# Sensor Field: A computational model [⋆]

C. Àlvarez, A. Duch, J. Gabarro, and M. Serna

ALBCOM. LSI Dept. Universitat Politècnica de Catalunya, Barcelona.
{alvarez,duch,gabarro,mjserna}@lsi.upc.edu

**Abstract.** In this work we introduce a formal model for studying networks of tiny artifacts, the static synchronous sensor field model (SSSF). The model consider that these devices communicate among them by means of a communication graph. A sensor field interacts with the environment with input/output data streams. We start analyzing the behavior of networks in which at every step each sensor takes a new input data item and outputs (with some possible latency) an output data item. Accordingly we introduce adequate performance measures like latency, message number, or message length. We study two sensing problems the Average Monitoring and the Alerting problems. For the Average Monitoring problem we give upper bounds on the latency, number of required messages and their size and optimal algorithms to solve it in specific topologies. We show that the Alerting problem can be solved with sensing devices of constant memory. When the SSSFs are allowed to use only devices with constant memory capacity we demonstrate that the decisional version of the functions computed by such SSSFs are in the class DSPACE($max(n, m)$) where $n$ is the number of nodes of the communication graph and $m$ its number of edges. If in this kind of SSSFs we consider the possibility of using more non-sensing tiny devices than input data streams we show that monitoring a property that is polynomial time computable can be solved by SSSFs of polynomial size and latency with respect to the number of input data streams.

**Keywords**. Tiny artifacts, sensors, sensor networks, computational models, data streams, average monitoring problem, alerting problem, static synchronous sensor field.

## 1 Introduction

The use of networks of tiny artifacts is becoming a key ingredient in the technological development of XXI century societies. An example of those networks are the networks with sensors, where some of the artifacts have the ability of sensing the environment and communicate among themselves. Depending on the purpose for which the networks where designed, they could have certain characteristics: pervasiveness, ubiquity, massive scale, device hetereogenity, subnetwork autonomy, decentralization, human interaction and emergent behavior. Therefore, the study of such systems involve several and very different areas of computing: hybrid and distributed systems, communication systems and protocols, circuit design, multi-agent systems, ad-hoc networks, algorithmic design, complexity theory or pervasive and ubiquous computing. Consequently, one can assume that this kind of systems are intrinsically complex. In fact, there is no easy way to design a universal sensor network that acts properly in all possible situations. In recent years, there has been an important amount of experimental results regarding these kinds of networks, in particular on networks with sensors. It is important to understand the computational process and behavior of the different types of artifact's networks, which will help in taking the maximum profit of the networks and in designing more efficient next generation networks. For instance in the particular case of networks with sensors several proposals (taxonomies and surveys) that elucidate the distinguishing

[⋆] Partially supported by the ICT Program of the European Union under contract number 215270 (FRONTS). The first, third, and fourth authors were also supported by the spanish project TIN-2007-66523 (FORMALISM). The second author was also supported by the spanish project TIN2006-11345 (ALINEX-2).

features of sensor networks have been published ([1, 6, 13, 16–18]). These proposals state clearly the need of formal models that capture the clue characteristics of sensor networks and that enable their study through the possibility of predict their behavior, efficiency and optimality in any kind of applications.

In this work we propose a formal model to capture the distinguishing features of networks including sensing devices proposed in the literature. The general sensing setting can be described by two elements: the observers or end users and the phenomenon, the entity of interest to the observers that is monitored and analyzed by network with sensors. The corresponding information is discretized in two ways: first the environment is sampled on a discrete set of locations (sensor positions), and second the measures taken by the sensors are digitalized to the corresponding precision. To analyze the correctness and performance of the system we are faced with a double task; on one side there is a computational problem to be solved by a particular network; on the other hand, it is necessary to assess whether the observation of the phenomenon is valid. Both tasks will require different analysis tools. The distinctive peculiarities of the computational system define new parameters to be evaluated in order to measure the performance of the system. Metrics are needed to allow us to estimate the suitability of an specific or generic network topology or the possibility of emergent behavior with pre-specified requirements.

In this proposal we focus on the first task combining the notion of graph automata [3] together with data streams, a combination inspired in similar ideas developed in the context of concurrent programming [8]. Existing models coming from distributed systems [14], hybrid systems and ad-hoc networks [7, 11] and circuit design [9, 15] capture some of such networks, but either they do not seem to capture specific characteristics of these systems in an appropriate way or they capture some particular features but not general ones. Alternative models coming from the area of population protocol models [2, 4] proposed protocols in order to represent sensor networks, supposing that the corresponding sensing devices are extremely limited mobile agents (a finite state machine) that interact only in pairs by means of a communication graph. These protocols can stably compute some properties of the communication graph and show the inclusion of the predicates computable by this model in the class NSPACE($m$) where $m$ is the number of edges of the communication graph.

We propose a very general formal model capturing more of the specific features of sensor networks than previously proposed models do. This model is flexible and suitable enough to capture non-homogeneous sensor fields (with different kinds of devices) and can be easily extended to asynchronous and dynamic (the communication graph changes in time either randomly or through an adversarial model) settings, although in this work we focus our attention to the study of *Static and Synchronous Sensor Fields*, the SSSF model. Our model includes a communication graph, an abstraction of the actual communication model, allowing independence of networking issues, but easier to specify. The model allow us to give a general and natural definition of sensing problems by means of input/output data streams as well as the corresponding measures of performance such as the latency, the message number or the message length among others.

We introduce first the Average Monitoring problem and, supposing that the sensor field has as many devices as input streams, we analyze it under several topologies. We use our defined measures of performance to obtain upper and lower bounds on the solutions of the problem and for concrete topologies we propose optimal algorithms to solve it.

Our proposed model can be seen as a non-uniform computational model in the sense that it is easy to introduce constraints to all or some of the devices of the sensor field and relate-it to classic complexity classes. By restricting their memory capacity to be constant, we show that the decisional version of the functions computed by this restricted SSSF belong to the class DSPACE($max(n, m)$) where $n$ is the number of nodes of the communication graph and $m$ its number of edges. Finally, by restricting the memory capacity to be constant and by allowing the inclusion in the network of non-sensing devices we show that there is a SSSF of polynomial size solving the monitoring problem for a property computable in polynomial time with polynomial latency.

This work is organized as follows. In Section 2 we introduce the SSSF its principal characteristics as well as the Average Monitoring and the Alerting problems. In Section 3 we study SSSFs solving the Average Monitoring problem. In particular we give lower bounds for the complexity measures in which the problem can be solved and propose optimal algorithms to solve it. Restricting the memory capacity of the devices in the SSSF (specifically supposing that it is constant), in Section 4 we study the solution of the Alerting problem and show the inclusion of the decisional version of the functions computed by a SSSF with constant memory devices in the class DSPACE($\max(n, m)$), where $n$ is the number of nodes of the communication graph and $m$ its number of edges. We then extend the studied networks to include nonsensing devices and we show in Section 5 that there is a constant memory SSSF of polynomial size and polynomial latency solving the monitoring problem for a polynomial time computable property. Finally, we give (Section 6) some conclusions and possibilities of future work.

## 2 Static Synchronous Sensor Field: the model

A *data stream* $w$ is a sequence of data items $w = w^1 w^2 \dots w^i \dots$ that can be infinite. For any $i \geq 1$ we denote by $w[i]$ the $i$-th element of $w$, i.e. $w[i] = w^i$. For any $i, j$, $1 \leq i \leq j$ we denote by $w[i, j]$ the subsequence of $w$ composed by all data items between the $i$-th and $j$-th positions, $w[i, j] = w^i \dots w^j$. For any $n \geq 1$, a $n$-data stream $\mathbf{w}$ is a $n$-tuple of data streams, $\mathbf{w} = (w_1, \dots, w_n)$ for $n \geq 1$. For any $i \geq 1$ we denote by $\mathbf{w}[i]$ the $n$-tuple composed by all the $i$-th elements of each data stream, $\mathbf{w}[i] = (w_1[i], \dots, w_n[i])$, for any $i \geq 1$. We denote by $\mathbf{w}[i, j]$ the $n$-tuple composed by the subsequences between the $i$-th and $j$-th positions of each data stream, $\mathbf{w}[i, j] = (w_1[i, j], \dots w_n[i, j])$ for any $i, j$ such that $1 \leq i \leq j$.

We use the standard graph notation. A *communication graph* is a directed graph $G = (N, E)$ where $N$ is the set of nodes and $E$ is the set of edges, $E \subseteq N \times N$. Let us consider that $N$ has $n$ nodes that are enumerated from 1 to $n$. Each node $k$ is associated to a device, let us say to device $k$. Each edge $(i, j) \in E$ specifies that device $i$ can send messages to device $j$ or what is the same, device $j$ can receive messages from node $i$. Given a device $k$ let us denote by $I(k) = \{i \mid (i, k) \in E\}$ the set of neighbors from which device $k$ can receive data items and by $O(k) = \{j \mid (k, j) \in E\}$ the set of neighbors to which device $k$ can send data. Let $in_k = |I(k)|$ and $out_k = |O(k)|$.

A *Static Synchronous Sensor Field* consists of a *set of devices* and a *communication graph.* The communication graph specifies how the devices communicate one to the other. For the moment and without loose of generality, we assume that all devices are sensing devices that can receive information from the environment and send information to the environment. Since the model we consider is static we assume that the edges are the same during all the computation time. Moreover each device executes its own process, communicates with their neighbors (devices associated to adjacent nodes) and also with the environment. All the devices work in a synchronous way, at each time step they receive data from their neighbors and from the environment, apply their own transition function changing in this way their actual configuration and send data to their neighbors and to the environment. Let us describe in detail the main components of the Static Synchronous Sensor Field.

Static Synchronous Sensor Field $\mathcal{F}$ (SSSF $\mathcal{F}$): Formally we define a Static Synchronous Sensor Field $\mathcal{F}$ by a tuple $\mathcal{F} = (N, E, U, T, X, (\delta_k, Q_k)_{k\in N})$ where

- $G_{\mathcal{F}} = (N, E)$ is the communication graph.
- $U$ is the alphabet of data items used to represent the input data streams that can be received from the environment.
- $T$ is the alphabet of items used to represent the output data streams that can be send to the environment.
- $X$ is the alphabet of items used to communicate each device to the others. Each $m \in X^*$ is called message or packet. $U, T \subseteq X$. We denote by data items the elements of alphabets $U$ and $T$ and by items the elements of the remaining alphabets.
- $(\delta_k, Q_k)$ defines for each device associated to a node $k \in N$ (device $k$) its set of local states and its transition function, respectively.

The *local computation of each device $k$ in $\mathcal{F}$* is defined by $(\delta_k, Q_k)$ and depends on the communication with its neighbors and with the environment. $Q_k$ is a (potentially infinite) set of local states and $\delta_k$ is a transition function. A state codifies the values of some local set of variables (ordinary program variables, message buffers, program counters ...) and all what is needed to describe completely the instantaneous configuration of the local computation. The transition function $\delta_k$ that depends on its local state $q_k \in Q_k$ as well as on:

- the items received by device $k$ from devices $i \in I(k)$,
- the data item that device $k$ receives as input from the environment,
- the items sent by device $k$ to devices $j \in O(k)$,
- and the item that device $k$ sends to the environment.

The transition function $\delta_k$ is defined as $\delta_k : Q_k \times (X^*)^{in_k} \times U \longrightarrow Q_k \times (X^*)^{out_k} \times T$. The meaning of $\delta_k(q_k, (x_{ik})_{i\in I(k)}, u_k) = (q'_k, (y_{kj})_{j\in O(k)}, v_k)$ is that if device $k$ of $\mathcal{F}$ is in its local state $q_k \in Q_k$, receives $x_{ik} \in X^*$ from each of its neighbors $i \in I(k)$, and receives the input data item $u_k \in U$ from the environment, then in one computation step device $k$ changes its local state to $q'_k \in Q_k$, sends $y_{kj} \in X^*$ to each of its neighbors $j \in O(k)$ and outputs $v_k \in T$ to the environment. In the case that device $k$ does not send or does not receive any value, we denote this 'no value' or 'it does not care' by the special symbol $\perp$. For any device $k$, let $q^0_k$ be the initial local state. For any $t \geq 1$, *the $t$-th computation step of device $k$* is described as follows: If the local state of device $k$ is $q^{t-1}_k$, and it receives $(x^t_{ik})_{i\in I(k)}$ from its input neighbors, $u^t_k$ from the environment and $\delta_k(q^{t-1}_k, (x^t_{ik})_{i\in I(k)}, u^t_k) = (q^t_k, (y^t_{kj})_{j\in O(k)}, v^t_k)$ then device $k$ changes its local state from $q^{t-1}_k$ to $q^t_k$, sends $(y^t_{kj})_{j\in O(k)}$ to its output neighbors and $v^t_k$ to the environment.

A *computation* of $\mathcal{F}$ is a sequence $\mathbf{c}^0, \mathbf{d}^1, \mathbf{c}^1, \mathbf{d}^2, \ldots, \mathbf{c}^{t-1}, \mathbf{d}^t, \mathbf{c}^t, \ldots$, eventually infinite, where $\mathbf{c}^0 = (q^0_k)_{k\in N}$ is the $n$-tuple of the initial local states of the $n$ devices, and for each $t \geq 1$, $\mathbf{c}^t = (q^t_k)_{k\in N}$ is the $n$-tuple of the local states after $t$ computation steps. $\mathbf{d}^t = (d^t_k)_{k\in N}$ represents the input/output data of the $t$-th computation step (i.e. the transition from time step $t-1$ to time step $t$). In particular, for device $k$ the input/output data of the $t$-th step is represented by $d^t_k = ((x^t_{ik})_{i\in I(k)}, u^t_k, (y^t_{kj})_{j\in O(k)}, v^t_k)$. Note that device $k$ receives $(x^t_{ik})_{i\in I(k)}$ from its neighbors, $x^t_{ik} = y^{t-1}_{ki}$ (assume that $x^1_{ki} = \lambda$) receives $u^t_k$ from the environment, changes its state from $q^{t-1}_k$ to $q^t_k$, sends $(y^t_{kj})_{j\in O(k)}$ to its neighbors and sends $v^t_k$ to the environment.

The *stream behavior of a computation* $\mathbf{c}^0, \mathbf{d}^1, \mathbf{c}^1, \mathbf{d}^2, \ldots, \mathbf{c}^{t-1}, \mathbf{d}^t, \mathbf{c}^t, \ldots$ *of* $\mathcal{F}$ is defined as $(\mathbf{u}, \mathbf{v})$ where $\mathbf{u} = (u_k)_{k \in N}$ is the tuple composed by the input data streams of each device $k$, $u_k = u_k^1 u_k^2 \ldots u_k^t \ldots$ and $\mathbf{v} = (v_k)_{k \in N}$ is the tuple composed by the output data stream of each device $v_k = v_k^1 v_k^2 \ldots v_k^t \ldots$ Notice that this information can be extracted from the computation $\mathbf{c}^0, \mathbf{d}^1, \mathbf{c}^1, \mathbf{d}^2, \ldots, \mathbf{c}^{t-1}, \mathbf{d}^t, \mathbf{c}^t, \ldots$. In this case we say that *the sensor field* $\mathcal{F}$ *computes* the tuple of output data streams $\mathbf{v} = (v_k)_{k \in N}$ given the the tuple of input data streams $\mathbf{u} = (u_k)_{k \in N}$ or what is the same, $\mathbf{v}[1, t]$ given $\mathbf{u}[1, t]$ for each $t \geq 1$.

Function computed by $\mathcal{F}$: We define the function $f_{\mathcal{F}}$ associated to the behavior $\mathcal{F}$ or equivalently, the function computed by $\mathcal{F}$ as follows:
Given any pair of tuples of data streams $\mathbf{u}$ and $\mathbf{v}$ and any $t \geq 1$, $f_{\mathcal{F}}(\mathbf{u}[1, t]) = \mathbf{v}[1, t]$ if and only if the sensor field $\mathcal{F}$ computes $\mathbf{v}[1, t]$ given $\mathbf{u}[1, t]$. A function $f$ *is computed by a sensor field* $\mathcal{F}$ if $f_{\mathcal{F}} = f$. A function $f$ *is computed by a sensor field* $\mathcal{F}$ *with latency* $d$ if for all (appropriate) tuple of data streams $\mathbf{u}$, and for all $t \geq 1$, $f_{\mathcal{F}}(\mathbf{u}[1, t + d])[t + d] = f(\mathbf{u}[1, t])[t]$.

Note that $\mathbf{u}$ and $\mathbf{v}$ have in general infinite length. In order to express formally the behavior of a SSSF we consider all the finite prefixes of the input stream $(\mathbf{u}[1, t])$ and those of the output stream $(\mathbf{v}[1, t])$. However, take into account that each sensor will output only one data item $(\mathbf{v}[t])$ per step. In general, computational problems that are susceptible of being solved by sensor fields can be stated in the following way:

Sensing Problem $\Pi$: Given a $n$-tuple of data streams $\mathbf{u} = (u_k)_{1 \leq k \leq n}$ for some $n \geq 1$, compute an $m$-tuple of data streams $\mathbf{v} = (v_k)_{1 \leq k \leq m}$ for some $m \leq n$ such that $R_\Pi(\mathbf{u}[1, t], \mathbf{v}[1, t])$ is satisfied for every $t \geq 1$. $R_\Pi$ is the relation that output data streams have to satisfy given the input data streams, i.e. the property that defines the problem.

A function $f$ between data streams is *consistent* with a relation $R$ when for every pair of data streams $\mathbf{u}$ and $\mathbf{v}$, and every $t \geq 1$, if $f(\mathbf{u}[1, t]) = \mathbf{v}[1, t]$ then $R(\mathbf{u}[1, t], \mathbf{v}[1, t])$.

Problem Solved by $\mathcal{F}$: A sensor field $\mathcal{F}$ *solves the problem* $\Pi$ if the function $f_{\mathcal{F}}$ computed by $\mathcal{F}$, is consistent with relation $R_\Pi$. A sensor field $\mathcal{F}$ *solves the problem* $\Pi$ *with latency* $d$, if the function computed by $\mathcal{F}$ with delay $d$ is consistent with relation $R_\Pi$.

Let us see two examples of sensing problems. First we consider the problem in which is needed to monitor continuously a wide area. This implies "sensing locally" and "informing locally" about an environmental phenomena. For instance it is is needed to know the average temperature of such area not in a particular location, but in every location in which the temperature is measured periodically.

Average Monitoring: Given $n$ data streams $(u_k)_{1 \leq k \leq n}$ for some $n \geq 1$, compute $n$ data streams $(v_k)_{1 \leq k \leq n}$ such that $v_k[t] = (u_1[t] + \cdots + u_k[t])/n$.

The second example we consider is related to "fire detection alarm". In this case it is desired to detect the situation in which there is a high risk of fire. One element to be measured is the level of smoke in the air of such area and if this level is higher than a certain value then the alert has to be activated. A specific device (number 1 for instance) acts as a master and outputs the result.

Alerting: Given $n$ data streams $(u_k)_{1 \leq k \leq n}$ for some $n \geq 1$, and threshold value $A$, device 1 has to compute a data stream $v_1$ such that

$$v_1[t] = \begin{cases} 1 & \text{if } \exists k : 1 \leq k \leq n : u_k[t] \geq A \\ \perp & \text{otherwise} \end{cases}$$

The computational resources needed to solve a problem of this kind will allow us to classify the computational complexity of such problem. The main resources used by a sensor field to solve a problem are the following. For each device and step of the device we can measure

- *Time of the device step.* The number of operations performed in the given step of the device. This is a rough estimation of the "physical time" needed to input data, receive information from other sensor, compute, send information and output data.
- *Space of the device step.* The space used by the device in such computation step.
- *Message Length of a device step.* The maximum number of data items of a message sent by the device in such computation time step.
- *Number of messages of a device step.* The maximum number of messages sent by the device in such device step.

Hence, we can define the following *complexity measures over any sensor field $\mathcal{F}$*:

- *Size*: The number of nodes or devices of the communication graph $G$, $|N| = n$.
- *Time*: The maximum time used by any device of $N$ in any of its steps.
- *Space*: The maximum space used by any device of $N$ in any of its steps.
- *MessageLength*: The maximum message length of any device of $N$ in any of its steps.
- *MessageNumber*: The maximum number of messages sent by any device of $N$ in any of its steps.

In general we will analyze these complexity measures with respect to the *Size* of the communication graph and taking $n = |N|$ we will denote by $\mathcal{T}(n)$ the *Time*, by $\mathcal{S}(n)$ the *Space*, by $\mathcal{L}(n)$ the *MessageLength* and by $\mathcal{M}(n)$ the *MessageNumber*.

## 3 SSSFs Solving the Average Monitoring Problem

We study the requirements of a SSSF for solving the Average Monitoring problem. It is divided into two parts. We start (Subsection 3.1) by giving lower bounds on the latency (Lemma 1), the *MessageNumber* (Lemma 2) and the *MessageLength* (Lemma 3), required by a SSSF for solving the Average Monitoring problem. Later we study (Subsection 3.2) some optimal algorithms.

### 3.1 Lower Bounds

In order to be able to state lower bounds, in this subsection we make two additional assumptions on the SSSF (i) the communication path for any pair of devices $i, j \in E$ is fixed and (ii) all the sent messages are formed only by tuples of input data items.

**Lemma 1.** *A SSSF $\mathcal{F}$ solving the Average Monitoring problem requires at least latency $\delta$, where $\delta$ is the diameter of its communication graph $G = (N, E)$.*

*Proof.* Let $\delta$ be the diameter of $G$, by definition there are at least two nodes (or devices) $i, j \in N$ such that the minimum distance between $i$ and $j$ is $\delta$, therefore, if at time $t$ node $i$ takes from the environment the input data $u_i[t]$ then, the node $j$ can't receive it in time $t' \leq t + \delta$. □

**Lemma 2.** *Let $\mathcal{F}$ be a SSSF solving the Average Monitoring problem with latency $\delta$, where $\delta$ is the diameter of its communication graph $G = (N, E)$. It holds that for any step $t > \delta$, there is a device sending at least $\beta(G)$ packets simultaneously, where $\beta(G) = \max_{k \in N} \beta(k)$ and $\beta(k)$ is the out-degree of node $k$ in the subgraph $G'$ of $G$ formed by $k$, all the nodes such that the communication paths from $k$ to each one of these nodes is of length $\delta$, all the nodes included in these communication paths and the corresponding edges.*

*Proof.* Any device $k$ has to send simultaneously a packet through at least each of its $\beta(k)$ outgoing edges in subgraph $G'$. Otherwise, by Lemma 1, it is not possible for $k$ to send less than $\beta(k)$ messages and reach all these devices with communication path of length $\delta$ with latency $\delta$. Taking the maximum among all the nodes in $G$ we get the bound $\beta(G)$. □

**Lemma 3.** *Let $\mathcal{F}$ be a SSSF solving the Average Monitoring problem with latency; $\delta$ equal to the diameter of its communication graph $G = (N, E)$. It holds that there is a step $t_0 > \delta$ such that for any step $t > t_0$, if $k_1, \ldots, k_{\delta+1}$ is the fixed communication path used by devices $k_1, \ldots, k_\delta$ to flood their data to $k_{\delta+1}$ then, there is a device receiving a message of at least $\delta$ data items.*

*Proof.* Let $t$ be any step such that $t > \delta$ and $k_1, \ldots, k_{\delta+1}$ the path determining the diameter and used by devices $k_1, \ldots, k_\delta$ to flood their data to device $k_{\delta+1}$. Let us suppose that device $k_\delta$ sends always less than $\delta$ data items. Then, the number $E_\delta$ of data items that arrive to $k_{\delta+1}$ from step 1 to step $t$ is such that $E_\delta \leq (t-\delta)(\delta-1)+1+2+\ldots+\delta-1+\delta-1 < (t-\delta)\delta+1+2+\ldots+\delta-1+2\delta-t$.

The number $R_{\delta+1}$ of data items that should have been arrived to device $k_{\delta+1}$ from step $\delta+1$ to step $t$ corresponding to devices $k_1, \ldots, k_{\delta+1}$ in order to output the correct average with latency $\delta$ is $R_{\delta+1} = \delta(t-\delta)$. Taking $t_0 = 1+2+\ldots+\delta-1+2\delta$ it is easy to see that $E_\delta < R_{\delta+1}$, implying that device $k_{\delta+1}$ has not receive enough data items to output the average, therefore contradicting the hypothesis that device $k_\delta$ can send less than $\delta$ data items by step. □

## 3.2 Algorithms

We consider optimal algorithms for solving the Average Monitoring problem in SSSFs. We first propose a generic algorithm of optimal latency for any SSSF with a strongly connected communication graph (Lemma 4). In general, this algorithm is not optimal in *Space*, *MessageLength* and *MessageNumber*, since in order to guaranteed correctness it has to send and storage more information than required. However, when the topology of the communication graph is known in advance, it is possible to improve the generic algorithm and obtain optimal algorithms for specific topologies (Lemma 5). In what follows, we assume that every device in the SSSF is aware of the total number of devices.

**Lemma 4.** *Let $G = (N, E)$ be a strongly connected communication graph with $|N| = n$, diameter $\delta$, in-degree $in_G$ and out-degree $out_G$. There is a SSSF $\mathcal{F}$ with communication graph $G$ solving the Average Monitoring problem with latency $\delta$, $\mathcal{T}(n) = O(in_G \times n \times \delta)$, $\mathcal{S}(n) = O(n \times \delta)$, $\mathcal{S}(n) = \Omega(\delta)$, $\mathcal{L}(n) = O(n \log n \times \delta)$ and $\mathcal{M}(n) = out_G$.*

*Proof Sketch.* We consider a flooding algorithm (see Algorithm 3.2) in which each item is formed by a data item together with the producer sensor id and its time stamp modulo $\delta$. An item has lifetime $\delta$. Each sensor receives a set of items form each in-neighbor. Composes the union of the incoming sets and incorporates the actual lecture. This data is used to actualize an internal matrix. All the alive items gathered by the sensor that have not been forwarded are sent to the out-neighbors. Due to lack of space the details of the implementation and the analysis are given in Appendix A. □

**Algorithms with optimal latency.** When the topology of the communication graph is known it is possible to specialized and improve Algorithm 3.2 to obtain optimal algorithms, as next lemma states. Next lemma follows by Lemmas 1, 2, 3 and 4 since they show that there is no way to reduce neither the *MessageNumber* nor the *MessageLength* of the algorithms presented in Appendix B.

**Algorithm 1** SSSF solving the Average Monitoring problem.

```
algorithm Generic Algorithm for Sensor $k$
    ▷ Initially
    $M[1 \ldots \delta] \times [1 \ldots n] = (\perp)_{\delta \times n}$
    ▷ Step
    // receive
    for $i \in I(k)$ { receive $S_i$ from incoming neighbors ; $S = S \cup S_i$ }
    take input data $u$
    $S = S \cup u$
    // compute
    for $p \in \{1, \ldots, \delta\}, q \in \{1, \ldots, n\}$ $\{(M, S) =$ update$(M, S)\}$
    compute average $v = (M[\delta][1] + \ldots + M[\delta][n])/n$
    // send
    for $j \in O(k)$ flood $S$
    output v
end algorithm
```

**Lemma 5.** *The Average Monitoring problem can be solved with optimal latency, MessageNumber and MessageLength by a SSSFs whose communication graph are bidirectional cliques, oriented rings, balanced binary trees or toroidal grids, respectively.*

**Improving the message length.** By data aggregation and allowing a larger latency, it is possible to improve the *MessageLength*. In this case, messages are no longer tuples of data items. Next Lemma follows by analyzing the referred algorithm. The approach is based in automata networks [5, 12], see details in Appendix B.2.

**Lemma 6.** *The Average Monitoring problem can be solved with latency $2n - 1$, $\mathcal{T}(n) = \Theta(n)$, $\mathcal{S}(n) = \Theta(n \log n)$, $\mathcal{L}(n) = \Theta(\log n)$ and $\mathcal{M}(n) = \Theta(1)$ by a SSSF in which the communication network is an oriented ring.*

## 4 SSSFs of Devices with Constant Memory Capacity

Up to now we have not considered the possible memory restrictions of the *tiny* devices involved in a SSSF, but in applications, devices can have limited memory. The SSSF model can be adapted to this fact, therefore we can consider devices with constant memory capacity. To this end, we also assume that each device has a buffer of constant size to store the data received from its neighbors and that flood or send data to them can be performed also in constant time and space. As we show in the following, the Alerting problem can be solved in such SSSF. At one step, there is a device that senses a value greater than the threshold, it has to notify one specific device or all the other devices. Device 1 outputs the corresponding data stream.

**Lemma 7.** *Let $G = (N, E)$ be a strongly connected communication graph with $|N| = n$, diameter $\delta$, out-degree $out_G$ and such that any device $i \in N$ is connected to device 1. There is a SSSF $\mathcal{F}$ with communication graph $G$ solving the Alerting problem with $\mathcal{T}(n) = \Theta(1)$, $\mathcal{S}(n) = \Theta(1)$, $\mathcal{L}(n) = \Theta(1)$ and $\mathcal{M}(n) = O(out_G)$.*

*Proof.* The proof consists on giving a generic algorithm (Algorithm 4) that solves the Alerting problem under the conditions stated by this lemma. Roughly speaking, at any step $t$, the algorithm for each device receives at most a constant number of items from its incoming neighbors and a data item from the environment, each device compares the data item $u$ from the environment against a threshold value $A$. If there is a 1 value coming from the neighbors or $u > A$ then, depending on whether the device is number 1 or not, it sends a 1 value to the environment or floods the received information to its outcoming neighbors. □

Regarding the latency it is at most $\delta$ (the diameter of $G$), since the worst case occurs when the distance between the device that sense the input data $d > A$ and device 1 is $\delta$.

**Algorithm 2** SSSF solving the Alerting problem.

```
algorithm Algorithm for a Sensor
    ▷ Initially
    $alert = 0$
    ▷ Step
    // receive
     receive $data$
     take input data $u$
    // compute
    if ($u > A$ or $data = 1$) $alert = 1$
    // send
    if ($alert$) {if (output node) output 1   else  flood 1}
end algorithm
```

In general, we can say that by restricting the memory capacity of each device to be a constant w.r.t. the total number of devices then the kind of problems solved by these SSSFs are not more difficult than the ones in DSPACE($O(m)$) where $m$ is the maximum between the number of nodes and the number of edges of the communication graph. In order to prove it formally let us define the decisional version of $f_{\mathcal{F}}$.

Language associated to $\mathcal{F}$: Let $\mathcal{F}$ be a SSSF and let $f_{\mathcal{F}}$ be the function computed by $\mathcal{F}$. We define the language associated to the behavior of $\mathcal{F}$, denoted by $L(\mathcal{F})$ as follows:

$$L(\mathcal{F}) = \{\langle \mathbf{u}[1], \mathbf{v}[1], \ldots, \mathbf{u}[t], \mathbf{v}[t]\rangle \mid t \geq 1 \text{ and } f_{\mathcal{F}}(\mathbf{u}[1,t]) = \mathbf{v}[1,t]\}.$$

**Theorem 1.** *Let $\mathcal{F}$ be a SSSF with communication graph $G = (N, E)$ and all of its devices of constant space. Let $m = \max\{|N|, |E|\}$. Then the language $L(\mathcal{F}) \in DSPACE(O(m))$.*

*Proof.* We are going to present a deterministic Turing machine $M$ that decides the language $L(\mathcal{F})$ in space $O(m)$. Since each device $k$ has constant memory capacity, then the size of $Q_k$ is also bounded by a constant as well as it is the number of items composing each sent or received packet. Recall that the behavior of $\mathcal{F}$ is described by a sequence $\mathbf{c}^0, \mathbf{d}^1, \mathbf{c}^1, \mathbf{d}^2, \ldots, \mathbf{c}^{i-1}, \mathbf{d}^i, \ldots$, eventually infinite, where $\mathbf{c}^0 = (q_k^0)_{k\in N}$ is the $n$-tuple of the initial local states of the $n$ devices, and for each $i \geq 1$, $\mathbf{c}^i = (q_k^i)_{k\in N}$ is the $n$-tuple of the local states after $i$ computation steps. $\mathbf{d}^i = (d_k^i)_{k\in N}$ represents the input/output data and the sent/received messages of each device $k$ in the transition from time step $i-1$ to time step $i$. The Turing machine $M$ on any input $\langle \mathbf{u}[1], \mathbf{v}[1], \ldots \mathbf{u}[t], \mathbf{v}[t]\rangle$ will compute such a sequence in the following way:

1. Initially $M$ computes the initial configuration $\mathbf{c}^0$ and suppose that devices have neither received a message nor an input data item.
2. Simulates the $i$-th computation step computing $(\mathbf{c}^i, \mathbf{d}^{i+1})$ from $(\mathbf{c}^{i-1}, \mathbf{d}^i)$. In order to do this, for each device $k$ $M$ applies $\delta_k$ considering that the input data item is given by $u_k[i]$ and verifies that the output data item is $v_k[i]$. If it is the case then $M$ considers the next computation step $i+1$; otherwise, $M$ rejects.
3. Once $M$ has consumed all its input word, it accepts.

$M$ needs space $O(m)$ to decide $L(\mathcal{F})$. Note that in the part 3 of the simulation $M$ only needs to store the messages send/received, one for each edge of the communication graph, and it also needs the current state for each one of the nodes of the graph. □

In [4] it is shown that all predicates stably computed in the model of Mediated Population Protocols are in the class of NSPACE($O(m)$). In this case the nondeterminism is required to verify that there exists a stable configuration reachable from the initial configuration.

## 5 Trading space/time for size

In this section we analyze SSSFs with an additional amount of nodes in the communication graph in which the attached devices participate in the computation but do not play any active role in sensing. In such a network we have a communication graph with $S$ nodes and we want to solve a problem that involves only $n < S$ input data streams. We study two different variants of these SSSFs. On the one hand, we consider devices with different space capacities going from constant to logarithmic on the number of sensing devices and we show that (Lemma 8) for a fixed topology there is a SSSF that solves the Average Monitoring problem with optimal latency constant time and better *MessageLength* and *MessageNumber*. Thus improving the sampling rate of the network. On the other hand we show (Lemma 9) that such extra capabilities allow to design networks that solve a generalization of the Monitoring problem with constant space.

**Constant time***:* In a balanced communication tree we suppose that there are $n$ sensing devices placed on the leaves of a balanced binary tree, edges to leaves are replaced by paths in such a way that all the leaves are at the same distance to the root. Thus, the tree has depth $O(\log n)$ and $n$ leaves. Next Lemma follows from analyzing Algorithm 6, see Appendix C.

**Lemma 8.** *Let $G = (N, E)$ be a balanced communication tree whose $n$ leaves are sensing devices of constant space and whose internal nodes are non-sensing devices of $\log n$ space. There is a SSSF $\mathcal{F}$ with communication graph $G$ solving the Average Monitoring problem with optimal latency, $\mathcal{L}(n) = O(\log n)$ and $\mathcal{T}(n) = \mathcal{M}(n) = O(1)$.*

In the algorithm described above the nodes have different levels of internal memory, ranging from constant at the leaves to $\log n$ in the upper levels. The following result shows that by increasing the number of auxiliary nodes we can solve sensing problems with constant memory components.

**Constant space devices***:* Let $\mathcal{P}$ be a property defined on $U^n$. We consider the following sensing problem:

Monitoring Problem for property $\mathcal{P}$: Given a $n$-tuple of data streams $u = (u_k)_{1\leq k\leq n}$ for some $n \geq 1$, compute an $n$-tuple of data streams $v = (v_k)_{1\leq k\leq m}$ such that $\mathbf{v}[t] = \mathcal{P}(\mathbf{u}[t])$ for every $t \geq 1$.

**Lemma 9.** *Let $\mathcal{P}$ be a property defined on $U^n$ computable in polynomial time. There is a constant space* SSSF *that solves the associated sensing problem in polynomial size and latency (with respect to $n$).*

*Proof.* Recall that any polynomially computable function can be solved by a uniform family of circuits with polynomial size. Furthermore those circuits can be assumed to be layered and to have bounded fan in and fan out by adding propagator gates. The communication network is formed by the circuit with sensors attached to the corresponding inputs together with a communication tree that flows the result to the inputs. As the circuit is layered we can guarantee the pipelined flow of partial computations. The total delay is exactly the circuit's depth plus the tree depth, and thus polynomial in $n$. □

## 6 Conclusions and Future Work

We have proposed a model for networks of artifacts and analyze the complexity of some sensing problems on them. This model abstracts the main characteristics of the problems that are expected to be solved on a network with sensors. In parallel we have introduced the corresponding problem type and the parameters that take part in the complexity of a protocol. The model has allowed us to analyze the complexity of some problems providing optimal protocols with respect to some parameters.

The analysis shows, as expected, that different sensing problems will require different sensor capabilities for storing data and message size. Our algorithms for the Average Monitoring problem can be easily adapted to solve the monitoring problem associated to other aggregation functions like: maximum, minimum, addition, median, etc, the complexity of the above algorithms differ depending on the structural properties of the aggregation function (see [10] for a classification) and the size of the aggregated data.

In this work we have focused on the memory allowed to each device as this reveals how tiny a network device must be to solve a problem. Taking a holistic function as, for example, the median, its computation require to have access to all the components, therefore any SSSF solving the corresponding monitoring problem requires space $\Omega(n)$. For the case of the Average Monitoring we have shown the existence of SSSF that solve the problem using $O(\log n)$ space, of course the problem can be solved with constant space on an adequate SSSF with polynomially many additional computing units. Finally we have shown that the Alerting problem can be solved with constant space in any network. There is a clear trade-off between the internal memory allowed to each device and the number of additional computing units in the network as show in Lemma 9. It will be of interest to characterize those sensing problems that can be solved with logarithmic or constant space with no (or a small number of) additional nodes.

Although in the present paper the communication network has been assumed to be fixed, the model is simple and flexible enough to incorporate a dynamic communication graph in which the connection at each time step are different. In this way we could incorporate networks with failures or networks with mobile nodes. It is also easy to adapt the model to a non-synchronous scenario in the usual way.

On the other hand the hypothesis of fixed communication graphs models the idea of maintaining a virtual fixed topology, this topology will be maintained until the network task changes. In this situation the communication graph will be perceived as the same graph, although the devices taking care of one node might change over time. Our complexity analysis on fixed topologies should be combined with a study of the conditions that guarantee the existence, creation an maintenance of the virtual topology.

All through the paper we have not considered the energy consumption as a performance measure. For making an energy analysis we should have to incorporate a particular energy model to the sensor field. The performance measures taken in this paper proportionate the basic ingredients in such models, as energy is usually computed as a function of number of messages and message size. We expect that tuning such parameters will lead to protocols with lower energy consumption. It is of interest (and topic of future research) to consider energy models in which each link in the communication graph has different weights (or set of weights) representing the constants in the function that determines the cost of sending a message along the link. Observe that those considerations can be easily incorporated to our sensor field model, although the energy analysis might require additional tools and techniques.

## References

1. Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A Survey on Sensor Networks. *IEEE Comunnications Magazine*, 2002.
2. D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed computing*, 2006.
3. Jean Berstel. Quelques applications des reseaux d'automates. Thèse de 3ème Cycle, juin 1967.
4. Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. Mediated Population Protocols. In *Proceedings of the ICALP 09, to appear*, 2009.
5. Christian Choffrut, editor. *Automata Networks, LITP Spring School on Theoretical Computer Science, Argelès-Village, France, May 12-16, 1986, Proceedings*, volume 316 of *Lecture Notes in Computer Science*. Springer, 1988.
6. Deborah Estrin, David Culler, Kris Pister, and Gaurav Sukatme. Connecting the Physical World with Pervasive Networks. *Pervasive Computing*, 2002.
7. Thomas A. Henzinger. The Theory of Hybrid Automata. In *Proceedings of the 11th Annual IEEE Symposium (LICS 96)*, 1996.
8. C. A. R. Hoare. A calculus of total correctness for communicating processes. *Sci. Comput. Program.*, 1(1-2):49–72, 1981.
9. Teijiro Isokawa, Ferdinand Peper, and Masahiko Mitsui. Computing by Swarm Networks. In *ACRI 2008, LNCS 5191*, volume 5191 of *LNCS*, pages 50–59, 2008.
10. Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks.* John Wiley & Sons Ltd, 2005.
11. Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O Automata Revisited. In Springer-Verlag, editor, *Hybrid Systems: Computation and Control: 4th International Workshop (HSCC'01)*, volume 2034, pages 403–417, 2001.
12. Jacques Mazoyer. An overview of the firing squad synchronization problem. In *Automata Networks*, Lecture Notes in Computer Science, pages 82–94. Springer, 1986.
13. D. J. Nagel. Wireless Sensor Systems and Networks: technologies, applications, implications and impact. *http://intranet.daiict.ac.in/˜ ranjan/isn2005/papers/APP/wireless.pdf*, 2005.
14. David Peleg. *Distributed Computing. A Locality-Sensitive Approach*, chapter 2. SIAM Monographs on Discrete Mathematics and Applications, 2003.
15. Maurice Henri ter Beek. *Team Automata. A Formal Approach to Modeling of Collaboration Between System Components.* PhD thesis, Leiden University, 2003.
16. Sameer Tilak, Nael B. Abu-Ghazaleh, and Wendi Heinzelman. A Taxonomy of Wireless Micro-Sensor Network Models. *Mobile Computing and Communications Review*, 2003.
17. M. Vinyals, J.A. Roderiguez-Aguilar, and J. Cerquides. A Survey on Sensor Networks from a Multi-Agent Perspective. In *Proc. of AAMAS 2008*, 2008.
18. N. Xu. A Survey of Sensor Network Applications. *http://courses.cs.tamu.edu/rabi/cpsc617/resources/sensor-%20nw-survey.pdf*, 2008.

## A Proof of Lemma 4.

*Proof.* (of Lemma 4) The proof consists on giving a generic algorithm (Algorithm 3.2) that solves the Average Monitoring problem under the conditions stated by this lemma.

Roughly speaking, at any step $t > \delta$, the algorithm for each device receives information from its incoming neighbors and from the environment, updates its stored information, computes an average, floods the received information to its outcoming neighbors and sends the average to the environment.

For each device, the algorithm uses a matrix $M_{\delta \times n}$ to store the input data of all the devices in the SSSF at $\delta$ consecutive steps in such a way that $M[i][j]$ corresponds to the input data received from the environment by device $j$ at step $i$. Initially, all the positions of $M$ are initialize to a don't care symbol ($\perp$) and incrementally the positions are filled with the corresponding values. In stationary, at every step $t > \delta$, every device fills its last ($\delta$) line and it's able to produce the average corresponding to step $t - \delta$.

The filling process of the positions of $M$ is achieved through the flow of messages among devices. Let the tuple $\langle u, k, t \mod \delta \rangle$ denote the input data $u$ received from the environment by sensor $k$ at step $t$ ($u_k^t$ is the value of $u$ in $k$ at step $t$). The message forwarded by sensor $i \in I(k)$ to sensor $k$ at the beginning of any step $t > \delta$ encodes the following set.

$$\{\langle u_i^{t-1}, i, (t-1) \rangle\} \cup \{\langle u_j^{t-\ell}, j, (t-\ell) \rangle \mid \exists \text{ path from } j \text{ to } i \text{ of length } \ , \ell < \delta\}$$

When this packet is received, the underlying subset is recovered. This is, the value $u$ belonging to the item$\langle u, j, t - \ell$ will be stored into $M[\ell][j]$. At the same step $t$, sensor $k$ takes the value $u_k^t$ and stores it at the corresponding position of $M$. Therefore, for $t > \delta$, it is easy to see that matrix $M$ contains $u_1[t-\delta], \ldots, u_n[t-\delta]$ at the right positions, which is the required information to compute the average to be send to the environment. Defining $avg(t-\delta) = (u_1[t-\delta] + \cdots + u_n[t-\delta])/n$ we get $v_k[t] = avg(t-\delta)$. This is the output value going to the environment at the end of the step. Finally we consider the packet to be forwarded to any $j \in O(k)$ (all the $j$ will receive the same packet). To flood the correct information, sensor $k$ collects all the incoming packets and computes the union of the corresponding encoded subsets, it adds the new item $\langle u, k, t \rangle$ and deletes the packets with time stamp $t - \delta$ (they are no longer necessary). An encoding of the resulting set is sent to all its neighbors.

The tuples have the structure $\langle u, k, t \rangle$ using the step $t$ as a time stamp. The algorithm can be coded without loss of generality replacing $t$ by $t \bmod \delta$.

Let us consider the complexity measures. At each step, one packet encoding a set as the ones described previously is flooded from every device to all its neighbors in $O(k)$, therefore the $\mathcal{M}(n) = O(out_G)$. A given packet can contain information about each node at different steps, this give us a $n \times \delta$ maximal number of tuples of the form $\langle u, j, t \rangle$ in each set. Since an encoding of such a tuple requires $O(\log n)$ bits this give us that $\mathcal{L}(n) = O(n \log n \times \delta)$. At each time step, a device receives, unfolds and stores all the incoming data into the corresponding positions of matrix $M$, this is achieved in $\mathcal{T}(n) = O(in_G \times n \times \delta)$ and $\mathcal{S}(n) = n \times \delta)$. Finally, at step $t = \delta + 1$ all the sensors output $v[1]$, at $t = \delta + 2$ all the sensors output $v[2]$) and successively, which gives a latency of $\delta$ steps. $\square$

## B SSSFs Specific Algorithms for the Average Monitoring Problem

### B.1 Algorithms with optimal latency.

The following algorithms are direct applications of Algorithm 3.2 to the given topology.

*Bidirectional clique:* when the communication graph is a clique of $n$ devices, the Average Monitoring problem can be solved in constant latency with $\mathcal{T}(n) = \Theta(n)$, $\mathcal{S}(n) = \Theta(n)$, $\mathcal{L}(n) = \Theta(1)$ and $\mathcal{M}(n) = n - 1$.

*Oriented ring:* we now consider a network with $n$ sensing devices connected into an oriented ring, where sensor $k$ is connected to sensor $(k + 1) \mod n$. In this case the Average Monitoring problem can be solved with an optimal latency of $n - 1$ steps by means of Algorithm B.1 that works as follows. At each step, device $k$ take a measure, receives $n - 1$ data items from its predecessor corresponding to measures taken by device at distance $i$ (for $i \in \{1, \ldots, n-1\}$) at the $i$-th previous step, computes and outputs the average of the $(n - 1)$th step and sends to its successor the $n - 1$ corresponding measures (the $n - 2$ received from its predecessor plus its new taken one). Hence, the complexity measures are as follows, $\mathcal{T}(n) = \Theta(n)$, $\mathcal{S}(n) = \Theta(n)$, $\mathcal{L}(n) = n - 1$ and $\mathcal{M}(n) = 1$.

**Algorithm 3** Algorithm for Minimun Latency in an Oriented Ring

```
algorithm for sensor $k$
    ▷ Initially
    $A[1, \ldots, n] = [0.0, \ldots, 0.0]$
    $X[1, \ldots, n-1] = [0.0, \ldots, 0.0]$
    ▷ Step
    // receive
    receive $X$
    take input data $u$
    // compute
    for $i \in \{2, \ldots, n\} A[i] = A[i-1] + X[i]$ end for
    $A[1] = u$
    for $i \in \{2, \ldots, n-1\}$ $X[i] = X[i-1]$
    $X[1] = u$
    $v = A[n]/n$
    // send and output
    send $X$
    output $v$
```

*Balanced binary tree:* let us consider a SSSF of $n = 2^h - 1$ devices connected into a balanced binary tree. In general terms each device is connected to its two children (except for the leaves) and to its parent (except for the root) and the algorithm works as follows. At each step, each device takes a measure from the environment, receives information from its two children (except for the leaves) that resends to its parent together with its own measure at previous step and receives information from its parent (except for the root) that outputs to the environment (after some required delay) and resends to its two children. The root accumulates all the values, computes the average and sends it down the tree.

**Algorithm 4** SSSF Algorithm for Balanced Binary Trees

```
algorithm for sensor 1 (the root)
    ▷ Initially
    const $h = \log_2(n+1) - 1$
    $up[1,\dots,h] = [0.0,\dots,0.0]$
    $down[1,\dots,h] = [0.0,\dots,0.0]$
    ▷ Step
    // receive
    take input data $u$
    receive $X', X''$
    // compute
    $X = X' + X'' + up[h]$
    for $i \in \{1,\dots,h-1\}$  $up[i+1] = up[i]$
    $up[1] = u$
    $Y = X$
    $v = down[h]/n$
    for $i \in \{1,\dots,h_k-1\}$  $down[i+1] = down[i]$
    $down[1] = Y$
    // send and output
    output $v$
    send $Y$ to children

algorithm for sensor $k$, with $1 < k <= \lfloor n/2 \rfloor$ (the internal nodes)
    ▷ Initially
    const $h = \log_2(n+1) - \lfloor \log_2 k \rfloor - 1$
    $up[1,\dots,h] = [0.0,\dots,0.0]$
    $down[1,\dots,h] = [0.0,\dots,0.0]$
    ▷ Step
    // receive
    take input data $u$
    receive $Y', X', X''$
    // compute
    $X = X' + X'' + up[h]$
    for $i \in \{1,\dots,h-1\}$  $up[i+1] = up[i]$
    $up[1] = u$
    $Y = Y'$
    $v = down[h]/n$
    for $i \in \{1,\dots,h_k-1\}$  $down[i+1] = down[i]$
    $down[1] = Y$
    // send and output
    output $v$
    send $X$ to parent
    send $Y$ to children

algorithm for sensor $k$, with $k \geq \lfloor n/2 \rfloor$ (the leaves)
    ▷ Step
    // receive
    take input data $u$
    receive $Y$
    // compute
    $X = u$
    $v = Y/n$
    // send and output
    output $v$
    send $X$ to parent
```

In more detail, device $k$ is connected to its parent (except for the root that corresponds to $k = 1$), device numbered $\lfloor \frac{k}{2} \rfloor$, and to its two children (except for the leaves that correspond to $k > \lfloor \frac{n}{2} \rfloor$), devices numbered $2k$ and $2k + 1$.

At each step, device $k$ takes a measure from the environment, receives information from and sends information to devices $\lfloor \frac{k}{2} \rfloor$ (except for $k = 1$), $2k$ and $2k+1$ (except for $k > \lfloor \frac{n}{2} \rfloor$) and outputs information received from devices $\lfloor \frac{k}{2} \rfloor$ (except for $k = 1$ that outputs information coming from 2 and 3, for $n \geq 3$).

The number $n = 2^h - 1$ of sensors is known. For each sensor $k$ we consider two vectors *up* and *down*, each one of size $h_k = h - \lfloor \log_2 k \rfloor - 1$, that store the information received from its children and its parent respectively. Moreover, for each sensor $k$ we consider a variable $X_k$ that stores the value to be send to its parent and a variable $Y_k$ that stores the value to be send to its children.

At step $h-1$ (the height of the tree) the first meaningful average is computed. At this step, the root (sensor 1) receives from its children (sensors 2 and 3) the values $X_2$ and $X_3$, (for $n \geq 3$), each containing the sum of the measures taken at time 1 by the devices in subtrees rooted at devices 2 and 3 and, before ending the step, the root sums up these two values to its own taken measure at time 1, stored in $up[h_1]$. The root sends this value to its children and stores it in $down[1]$ in order to retard its output the $h - 1$ steps (a path down the tree) required for the leaves to receive and output this information.

The Average Monitoring problem can be solved in a balanced binary tree topology of $n = 2^h - 1$ devices with Algorithm 4 with optimal latency $2h - 2$. Any device labelled $k$ requires time and space $\Theta(h_k = h - \lfloor \log_2 k \rfloor)$. Therefore, $\mathcal{T}(n) = \Theta(\log n)$, $\mathcal{S}(n) = \Theta(\log n)$, $\mathcal{L}(n) = \Theta(\log n)$ and $\mathcal{M}(n) = \Theta(1)$.

*Toroidal grid:* the communication graph in this case consists of $n^2$ sensors arranged in a $n \times n$ squared matrix. In this matrix each line of sensors form a directed ring as well as does each column. It is possible to solve the Average Monitoring problem with optimal latency $2n - 2$ in this sensor field by applying twice the previous minimum latency algorithm for oriented rings.

Firstly, the algorithm is executed in each line, so, after $n - 1$ steps, all the sensors in the same line have the total sum of the measures taken in their line at the $(n - 1)$th step. Secondly, the algorithm is executed in each column and, after $n-1$ additional steps, all the sensors have the total sum of measures in the matrix (adding the sum of the line to the sum of the column). Therefore, the Average Monitoring problem is solved by applying the oriented ring algorithm $2n$ times. Therefore, $\mathcal{T}(n) = \Theta(n)$, $\mathcal{S}(n) = \Theta(n)$, $\mathcal{L}(n) = \Theta(n-1)$ and $\mathcal{M}(n) = 1$.

## B.2 Improving the $MessageLength$

By data aggregation and allowing a larger latency, it is possible to improve the $MessageLength$ as we show below.

*Oriented ring:* considering again a communication graph of $n$ devices under an oriented ring topology, Algorithm 5 solves the problem with constant $MessageNumber$ and logarithmic $MessageLength$.

**Algorithm 5** Algorithms for Sending Minimum Messages in a Ring

```
algorithm for sensor 1
    ▷ Initially
    $D[1] = [0.0], A[1, \dots, n-1] = [0.0, \dots, 0.0]$
    ▷ Step
    // receive
    receive $avg$
    take input data $u$
    // compute
    $D[1] = u$
    $sum = D[1]$
    $A[1] = avg$
    // send and output
    send $sum$
    output $A[n-1]$
end algorithm

algorithm for sensor $k$
    ▷ Initially
    $D[1, \dots, k] = [0.0, \dots, 0.0], A[1, \dots, n-k] = [0.0, \dots, 0.0]$
    ▷ Step
    // receive
    receive $sum, avg$
    take input data $u$
    // compute
    for $i \in \{2, \dots, k\} D[i] = D[i-1]$ end for
    $D[1] = u$
    $sum = sum + D[k]$
    for $i \in \{2, \dots, n-k\} A[i] = A[i-1]$ end for
    $A[1] = avg$
    // send and output
    if $k = n-1$ then send $sum$ end if
    if $k \neq n-1$ then send $sum, A[1]$ end if
    output $A[n-k]$
end algorithm

algorithm for sensor $n$
    ▷ Initially
    $D[1, \dots, n] = [0.0, \dots, 0.0], A[1, \dots, n] = [0.0, \dots, 0.0]$
    ▷ Step
    // receive
    receive $sum$
    take input data $u$
    // compute
    for $i \in \{2, \dots, n\} D[i] = D[i-1]$ end for
    $D[1] = u$
    $sum = sum + D[n]$
    $A[1] = sum/n$
    // send and output
    send $A[1]$
    output $A[1]$
end algorithm
```

Informally, one device acts as a leader (say device 1) and another device (say device $n$) computes, collects and distributes the averages. It is assumed that each sensor knows the number of sensors $n$ and its position inside the ring. As before, all the nodes start reading from the environment at the same time.

Device 1 takes and floods its first taken measure to device 2 at step 1. At step 2, device 2 receives the first taken measure of device 1, adds it to its own taken measure at step 1 and forwards the sum to device 3. Eventually, sensor $n$ receives the sum of measures taken by devices $1, 2, \ldots, n-1$ at step 1, adds it to its own taken measure at step 1 and computes the first meaningful average and forwards it to other devices.

A SSSF in which the communication network is an oriented ring and device $k$ executes Algorithm 5 solves the Average Monitoring with latency $2n-1$, $\mathcal{T}(n) = \Theta(n)$, $\mathcal{S}(n) = \Theta(n \log n)$, $\mathcal{L}(n) = \Theta(\log n)$ and $\mathcal{M}(n) = \Theta(1)$.

## C Proof of Lemma 8

*Proof.* (of Lemma 8) Let us consider a SSSF whose communication graph is a balanced communication tree with $n$ leaves and $S$ nodes, and height $h = O(\log n)$, recall that all the leaves are at distance $h$ from the root. In general terms each device is connected to its children (one or two) (except for the leaves) and to its parent (except for the root) and the algorithm works as follows. The fact that all the distances from the root to the leaves are equal guarantee that all the messages aggregating data produced at the leaves at some time step will arrive to a common predecessor at the same timestep. The SSSF keeps a flow from the leaves to the root in parallel with a flow from the root to the leaves. Message sent by a node $k$ two its parent contains two values, $< Av_k, n_k >$, containing the average and the number of the measured data at the leaves in the corresponding subtree. The message forwarded from the root contains the monitored average of the complete field that is distributed to the node leaves. The leaves output all the items received from their parent.

The number $n$ of sensing devices is only learned by the root of the communication tree. At step $h-1$ (the height of the tree) the first meaningful average is computed and starts to go downwards to the leaves, thus giving a optimal latency for the algorithm.

The Average Monitoring problem can be solved in a balanced communication tree topology with $n$ leaves with Algorithm 6 with optimal latency $2h-2$. Any internal device requires space $\Omega(\log_n)$ and leaves only constant space. Furthermore, $\mathcal{T}(n) = \Theta(1)$, $\mathcal{S}(n) = \Theta(\log n)$, $\mathcal{L}(n) = \Theta(\log n)$ and $\mathcal{M}(n) = \Theta(1)$. $\square$

Observe that each node in the network requires to know whether it is the root or not and whether the number of children is one, two or zero. This information can be stored in constant space.

It is worth to note that this algorithm allows additional improvements, for example the message size can be reduced after the first data is sent to the parent. Making that any internal device stores the number of leaves in each children subtree, subsequent messages can be discarded from this component, this will decrease the totak energy consumption in later steps, although not the upperbound on the message length.

**Algorithm 6** SSSF Algorithm for Balanced Communication Trees

```
algorithm for the root
    ▷ Step
    receive $A1, n1, A2, n2$ from children  $Av$ from parent
    // compute
    $n = n1 + n2$
    $A = A1 * (n1/n) + A2 * (n2/n)$
    // send and output
    send $A$ to children

algorithm for internal nodes with two children
    ▷ Step
    // receive
    receive $A1, n1, A2, n2$ from children  $Av$ from parent
    // compute
    $n = n1 + n2$
    $A = A1 * (n1/n) + A2 * (n2/n)$
    // send and output
    send $A, n$ to parent
    send $Av$ to children

algorithm for internal nodes with one children
    ▷ Step
    // receive
    receive $A1, n1$ from children  $Av$ from parent
    // send and output
    send $A1, n1$ to parent
    send $Av$ to children

algorithm for sensor at the leaves
    ▷ Step
    // receive
    take input data $u$
    receive $Av$ from parent
    // send and output
    output $Av$
    send $u, 1$ to parent
```