ll} (0,0) \to (0,0) & \\ (1,0) \to (0,1) & (0,1) \to (1,0) \\ (1,1) \to (0,1) & \end{array}$$

Based on the above transitions and a fair scheduler, this protocol starting with the initial configuration in any interaction graph, it will eventually reach a configuration where only one agent will be in state 1 (the leader) and every other agent will be in state 0. When this configuration is reached we say that

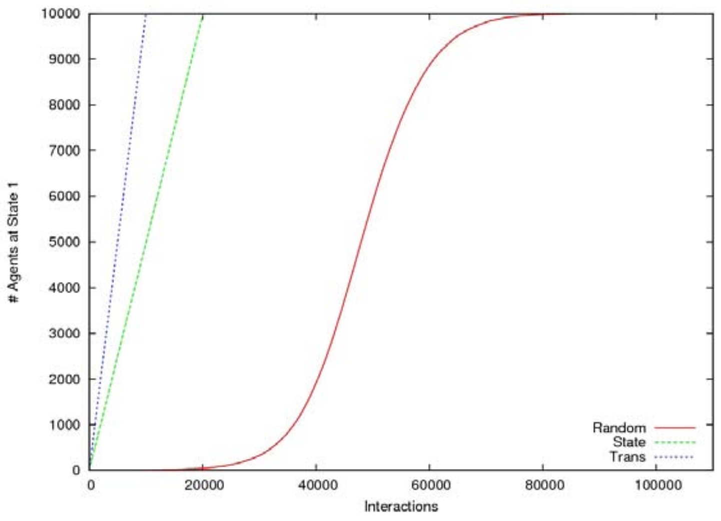


Fig. 3. Number of agents at state 1 per protocol interactions under different schedulers when $n = 10000$.

the protocol has stabilized. Interestingly, in this protocol, the agent elected as the leader does not remain fixed. Due to the transitions (2) and (3) the leader continuously changes.

We start by evaluating the protocol stability and the effect of the scheduler and the interaction graph on the number of interactions required for $n = \{10^1, \ldots, 10^7\}$. Figure 4 shows the results for the case of the unrestricted schedulers. We note that these are the first results (either via rigorous mathematical analysis or via experimentation) on the performance of this very basic protocol. By taking into account a possible amount of statistical error, our results indicate that the agnostic scheduler requires $\mathcal{O}(n^2)$ interactions while the protocol aware schedulers require $\mathcal{O}(n)$ interactions.

In the second set of experiments we evaluate the population dynamics. The results are shown in Figure 5. As with the Or protocol, the heavy tail indicates that stabilization requires a very large number of interactions (regardless of the interaction graph) except for if we apply a protocol aware scheduler.

7 Majority Protocol (Propagation of conflicting epidemics)

We now consider a very broad application where nodes are carried by individuals that wish to vote. There are 2 candidates (X or Y) and individuals are not forced to vote. The nodes play the role of the ballot; by pressing the corresponding button an input value is generated for the software agents. We wish to execute a protocol to count the ballots and outputs the winner. We here use

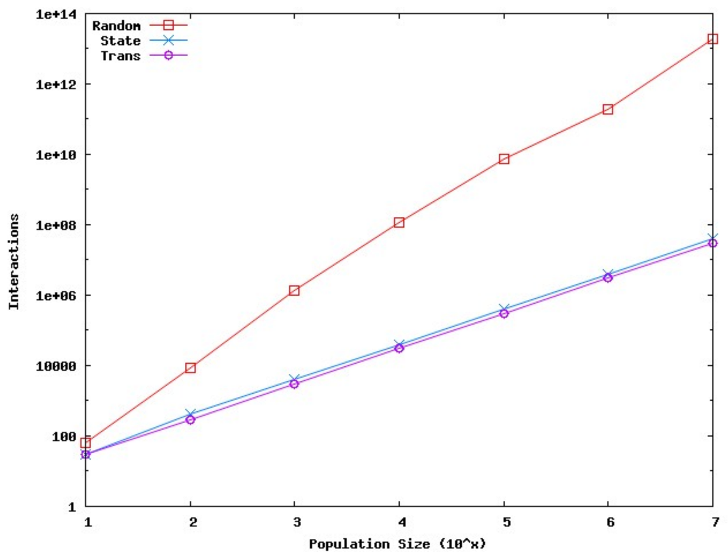


Fig. 4. Total number of interactions for the Leader Election protocol to stabilize for different schedulers.

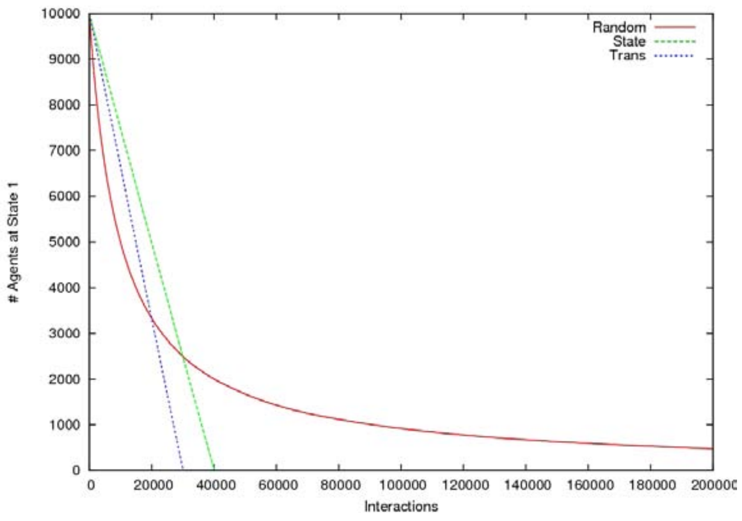


Fig. 5. Number of agents at state 1 per protocol interactions under different schedulers when $n = 10000$.

a simple protocol based on [17] where each agent has three states: X, Y or B (for undecided). Formally, we have $Q = X = Y = \{X, Y, B\}$ and the transition function T is defined as follows:

$(X, X) \rightarrow (X, X)$
$(Y, Y) \rightarrow (Y, Y)$
$(B, B) \rightarrow (B, B)$
$(X, B) \rightarrow (X, X) \quad (B, X) \rightarrow (X, X)$
$(Y, B) \rightarrow (Y, Y) \quad (B, Y) \rightarrow (Y, Y)$
$(X, Y) \rightarrow (X, B)$
$(Y, X) \rightarrow (Y, B)$

One can view the above protocol as an extention of the Or protocol of Sec. 8.1 to three states. If we remove all transitions related to state Y we get the simple Or protocol. For this we view this protocol as the propagation of **conflicting epidemics**; the epidemic of X and the epidemic of Y. The nodes at state B are not infected and nodes at states X and Y attempt to infect the nodes they meet with their respective state. Such nodes immediately infect a non infected node while they cure a node of the opposing epidemic.

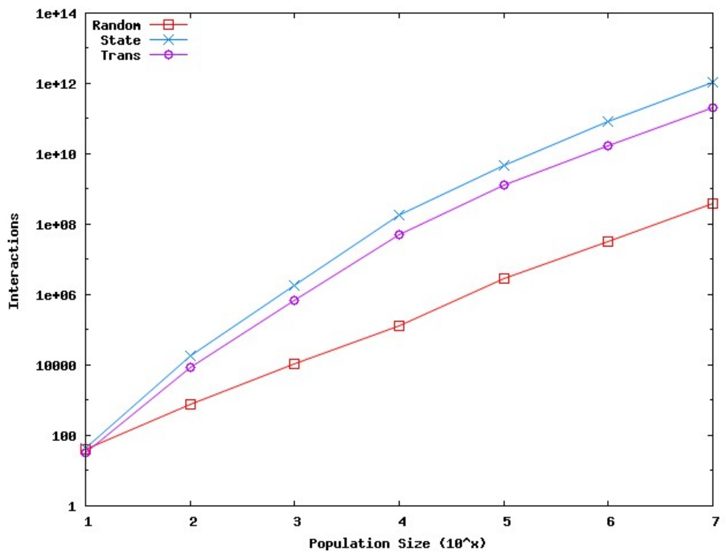


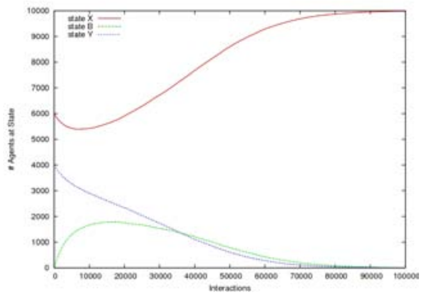
Fig. 6. Total number of interactions for the Majority protocol to stabilize for different schedulers.

We start by evaluating the stability of the protocol. Figure 6 depicts the results for the unrestricted schedulers. The results for the Random scheduler experimentally validate the theoretical analysis of [17]; from any non-blank configuration the protocol requires $\mathcal{O}(n \log n)$ interactions to reach a stable state. However the very interesting result lies in the behavior of the two protocol aware schedulers. For the first time they are outperformed (i.e., need larger number of interactions) by the protocol agnostic scheduler.

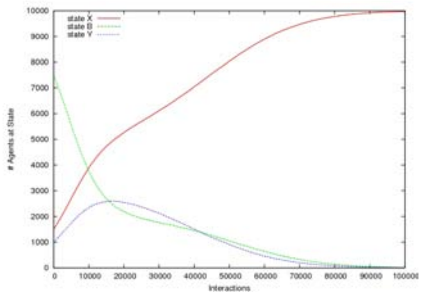
In order to explain this finding we use the results of the second set of experiments. Figure 7(a) depicts the protocol dynamics when 60% of the agents are set to state X and the remaining 40% to state Y. Initially, the population of both states drops since the interaction of X with Y generates new Bs. Interestingly, after a certain number of interactions n_y matches n_b and after that point they decrease with similar rates. As with the previous two protocols, the convergence to a stable configuration slows down as n_x increases and becomes very slow as $n_x \to n$.

To better understand the behavior of the protocol we conduct a second experiment where 20% of the agents are set to state X and 10% are set to state Y. Based on Figure 7(b), we observe that initially both n_x and n_y increase. After a while n_y is dominated by n_x and starts to drop. Again, at some point n_y matches n_b and after that point they decrease with similar rates.

In both experiments the key reason for achieving stability is the superiority of X over Y in number of agents. As n_x increases, the Random scheduler selects more often agents at state X than agents at state Y or B. Thus the X epidemic will propagate faster than Y. However this is not the case for the *State scheduler* that offers equal probabilities to all states or the *Transition Function scheduler* that also offers similar probabilities to all states. Under these protocol aware schedulers it will take a $\mathcal{O}(n^2)$ interactions (instead of $\mathcal{O}(n \log n)$) for state X to overdominate the other two states and lead the execution to a stable state.



(a) Initially X=6000, Y=4000



(b) Initially X=2000, Y=1000

Fig. 7. Number of agents at each state of the protocol per total interactions in a Compete graph with the Random Scheduler and when $n = 10000$.

8 Equivalence Between Schedulers

Definition 5. *Two fair probabilistic schedulers S_1 and S_2 are called* time equivalent *w.r.t. a protocol $\mathcal{A}$ iff all computations of $\mathcal{A}$ under S_1 and S_2 beginning from the same initial configuration take asymptotically the same expected time (number of steps) to convergence.*

Definition 6. *Two fair probabilistic schedulers S_1 and S_2 are called* computationally equivalent *w.r.t. a protocol $\mathcal{A}$ iff for all computations of $\mathcal{A}$ under S_1 and S_2 beginning from the same initial configuration, w.h.p., $\mathcal{A}$ stabilizes to the same output assignment (the output assignment of a configuration C is $y_C : V \to Y$ defined as $y_C(u) = O(C(u))$ for all $u \in V$).*

8.1 Not All Fair Probabilistic Schedulers are Time Equivalent

We use a simple protocol, called the *OR Protocol* or the *One-Way Epidemic Protocol*, based on an example of [9], in which each agent with input 0 simply outputs 1 as soon as it interacts with some agent in state 1. We, also, assume that the underlying communication graph is complete. Formally, we have $Q = X = Y = \{0, 1\}$ and the transitions defined by δ are the following:

$$\begin{array}{ll} (0,0) \to (0,0) & (1,0) \to (1,1) \\ (0,1) \to (1,1) & (1,1) \to (1,1) \end{array}$$

Essentially, if all agents have input 0, no agent will ever be in state 1. If some agent has input 1, given a fair scheduler, we expect that the number of agents with state 1 will increase and will eventually reach n. In both cases, if a fair scheduler is assumed then all agents will eventually stabilize to the correct output value, though an important fundamental questions is "how fast is stability reached?" and "how do different schedulers affect the performance of the protocol?".

In [**?**], Angluin et al. characterized the behavior of the OR Protocol in complete communication graphs as a *one-way epidemic*. They showed that the number of interactions for the epidemic to finish in the case of the Random Scheduler is $\Theta(n \log n)$ w.h.p., by using arguments from the well-known coupon collector problem.

Theorem 5. *The State Scheduler and the Transition Function Scheduler are time equivalent w.r.t. the One-Way Epidemic Protocol.*

Proof. The State Scheduler and the Transition Function Scheduler both require only $\mathcal{O}(n)$ interactions. In particular, the Transition Function Scheduler can choose only between transitions $(1, 0) \to (1, 1)$ and $(0, 1) \to (1, 1)$ that both increase the number of agents in state 1 by one. If initially at least one agent is in state 1, then in each step one agent goes from state 0 to state 1 (no new agents in state 0 emerge) and because the agents are n, in at most $n-1$ steps all agents will be in state 1 and stability will have been reached. In the case of the State Scheduler, assume the worst-case scenario in which initially only one agent is in state 1. Because the graph is complete, the interaction candidates are in the first step $(0, 0)$, $(0, 1)$, and $(1, 0)$ ($(1, 1)$ is not, because there exists only one agent in state 1). So, initially, there is a $2/3$ probability to select a transition that gives birth to a new 1. When this happens, in an expected number of 1.5 steps, all four left-hand sides of the rules of δ will be interaction candidates (until the step in which only one 0 remains, when again the probability of progress becomes $2/3$).

Because in all other possible configurations the probability to progress is $1/2$, it follows that progress is always made with at least probability $1/2$, which in turn implies that on average at most $2(n-1)$ (i.e. again $\mathcal{O}(n)$) steps are expected until stability is reached. □

The above discussion indicates that the performance of a population protocol clearly depends on the scheduler's functionality. In fact, it seems here that the additional knowledge, concerning the transition function, allowed to the State Scheduler and the Transition Function Scheduler provides us with interaction patterns that always lead to optimal computations. However, we can show that the same knowledge may also allow the definition of fair schedulers that lead the protocols to worst-case scenarios. To do so we will slightly modify the State Scheduler to obtain a new scheduler, called the *Modified Scheduler*. Let us consider the case in which the scheduler is *weakly protocol-aware* in the sense that it can only partition the rules of the transition function to classes (possibly with elements sharing some common property and assign some probability to each class.

Definition 7. *The Modified Scheduler selects from the class of the identity rules (rules that leave both the state of the initiator and that of the responder unaffected) with probability* $1-\varepsilon$ *and from all the remaining rules with probability* ε*, where* $0<\varepsilon<1$*. Those probabilities are then evenly divided into the corresponding class members. All other components of the Modified Scheduler's definition remain the same as in the case of the State Scheduler.*

Theorem 6. *The Modified Scheduler can lead the One-Way Epidemic Protocol to arbitrarily bad performance.*

Proof. First of all, note that the Modified Scheduler is fair with probability 1, because the transition probabilities may have been modified but still remain non-zero for non-loop arcs of T and independent of the number of steps. Consider now the situation in which $n-2$ nodes are initially in state 0 and the remaining 2 are in state 1. Because $n-2$ 0s have to be converted to 1s, it follows that the probability that the computation stabilizes in less than $n-2$ steps is 0. Let the random variable D denote the number of steps until the computation stabilizes (all agents become 1). We have already shown that $\mathbb{P}[D=i]=0$ for $i<n-2$. Note that $\mathbb{P}[D=i]$ equals $\mathbb{P}[$the last remaining 0 becomes 1 in step $i]$. Let also N_i denote the number of non-identity rules that have appeared in i steps. For the computation to stabilize in i steps, exactly $n-3$ non-identity rules must have been chosen in the first $i-1$ steps ($n-3$ 0s converted to 1s and one 0 remaining) and also a non-identity rule in the last step (the last 0 is converted to a 1). Note that N_i is a binomial random variable having parameters (i,ε).

Then for all $i \geq n-2$

$$\begin{aligned}
\mathbb{P}[D=i] &= \mathbb{P}[N_{i-1}=n-3] \cdot \mathbb{P}[\text{non-identity rule appears in step } i] \\
&= \left[\binom{i-1}{n-3}\varepsilon^{n-3}(1-\varepsilon)^{i-1-(n-3)}\right] \cdot \varepsilon \\
&= \binom{i-1}{n-3}\varepsilon^{n-2}(1-\varepsilon)^{i-n+2}
\end{aligned}$$

and the expectation of D is

$$\mathbb{E}[D] = \frac{(n-2)}{\varepsilon}.$$

The calculation of the above result can be found in the technical report at http://fronts.cti.gr/aigaion/?TR=93.

Obviously, $\mathbb{E}[D]$ can become arbitrarily large, by decreasing ε (that is, the probability that a non-identity rule is selected) and the theorem follows. If, for example, we set $\varepsilon = (n-2)/2^n$ given that $n > 2$ (because then $0 < \varepsilon < 1$), then the expected number of steps to convergence is exponential in the size of the population (equal to 2^n). □

Thus, it is evident that the fairness condition, as has been defined by Angluin et al. in [1], is not sufficient to guarantee the construction of protocols that perform well under all kinds of allowed schedulers. It seems that a protocol may perform optimally under some fair scheduler but at the same time reach its worst-case performance under some other, also provably fair, scheduler. Obviously, either some stronger definition of fairness needs to be proposed, that, for example, would characterize the Modified Scheduler as unfair in the case in which ε is far away from $1/2$, possibly because it always seems to prefer some class of rules from others, or maybe protocol-aware schedulers and other kinds of yet unknown schedulers that can be adjusted to lead to divergent performance scenarios, should somehow be formally prohibited.

Theorem 7. *There exists at least one protocol w.r.t. which some fair probabilistic schedulers are not time equivalent.*

Proof. Follows by comparing the expected running time of the One-Way Epidemic Protocol under the State and Transition Function Schedulers to its expected running time under the Random and Modified Schedulers (the latter expected times are from [**?**] and Theorem 6). □

8.2 Not all Fair Probabilistic Schedulers are Computationally Equivalent

Now we are about to show that, due to the weakness characterizing the selected notion of fairness, not only performance but also protocol correctness depends greatly on the underlying scheduler. Assume that each agent initially votes for one of some election candidates x and y or chooses to vote blank, denoted by b. If x is the majority vote, then we want every agent in the population to eventually output x, otherwise y (we assume here that the state of an agent is also its output). Now let us consider the following one-way protocol that was proposed in [17].

$$\begin{array}{ll} (x,b) \to (x,x) & (x,y) \to (x,b) \\ (y,b) \to (y,y) & (y,x) \to (y,b) \end{array}$$

In words, when an x meets a b it convinces it to vote x, when a y meets a b it convinces it to vote y, an x switches a y to the blank-undecidable state, and a y does the same to an x. Given an initial configuration of xs, ys and blanks that contains at least one non-blank, the goal is for the agents to reach consensus on one of the values x or y. Additionally, the value chosen should be the majority non-blank initial value, provided it exceeds the minority by a sufficient margin. In [17] it was proven that if the above protocol runs under the Random Scheduler on any complete graph with n nodes then with high probability consensus is reached in $\mathcal{O}(n \log n)$ interactions and the value chosen is the majority provided that its initial margin is $\omega(\sqrt{n \log n})$.

It seems that this is not the case when the underlying scheduler is the Transition Function Scheduler. Intuitively, the Transition Function Scheduler does not take into great account the advantage of xs. Let $N_x(t)$, $N_y(t)$, and $N_b(t)$ denote the number of xs, ys, and bs before step $t+1$, respectively. Note that when all xs, ys and bs appear in the population then the probability of $N_x(t+1) = N_x(t)+1$ is $1/4$ and the same holds for $N_y(t+1) = N_y(t)+1$. On the other hand, when the Random Scheduler is assumed, then the greater the number of xs, the more the arcs leading from xs to bs, thus the greater the probability of $N_x(t+1) = N_x(t)+1$.

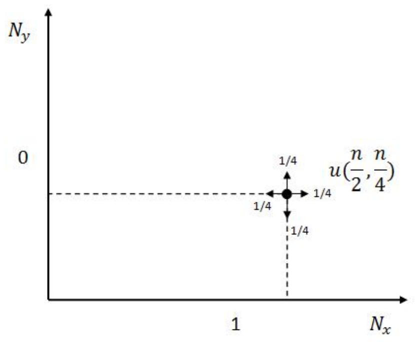


Fig. 8. The two-dimensional symmetric random walk. We show that the probability that the particle will reach the N_y axis before reaching the N_x axis is constant.

Lemma 1. *The Majority Protocol errs under the Transition Function Scheduler with constant probability, when $x = \Theta(y)$ in the initial configuration (x and y are used instead of N_x and N_y, respectively).*

Proof. The probability of the minority to win is equal to the probability that the symmetric walk (N_x, N_y) beginning from the initial point (x_0, y_0) will meet the N_y axis before meeting the N_x axis. The particle moves to each of its 4 neighboring points with probability 1/4 (see Figure 8, where 0 and 1 are the probabilities that we assign to the boundaries that constitute the collection of points for which the system stabilizes to a winning vote). The only exception is when the number of bs becomes equal to zero. But in this case the xs decrease by one with probability 1/2, the same holds for the ys and with probability 1 a b appears again and the walk returns to its initial symmetric distribution (to simplify the argument we ignore those states, because they do not strongly affect the probability that we want to compute). To the best of our knowledge, this kind of symmetric random walk in two dimensions has only been studied in [**?**], a paper cited by Feller [**?**], and is closely related to the Dirichlet problem. For any interior point (x, y), if $u(x, y)$ denotes the probability that the minority wins (the walk meets the N_y axis before meeting the N_x axis), then

$$u(x,y) = \frac{1}{4}(u(x+1,y) + u(x,y+1) + u(x-1,y) + u(x,y-1)), \qquad (1)$$

and we are interested in the value of $u(x, y)$ when $x = \Theta(y)$, that is the initial number of xs and the initial number of ys are of the same order (e.g. $x = n/2$ and $y = n/4$). The homogeneous solution of (1) is $u(x,y) = \frac{x+y}{2n}$ and the general (with the boundary conditions into account) is $\frac{x+y}{2n} + f(x,y)$, where $f(x, y)$ is a particular non-homogeneous solution. When $x, y = \Theta(n)$ the $u(x, y)$ behaves as the homogeneous, thus $u(n/2, n/4)$ is equal to 3/8, which is constant. □

Theorem 8. *There exists at least one protocol w.r.t. which two fair probabilistic schedulers are not computationally equivalent.*

Proof. The Random Scheduler is not computationally equivalent to the Transition Function Scheduler w.r.t. the Majority Protocol, because there exists some initial margin in the case in which the majority is x, which is $\omega(\sqrt{n \log n})$ and also the initial number of xs and the initial number of ys are of the same order. For example, in the case in which $x = 3n/4 - k$ (where $k \ll n$) and $y = n/4$, x and y are of the same order and $x - y \simeq n/2 = \omega(\sqrt{n \log n})$ for sufficiently large n. But given an initial configuration satisfying the above dynamics, under the Random Scheduler the protocol w.h.p. stabilizes to a majority winning configuration, while under the Transition Function Scheduler from Lemma 1 there is a constant probability that the protocol will stabilize to a minority winning configuration. Thus, it does not hold that w.h.p. those schedulers make the protocol stabilize to the same output assignment (see again Definition 6) and the theorem follows. □

9 Future Research Directions

In the area initiated by the proposal of the Population Protocol model [1] many unresolved problems remain. The Population Protocol model makes absolutely minimal assumptions about the underlying system. The agents follow a completely unpredictable movement, they cannot store unique identifiers, and even a single Byzantine failure can lead to global failure of the system. How can we readjust (relax) those assumptions to more accurately reflect practical sensor network systems? For example in [18], Guerraoui and Ruppert assumed that the agents are equipped with read-only IDs (from the industry) and that they are also capable of storing a constant number of other agents' IDs. In this manner they obtained a very strong model, which they call the *Community Protocol* model (because the agents are now named individuals, like members of a community), that can solve any decision problem in $NSPACE(n \log n)$ (and is additionally robust to Byzantine failures of a constant number of agents). Moreover in [7] they allowed the communication links to store states from a set of cardinality that is independent of the population size, to obtain the *Mediated Population Protocol* model that is also stronger than the Population Protocol model. In the case of wireless communication is there some architecture to reasonably implement the proposed model without using a global storage (for more information about the global storage idea the reader is referred to `http://fronts.cti.gr/aigaion/?TR=65`, i.e. the corresponding technical report of [7])? In the latter model either an exact characterization of the class of solvable problems has to be found or at least some impossibility results should appear to provide a first insight of what the model is not capable of computing (a first attempt can be found in [8], and in [7] it was proven that all stably computable predicates belong to $NSPACE(m)$, where m denotes the number of edges of the communication graph). Finally, how can someone verify safely and quickly, in a distributed or centralized way, that a specific protocol meets its design objectives? This is a crucial problem that remains open and has to be solved if our protocols are to be run in real critical application scenarios (e.g. fire detection, vehicle malfunction report and/or fixing, patient health status report, and generally any kind of critical data transmission).

References

1. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC), New York, NY, USA, ACM (2004) 290–299
2. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed Computing **18** (2006) 235–253
3. Queille, J., Sifakis, J.: Fairness and related properties in transition systemsa temporal logic to deal with fairness. Acta Informatica **19** (1983) 195–220

4. Kröller, A., Pfisterer, D., Buschmann, C., Fekete, S.P., Fischer, S.: Shawn: A new approach to simulating wireless sensor networks. In: Design, Analysis, and Simulation of Distributed Systems (DASD05). (2005) 117–124
5. Diamadi, Z., Fischer, M.J.: A simple game for the study of trust in distributed systems. Wuhan University Journal of Natural Sciences **6** (2001) 72–82 Also appears as Yale Technical Report TR-1207, January 2001, `ftp://ftp.cs.yale.edu/pub/TR/tr1207.ps`.
6. Angluin, D., Aspnes, J., Chan, M., Fischer, M.J., Jiang, H., Peralta, R.: Stably computable properties of network graphs. In: 1st IEEE/ACM International Conference on Distributed Computing in Sensor Systems (DCOSS 2005). Volume 3560 of Lecture Notes in Computer Science., Springer (2005) 63–74
7. Chatzigiannakis, I., Michail, O., Spirakis, P.G.: Mediated population protocols. In: 36th Internatilonal Collogquium on Automata, Languages and Programming (ICALP 2009). (2009) 363–374
8. Chatzigiannakis, I., Michail, O., Spirakis, P.G.: Brief announcement: Decidable graph languages by mediated population protocols. In: 23rd International Symposium on Distributed Computing (DISC 2009). (2009) 239–240
9. Aspnes, J., Ruppert, E.: An introduction to population protocols. Bulletin of the European Association for Theoretical Computer Science **93** (2007) 98–117
10. Chatzigiannakis, I., Michail, O., Spirakis, P.G.: Recent advances in population protocols. In: 34th International Symposium on Mathematical Foundations of Computer Science (MFCS 2009). (2009) 56–76
11. : (The network simulator – ns2) http://www. isi. edu/nsnam/ns/.
12. : (OPNET modeller, OPNET technologies, inc.) http://www.opnet.com.
13. : (AlgoSenSim) URL: http://tcs.unige.ch/doku.php/code/algosensim/overview.
14. Chatzigiannakis, I., Koninis, C., Prasinos, G., Zaroliagis, C.: Distributed simulation of heterogeneous systems of small programmable objects and traditional processors. In: 6th ACM Workshop on Mobility Management and Wireless Access (MOBIWAC 08), ACM, ACM (2008) 133–140
15. Karagiannis, M., Chatzigiannakis, I., Rolim, J.: WSNGE: A platform for simulating complex wireless sensor networks rich network visualization and online interactivity. In: 42nd Annual Simulation Symposium (ANSS 2009). (2009) IEEE Press, to appear.
16. Angluin, D., Aspnes, J., Eisenstat, D.: Fast computation by population protocols with a leader. In: 20th International Symposium on Distributed Computing (DISC). Volume 4167 of Lecture Notes in Computer Science., Springer (2006) 61–75
17. Angluin, D., Aspnes, J., Eisenstat, D.: A simple population protocol for fast robust approximate majority. In: 21st International Symposium on Distributed Computing (DISC). Volume 4731 of Lecture Notes in Computer Science., Springer (2007) 20–32
18. Guerraoui, R., Ruppert, E.: Names trump malice: Tiny mobile agents can tolerate byzantine failures. In: 36th Internatilonal Collogquium on Automata, Languages and Programming (ICALP 2009). (2009) 484–495