Computational Models for Wireless Sensor Networks: A Survey*

Apostolos Filippas[2], Stavros Nikolaou[2], Andreas Pavlogiannis[2],
Othon Michail[1,2], Ioannis Chatzigiannakis[1,2], and Paul G. Spirakis[1,2]

[1] Research Academic Computer Technology Institute (RACTI), Patras, Greece
[2] Computer Engineering and Informatics Department (CEID), University of Patras, Patras, Greece. **
Email:{filippas, snikolaou, paulogiann}@ceid.upatras.gr
{michailo, ichatz, spirakis}@cti.gr

Abstract

Here we survey various computational models for Wireless Sensor Networks (WSNs). The *population protocol* model (PP) considers networks of tiny mobile finite-state artifacts that can sense the environment and communicate in pairs to perform a computation. The *mediated population protocol model* (MPP) enhances the previous model by allowing the communication links to have a constant size buffer, providing more computational power. The *graph decision MPP model* (GDM) is a special case of MPP that focuses on the MPP's ability to decide graph properties of the network. Another direction towards enhancing the PP is followed by the PALOMA model in which the artifacts are no longer finite-state automata but Turing Machines of logarithmic memory in the population size. A different approach to modeling WSNs is the *static synchronous sensor field model* (SSSF) which describes devices communicating through a fixed communication graph and interacting with their environment via input and output data streams. In this survey, we present the computational capabilities of each model and provide directions for further research.

Keywords: population protocols, wireless sensor networks, diffuse computation

1 Introduction

Given the emergence of pervasive computing *Wireless Sensors Networks* (WSNs) will play an increasingly important role in future critical systems' infrastructure and should subsequently be correct, reliable and robust. Theoretical models for WSNs have recently received great attention as they constitute an abstract but yet formal and precise method for understanding the laws and inherent properties of this widely applicable new technology. The *population protocol* (PP) model was designed to represent WSNs consisting of severely limited mobile agents with no control over their own movement that compute by direct pairwise interaction.

* This work has been partially supported by the ICT Programme of the European Union under contract number ICT-258885 (SpitFIRE).

** A. Filippas, S. Nikolaou, and A. Pavlogiannis are undergraduate students at the time of submission.

A reason towards studying extremely limited computational devices is that in real WSNs application scenarios, having limited resources is inevitable as constraints on each node's size and cost translate into severe limitations in power, storage, processing and communication. Another reason is that the minimalistic nature of the PP model makes it a concrete and realistic model for WSNs and provides a starting point in the understanding of their limitations and capabilities. Furthermore, while at a first glance PPs may seem like they are only related to WSNs, applications in other fields exist and are discussed in [Angluin, Aspnes, Diamadi, Fischer and Peralta (2006)]. For example, PPs can be used in modeling collections of molecules undergoing chemical reactions. A defining characteristic of the PPs that diversifies them from classic distributed systems is the total inability of the agents to control or predict their underlying mobility pattern. Their movement pattern is usually the result of some natural phenomenon, for example river flow, wind, or sensors attached to some carrier, and is known as *passive mobility*.

However, the basic model may sometimes prove to be too minimalistic. The next natural step is to enhance the PP model with realistic and implementable assumptions in order to increase its computational power. First, in the MPP model the PP is extended with a Mediator capable of storing limited information for each communication link. Another extension is the PALOMA model, where agents are equipped with memory logarithmic in the population size, enabling them to have unique ids. This assumption extremely strengthens the model and is surprisingly natural: only 256 bits are required for 2^{256} agents, which is an astronomical population size. Another model is the SSSF, where mobility no longer holds. Agents operate synchronously and exchange streaming data with their environment. In this way, SSSF can target sensing problems that require constant communication with the environment.

2 The Population Protocol Model

2.1 Formal Definition

A *Population Protocol (PP)*, proposed in [Angluin, Aspnes, Diamadi, Fischer and Peralta (2006)], consists of a finite set of states Q, finite input and output alphabets X and Y, an input function $I : X \to Q$ mapping inputs to states, an output function $O : Q \to Y$ mapping states to outputs, and a *transition function* $\delta : Q \times Q \to Q \times Q$ that describes how pairs of agents interact. If $\delta(p, q) = (p', q')$, we call $(p, q) \to (p', q')$ a *transition* and we define $\delta_1(p, q) = p'$ and $\delta_2(p, q) = q'$. If for all $(p, q) \in Q^2$ there is only one possible transition $(p, q) \mapsto (p', q')$, then the PP is *deterministic*.

A PP runs on a directed communication graph $G = (V, E)$, where $n = |V|$ is the number of nodes, and therefore the population size, and $m = |E|$ is the number of edges. Intuitively, an edge $(u, v) \in E$ means that u and v are able to interact with u

playing the role of the *initiator* and v being the *responder*. The distinct roles of the two agents is a fundamental symmetry breaking assumption of the PP model.

A *(population) configuration* $C : V \to Q$ is a snapshot of the population states and can be described by a vector of all of the agent states. Let C,C' be population configurations and u, v be distinct agents. We say that C *goes to* C' *via encounter* $e = (u, v)$, denoted $C \overset{e}{\to} C'$, if C' is the result of the interaction of the pair (u, v) under configuration C. We say that C *goes to* C' *in one step*, denoted $C \to C'$, if $C \overset{e}{\to} C'$ for some encounter $e \in E$. We write $C \overset{*}{\to} C'$ if there is a sequence of configurations $C = C_0, C_1, \dots, C_n = C'$, such that $C_i \to C_{i+1}$ for all i, $0 \leq i < n$, in which case we say that C' *is reachable from* C.

An *execution* is a finite or infinite sequence of configurations $C_0, C_1, \dots$ where C_0 is the initial configuration and $C_i \to C_{i+1}$ for all $i \geq 0$. The schedule of interactions can be thought as being chosen by an adversarial scheduler, on whom a *strong global fairness condition* is imposed. Intuitively, the scheduler cannot partition the agents into non-communicating clusters by avoiding a single interaction forever. More formally, if C occurs infinitely often in a fair execution and $C \to C'$, then C' also occurs infinitely often in the execution. A *computation* is an infinite fair execution.

Two of the most critical properties of the PP model are its *uniformity* and *anonymity*. A PP is uniform if its specification is independent of the population size. PPs are anonymous, as the agents are not equipped with unique identifiers and are treated in the same way by the transition function.

2.2 Stable Computation

An *input assignment* $x : V \to X$ is a mapping describing the input of each agent in the population. An *output assignment* $y : V \to Y$ is a mapping describing the output of each agent. Let $\mathcal{X} = X^V$ denote the set of all input assignments, $\mathcal{Y} = Y^V$ the set of all output assignments and $\mathcal{C} = Q^V$ the set of all possible configurations. The input function is now naturally extended to a mapping from input assignments to configurations, $I : \mathcal{X} \to \mathcal{C}$, by writing $I(x) = C_x$. Similarly, the output function is naturally extended to a mapping from configurations to output assignments, $I : \mathcal{C} \to \mathcal{Y}$, by writing $O(C) = y_C$.

Unlike a Turing Machine, a PP controlled by an infinitely working scheduler does not halt but rather converges. For this reason, the notion of stability is introduced. A configuration C is *output-stable* if $O(C) = O(C')$ for all C' reachable from C. An infinite computation *output-stabilizes* if it contains an output-stable configuration C, in which case we say that it *stabilizes to output* $y = O(C)$. A PP $\mathcal{A}$ running in a population V *stably computes* an input-output relation $R_\mathcal{A}$ if $\forall x \in \mathcal{X}, \forall y \in \mathcal{Y}, R_\mathcal{A}(x, y)$ holds iff there is a computation of $\mathcal{A}$ beginning in configuration $I(x)$ that stabilizes to output y.

In the special case in which $R_{\mathcal{A}}$ is single-valued, we write $F_{\mathcal{A}}(x) = y$ and say that $\mathcal{A}$ stably computes the partial function $F_{\mathcal{A}} : \mathcal{X} \to \mathcal{Y}$.

A *predicate* is a function $p : \mathcal{X} \to \{0, 1\}$ and is *stably computable*, if some PP stably computes it. A predicate p is *symmetric* if $\forall x_1, x_2 \in \mathcal{X}$ where x_1 is a permutation of x_2's symbols, it holds that $p(x_1) = p(x_2)$. Because of the anonymity property and the communication graph's symmetry, the PP model can stably compute only symmetric predicates. *Semilinear predicates* are predicates whose support is a semilinear set; that is, a finite union of linear sets. A set of vectors in $\mathbb{N}^k$ is linear if it is of the form $\{b + l_1 a_1 + l_2 a_2 + ... + l_m a_m \mid l_i \in \mathbb{N}\}$, where b is a base vector, a_i are basis vectors and l_i are non-negative coefficients. As proved in [Ginsburg and Spanier (1966)] , semilinear predicates are precisely those that can be defined by first-order logical formulas in Presburger arithmetic. Authors proved in [Angluin, Aspnes and Eisenstat (2006)], that *a predicate is stably computable by the basic PP model iff it is semilinear*, thus providing an exact characterization of the computational power of the model.

3 Mediated Population Protocols

Since the computational power of the basic PP model has been characterized completely, several extensions have been proposed in order to circumvent some of its fundamental restrictions. In [Chatzigiannakis et al. (2009*b*)], the authors considered the case in which not only the agents but also the communication links are equipped with a buffer of constant size. Intuitively, the interaction graph can then be seen as a distributed memory of size $\mathcal{O}(n + m)$ instead of just $\mathcal{O}(n)$, where m is the number of edges. However, since the agent memory remains constant, this distributed memory is not easily manageable. Despite the inherent difficulties, the authors managed to address some problems concerning graph properties, as well as compute predicates that are non-semilinear.

3.1 The MPP Model

The *Mediated Population Protocol* (*MPP*) model consists of the basic Population Protocol Model with the following extensions: a *set of edge states* S and the transition function $\delta : Q \times Q \times S \to Q \times Q \times S$.

As in the PP model, the agents receive an input symbol and enter an initial state. We also assume that the edges are initialized through an *edge initialization function* ι which is not part of the protocol but models some preprocessing on the network that has taken place before the protocol's execution. The definition of a *network configuration* is slightly modified to take into account the agent states as well as the edge states. Finally, the notions of *execution*, *fairness* and *stable computation* remain unchanged.

3.2 Computation of a Non-Semilinear Predicate

Problem 1. Upon initialization, every agent receives an input symbol a, b or c. Let N_x be the number of agents initialized with symbol x. Compute the predicate $N_c = N_a \cdot N_b$.

Note that on complete communication graphs the number of edges between agents with input symbol a and agents with input symbol b is $N_a \cdot N_b$. With that in mind it is easy to devise a protocol that stably computes the multiplication predicate.

Assume that each agent receives an input symbol from $X = \{a, b, c\}$ and enters a special state q_a, q_b or q_c. Moreover, we assume that each edge is either $marked$ or $unmarked$. Initially, all edges are $unmarked$. The computation goes as follows: whenever two agents with input symbols a and b interact via an $unmarked$ edge, the a-agent enters a special state $\dot{a}$. Being in this state, the agent will only interact with a c-agent in order to mark it.

Due to the observation made above, a fair execution will lead all the a-agents in the state $\dot{a}$ exactly $N_a \cdot N_b$ times in total. Each time an agent enters state $\dot{a}$ it searches for a c-agent to mark, while the $\dot{a}$-agent enters state a. If the predicate holds, such markings will eventually leave agents neither in state c nor in state $\dot{a}$. In any other case, some redundant c or $\dot{a}$ states will always be present in the population (denoting that $N_c > N_a \cdot N_b$ or $N_c < N_a \cdot N_b$, respectively).

But how can the population determine whether no agent is either in state c or in state $\dot{a}$? Notice that this is a semilinear predicate and thus can be stably computed by a population protocol. So, the population can launch a simple protocol in parallel, and determine the outcome of the computation of the multiplication predicate.

It is well known that the multiplication predicate is not semilinear, thus the MPP model is strictly more powerful than the basic PP. To gain an upper bound in computability one can easily construct a nondeterministic TM M of space $\mathcal{O}(m)$ that simulates an MPP having m edges. Intuitively, M has to store a network configuration each time. At each step a nondeterministic choice selects a pair of agents to interact, as well as the edge between them, and updates the configuration appropriately. In order to accept, M has to compute whether a configuration C in which all agents output 1 is reachable from the initial configuration. Moreover, it needs to verify that no configuration C' is reachable from C in which at least one agent changes its output. This condition is the complement of a similar reachability problem. Since $NSPACE$ is closed under complement [Immerman (1988)], M can successfully verify this condition and thus simulate the given MPP correctly. Whether this bound is tight remains an open problem.

4 GDM

The *Graph Decision Mediated Population Protocol* (*GDM*) model is a special case of the MPP model concerned with *graph languages*. The following definition holds w.r.t. some fixed graph universe U.

Definition 1. *A graph language L is a subset of U containing communication graphs that possibly share a common property.*

Some examples of graph languages are: The graph language consisting of all strongly connected members of U, $L = \{G \in U | \text{G contains a directed hamiltonian path}\}$, and $L = \{G \in U | \text{G has an even number of edges}\}$.

In general, we are not just interested in recognizing whether a certain property holds in a communication graph, but also obtaining a specific subgraph that satisfies that property. In order to do that, the authors equipped the model with an *output instruction r* that informs the output viewer on how to interpret the output. For example, if a problem had to do with choosing some nodes, a protocol would mark those nodes and r would be: *Choose any marked node*.

The authors in [Chatzigiannakis et al. (2009*a*)] showed that in order to compute any non-trivial graph language, the network graph has to be weakly connected. Indeed, communication graphs that are not even weakly connected consist of several weakly connected components. Then any nontrivial property might hold in only some of them. Since agents between different weakly connected components cannot interact, no uniform decision can be made.

With all the above in mind, we demonstrate the power of the GDM model by presenting a protocol that computes a graph language.

4.1 A Simple Graph Protocol

Definition 2. *Given a graph $G = (V, E)$, a matching M in G is a set of pairwise non-adjacent edges; that is, no two edges in M share a common vertex.*

Problem 2. Given a graph $G = (V, E)$, find a maximal matching; that is, a matching M such that when any edge not in M is added in M, it is no longer a matching.

In Protocol 1 we present a simple approach to solve the Maximal Matching problem.

It is easy to verify that no two adjacent edges go to state 1, since the node that they share can only take the $q_0 \to q_1$ transition only once. Moreover, due to fairness, any two agents in state q_0 will eventually interact, thus, marking the shared edge as “1”.

Finally, in order to address more complicated problems, the authors have observed that the class of graph languages that are stably computable by the GDM model is closed under complement, union and intersection operations.

Protocol 1 *Maximal Matching*

```
1: $X = \{0\}, Y = \{0, 1\}$
2: $Q = \{q_0, q_1\}, S = \{0, 1\}$
3: $I(0) = q_0$
4: $\iota(e) = 0, \forall e \in E.$
5: $r$: “Get every $e \in E$ for which $s_e = 1$ (where $s_e$ is the state of e)”
6: $\delta(q_0, q_0, 0) \to (q_1, q_1, 1)$
```

5 Extending to Logarithmic Space

In [Chatzigiannakis et al. (2010)], the authors examined the case of equipping the agents with read/write memory that is logarithmic in the size of the population. Notice that this is the weakest assumption required to support unique ids across the population. However, in order to preserve the *anonymity* constraint, all the agents must be identical, thus no such ids could be present at the beginning of the computation. Despite this restriction, the authors devised an algorithm that generates unique ids from scratch.

The $PALOMA$ (standing for “PAssively-mobile LOgarithmic MAchines”) model assumes that agents are Turing Machines of memory that is logarithmic in the size of the population. An interaction between two agents results in the exchange of some information that can be logarithmic in length. After this exchange has taken place, each agent will process the information it has received. As this will usually take some time, each agent is equipped with a *working flag* that indicates whether an agent is busy doing internal computation, or waits for an interaction. When the working flag of an agent is set, any interactions in which that agent is one of the participants are aborted.

As the agents now have enough memory to store unique ids, the authors exploited the symmetry breaking property of the initiator-responder distinction and provided a protocol that generates unique ids from scratch. Intuitively, at first all agents have the same id, that is, 0. At each point, any interaction between agents that have the same id will cause the id of the initiator to be incremented. Of course, the protocol itself cannot detect the termination of the id generation process. However, it is easy to verify that this process will eventually stabilize to a correct id-assignment when the agents having the ids $n - 2$ will interact (we know that this will eventually happen because of the fairness condition). Observe that at that point the agents are not only distinguishable, but also aware of the size of the population n.

Having each agent associated with a unique id, the authors managed to simulate a deterministic TM of space $\mathcal{O}(n \log n)$. Moreover, by computation reinitiation techniques and by exploiting the fairness condition the authors managed to simulate a non-deterministic TM of the same space bounds. Finally, by applying some techniques presented in the case of the MPP model it is easy to verify that any PALOMA protocol can be simulated by a nondeterministic TM of space $\mathcal{O}(n \log n)$. This concludes the characterization of the PALOMA computability: it is exactly *the class of symmetric predicates that are computed by a nondeterministic TM of space $\mathcal{O}(n \log n)$*.

6 The Sensor Field Model

The *Static Synchronous Sensor Field* (SSSF) is a recently proposed model [Alvarez et al. (2009)] which addresses networks of tiny heterogeneous artifacts and allows processing over constant flows (streams) of data originating from the environment. This feature is not provided by the models presented so far and is required by various sensing problems.

The *SSSF* model consists of a set of *devices* and a *communication graph G*. The devices come with the ability to send and receive streaming information to and from the environment. The model permits however, the inclusion of devices with various constrains like for example absence of sensing capabilities or restricted memory capacity, allowing heterogeneity in the network and breaking uniformity (in contrast with the PP model). An additional assumption is that the devices are synchronized and in each round they do the following: They receive data from the environment (items of alphabet U) and their neighbors (items of alphabet X also called *communication items*) in G. Then they apply the transition function (δ_k for *each* device k) on their current state (from the set of states Q_k) and input, updating their configuration. Finally they send data to their neighbors and environment (items of V). It should be noted that the transition function is not global as in the the models presented so far. Also note that G is static, so that every round a device *interacts* with the *same neighboring devices* in G. This differs from the previously described models where *arbitrary pairs* of neighboring agents interact.

Each device k (w.l.o.g. from now on only devices with sensing capabilities are considered) is associated with a *data stream* w_k. A data stream is a sequence of data items w_k^i for $i \geq 1$ and each such item is received in round i by the device k as an input from the environment. In contrast to the models described in other sections the devices in SSSF receive and use environmental input constantly during the computation. Furthermore, in SSSF each device k outputs in each round $i \geq 1$ a data item v_k^i.

A computation in SSSF is an infinite sequence of snapshots of the devices' network. The first snapshot is a tuple of the initial states of all devices. Each subsequent snapshot represents one round t and includes the states of all devices as well as the inputs and outputs (both environment and communication data items) of every device on that round. The devices' states in each snapshot corresponding to round t are produced by application of the transition function δ_k for each device k on the states of the previous snapshot (representing round $t-1$) using the input data of the round.

Due to the streaming data exchange with the environment the computation of SSSF is related to a *stream behavior* which associates the input streams of all devices to their output streams. Note that both input and output streams concern the environment and not the communication with neighboring devices. This association defines the function f_F that a SSSF F computes. Given a tuple of input data streams and an integer $t \geq 1$, f_F outputs the data streams produced by the computation of F on the previous input until round t. It is said that *sensor field F computes function f* (defined on data streams) *with latency d* if given a tuple of input data streams, F outputs the t-th element of f for the same input, on round $t + d$.

A SSSF F solves a sensing problem Π if given a n-tuple of data streams $\mathbf{u} = (u_k)_{1 \leq k \leq n}$ as input, it computes a m-tuple $\mathbf{v} = (v_k)_{1 \leq k \leq m}$ (where $m \leq n$) so that the relation R_Π defined by the problem Π, is satisfied between $\mathbf{u}$ and $\mathbf{v}$. An example of such problem is the *Average Monitoring*, studied in [Alvarez et al. (2009)], where given n data streams $(u_k)_{1 \leq k \leq n}$ for $n \geq 1$, n data streams $(v_k)_{1 \leq k \leq n}$ must be computed so that each device k outputs in any round t the average of the input data values of devices $1, \ldots, k$ in t. Optimal algorithms have been proposed for the above problem w.r.t. latency and other metrics, on certain topologies of G.

Although the basic SSSF model does not place any memory restrictions, it can be easily adapted to support computation on memory-restricted devices. A variation of SSSF called *constant memory SSSF* assumes that each device has constant-size memory to store input data (originating from the environment and neighbors) and therefore only a constant number of packets (chosen arbitrarily) can be received in each round whereas sending packets can be done via a *broadcast operation* to all neighbors.

For each SSSF F a language $L(F)$ is associated to its function f_F including the tuples of all input data items and their corresponding to f_F outputs. It has been proven [Alvarez et al. (2009)] that the languages associated to constant memory SSSFs belong to the class $DSPACE(n + m)$ where n is the number of devices and m the communication links. Finally, including additional non-sensing devices in G (since as stated before SSSF is a non-uniform model) enables the solution of monitoring a property P computable in polynomial time, using a constant memory SSSF of polynomial size and latency w.r.t. the number of sensing devices n. Clearly, there is a trade-off between the memory size of each device and the number of additional devices in G.

7 Conclusions - Future Research Directions

So far, we have presented some computational models for the field of wireless sensor networks. Although most issues concerning the basic PP model have been closed, there exists a wide area of open problems that constitute an interesting direction for future work on the subject in general.

It is yet unknown whether the MPP model can take advantage of its whole memory effectively. It seems that a careful manipulation of the graph edges could let us align the agents to a single line, thus building the infrastructure of a TM. However, no such proof is yet available.

It is also interesting to explore the computational power of populations equipped with $f(n)$ memory in general. Are there any thresholds of $f(n)$ that define the behavior of such populations? For example, it seems that when $f(n) = \Omega(\log n)$, the simulation of a TM of $n \cdot f(n)$ memory is possible. Does a space hierarchy theorem hold for symmetric predicates in these cases? And what is the computational power of populations when $f(n) = o(\log n)$?

Finally, the SSSF model is still young and there are many directions to explore. The support of dynamic communication graphs which is a common scenario in real systems or the potential need for creating and managing virtual topologies so that the static communication graph analysis can be used, require both additional effort and study. The energy consumption has not yet been considered although it is an important performance measure and therefore an appropriate energy model would be an interesting feature to import.

References

Alvarez, C., Duch, A., Gabarro, J. and Serna, M. (2009), Sensor field: A computational model, *in* 'Algorithmic Aspects of Wireless Sensor Networks', pp. 3–14.

Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M. J. and Peralta, R. (2006), 'Computation in networks of passively mobile finite-state sensors', *Distributed Computing* pp. 235–253.

Angluin, D., Aspnes, J. and Eisenstat, D. (2006), Stably computable predicates are semilinear, *in* 'PODC '06: Proceedings of the 25th annual ACM Symposium on Principles of Distributed Computing', ACM Press, New York, NY, USA, pp. 292–299.

Chatzigiannakis, I., Michail, O., Nikolaou, S., Pavlogiannis, A. and Spirakis, P. (2010), Passively mobile communicating logarithmic space machines, Technical Report FRONTS-TR-2010-16. `http://fronts.cti.gr/aigaion/?TR=154`.

Chatzigiannakis, I., Michail, O. and Spirakis, P. (2009*a*), Decidable graph languages by mediated population protocols, *in* '23rd International Symposium on Distributed Computing (DISC 2009)', Vol. 5805 of *LNCS*, pp. 239–240.

Chatzigiannakis, I., Michail, O. and Spirakis, P. (2009*b*), Mediated population protocols, *in* '36th International Colloquium on Automata, Languages and Programming (ICALP 2009)', Vol. 5556 of *LNCS*, pp. 363–374.

Ginsburg, S. and Spanier, E. H. (1966), 'Semigroups, Presburger formulas, and languages.', *Pac. J. Math.* **16**, 285–296.

Immerman, N. (1988), 'Nondeterministic space is closed under complementation', *SIAM J. Comput.* **17**(5), 935–938.