TAN: a Distributed Algorithm for Dynamic Task Assignment in WSNs

Virginia Pilloni*, Pirabakaran Navaratnam†, Serdar Vural†, Luigi Atzori*, Rahim Tafazolli†
*DIEE, University of Cagliari, Italy {*virginia.pilloni,l.atzori*}*@diee.unica.it*
†CCSR, University of Surrey, Guildford, UK {*p.navaratnam,s.vural,r.tafazolli*}*@surrey.ac.uk*

Abstract—Wireless Sensor Networks (WSNs) have evolved into increasingly complex systems with the added capabilities to run distributed applications. In order to meet application requirements, such as execution time, and to efficiently use limited node energy resources, a task allocation strategy is required. Basically, the set of tasks that have to be completed in order to finalise an application has to be distributed among sensor nodes and scheduled to be processed in the best possible way.

Centralised task allocation algorithms, in which decisions on task execution are taken by a single node, can avoid the exchange of data packets between task execution nodes. However, such algorithms cannot adapt to dynamic network conditions, and suffer from computational complexity. Therefore, in this paper, we propose an adaptive and decentralised Task Allocation Negotiation algorithm (TAN) based on non-cooperative game theory, where neighbouring nodes engage in negotiations to maximize their own utility functions. TAN is compared to three other algorithms: (i) baseline setting with no task assignment to multiple nodes, (ii) centralised task assignment lifetime optimisation, and (iii) a dynamic distributed algorithm, DLMA. Simulation results show that TAN outperforms these algorithms in terms of application completion time and average energy consumption.

Index Terms—Wireless Sensor Networks, task assignment, game theory.

1 Introduction

The progress in Wireless Sensor Network (WSN) technology, both in terms of processing capability and energy consumption reduction, has evolved WSNs into complex systems that can gather information about the monitored environment and make prompt and intelligent decisions. These developments have contributed to the efforts on integrating sensor devices in Future Internet (FI) architectures [1], which enables effective interfacing with other technologies. In doing so, a horizontal ambient intelligent infrastructure is made possible, wherein the sensing, computing, and communicating tasks can be completed using programmable middleware that enables quick deployment of different applications and services.

One of the major issues with WSNs is maximisation of network lifetime, due to the fact that sensors are mainly battery powered. Many efforts have been made so as to extend network lifetime, one of which is the development of algorithms that effectively assign execution tasks to network nodes. One method to perform task assignment is the use of a central controller that divides large application programs into smaller and easily executable tasks and then distributes these tasks to nodes. Task allocation solutions that consider a central controller are called centralised solutions. For instance, in [2] a centralised solution for the maximization of the WSN lifetime is proposed. In [3][4][5], centralised algorithms for both reducing network energy consumption and application execution time are proposed. However, these centralised algorithms suffer from high computational complexity, especially for WSNs with a large number of nodes. Furthermore, centralised algorithms have to frequently collect updates from nodes in order to adapt to network dynamics, which is difficult to achieve and incurs large control packet overhead.

Due to the drawbacks of centralised solutions, distributed solutions are recently proposed to perform task allocation in WSNs. For instance, in [6] a communication scheme based on *gossiping* is used, while in [7] a particle swarm optimization algorithm is adopted. These distributed algorithms reduce the problem complexity, as only local areas are considered rather than the whole network. The communication overhead caused by packet exchanges between network nodes and the central controller is also avoided. Nevertheless, none of them take into account the execution time of the application assigned to the network. This could lead to an inconvenient task assignment strategy, which may entail that the application deadline is reached before the application is executed.

In this paper, a new distributed algorithm for the allocation of application tasks among WSN nodes is proposed, which is named Task Allocation Negotiation (TAN). This algorithm aims to reduce application execution time as much as possible, with minimal network energy consumption while inducing less computational complexity than existing algorithms have. The major contribution is the adoption of the rules of non-cooperative game theory [8]. Sensor nodes negotiate among each other to set the application configuration. In doing so, each node aims at maximising a *node* *utility function* under the constraints set by neighbouring sensors. We then prove that the scenario under consideration is a *potential game* [9], which means that every improvement of the nodes' utility functions corresponds to the same improvement in the utility perceived by the whole network, which implies that the problem has a unique outcome that is reachable in finite time.

The obtained solution reduces the average node energy consumption rate (and hence contributes to network lifetime extension) and simultaneously meets the application completion time requirements. Simulation results show significant performance improvements in both average node energy consumption and application completion time in a typical reference scenario, with respect to three settings: (i) the case where the whole application is carried out by a single sensor node, (ii) the centralized task assignment approach presented in [2], and (iii) the distributed task assignment approach presented in [6]. Furthermore, TAN is tested for random WSN scenarios. Noticeable gains are obtained for networks with high node density and highly populated clusters, especially when TAN is applied to complex applications.

This paper is organized as follows. Section 2 presents the related works, Section 3 introduces the problem and briefly describes the proposed approach, Section 4 describes the task assignment model, and Section 5 describes the algorithm. Finally, Sections 6 and 7 present the simulation results and conclusions.

2 Related Work

Since nodes in WSNs are battery powered, a great deal of effort has been made by researchers to find effective strategies that increase network lifetime. Different approaches have been proposed to this end. Some examples are: convenient deployment of sensors [10], use of efficient routing techniques [11], and use of static or mobile relay nodes that help balance network energy consumption among nodes [12][13][14]. However, advances in microchip technology have provided additional capabilities to sensors. Therefore, more advanced methods that extend network lifetime are now possible, besides these traditional techniques that either modify network topology or provide an energy-efficient routing protocol. Hence, not only is network lifetime optimisation centred on reduction of packet transmission power, but it also involves convenient data processing that reduces the amount of data delivered to data sinks. This is the principle behind node clustering protocols, such as LEACH [15] and EC [16], in which cluster head nodes aggregate data and reduce transmitted data volume, which in turn reduces the overall transmission energy consumption of the network.

A step forward in extending network lifetime is to consider not only data aggregation to reduce data volume, but also any possible data processing tasks based on network topology, battery power, and node processing capabilities. However, existing methods have limited scope in studying lifetime extension with regards to application data processing. For instance, in [17], maximization of cluster lifetimes is studied. However, this approach considers only communication tasks, but not the tasks generated by applications and assigned to the network for execution. Furthermore, it only focuses on homogeneous networks, which are not common in real scenarios. In contrast, [18] considers execution of application tasks, and provides an adaptive task allocation algorithm that aims at reducing the overall energy consumption by balancing residual node energy levels. However, this mechanism requires exchange of some additional messages among all the nodes in the network, which considerably increases packet overhead.

Some other studies in [3][4][5], improve network lifetime while reducing task execution time. However, the algorithms in these works are centralised, and therefore suffer from high computational complexity, as well as large control packet overhead due to frequent updates collected from nodes in order to adapt to network dynamics.

To overcome the issues encountered by centralised solutions, the authors in [6] propose an overlaying framework that determines the distribution of tasks among the nodes in a WSN by means of a distributed optimization algorithm, based on a gossip communication scheme, aimed at maximizing network lifetime. A similar approach is studied in [7], where a distributed algorithm is proposed based on particle swarm optimization. However, the major drawback of these studies is that they do not take into account the deadline of the applications assigned to the network.

3 Problem Formulation

The reference scenario considered in this paper is that of a hierarchical heterogeneous WSN. The WSN is hierarchically organised into clusters, with sensor nodes as cluster members, and cluster heads. The heterogeneity reflects the fact that, in most real scenarios, nodes with different characteristic parameters are used. The WSN under examination needs to run a required application, which can be divided into smaller tasks that can be assigned to network nodes to be executed. Due to the heterogeneous node characteristics, some nodes can perform the same task faster than others, and even spend less energy. The objective of the proposed algorithm, TAN, is to assign the tasks to suitable nodes, such that the processing time of the application is as short as possible and the network lifetime is as long as possible. TAN includes negotiations among nodes to determine task assignments. Whenever a node receives some data (along with the information on which particular tasks have to be performed on the data) it decides whether it should perform some tasks or not, depending on the potential contribution to the network in terms of faster processing time and longer lifetime.

3.1 Network Model

The network is modelled as a Directed Acyclic Graph (DAG) $\mathcal{G}_X = (X, E_X)$, where the vertices represent the nodes $X = \{1, \ldots, i, \ldots, N\}$, while the links are described by the set of edges $E_X = (e_{ij})$, where each edge represents a connection from node i to node j. The network is organised into clusters. Each cluster is controlled by a single cluster head (CH) which collects sensory data from nodes in its cluster. There is a single hop between a sensor node and its corresponding CH. At the top of the hierarchy is a gateway (GW) node, which collects data received from the CHs. Sensors transmit their data to the CHs, which then deliver the collected data to the gateway through a path of CHs. Routing paths over the CHs is determined by conventional routing protocols [11], such as Self-Organizing Protocol, Location-Based Routing Protocol, etc. In this paper, Hierarchical Power-Aware Routing protocol is used [19], due to the fact that it is based on a trade-off between minimizing power consumption and maximizing the minimal residual power of the network.

Negotiations are performed among neighbour nodes; multihop negotiations do not take place, and this significantly reduces the number of message exchanges. No communication is allowed between sensors belonging to different clusters. This means that, given two nodes i and j, $e_{ij} \in E_X$ if and only if i and j are in the same cluster. Such an architecture helps reduce the overhead caused by the negotiations. Figure 1 shows the reference architecture of the network under examination. In this Figure, SN_i and CH_j are communicating with their neighbours.

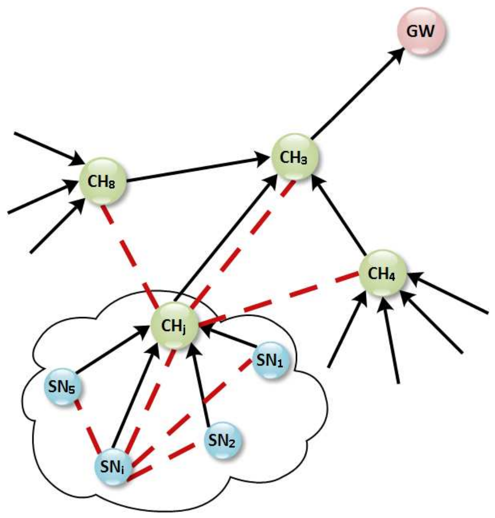


Fig. 1. Architecture of the network. Solid lines represent connections formed by the routing protocol; dashed lines represent connections between negotiation nodes.

3.2 Task Model

We consider a single large application assigned to the network for execution, which is decomposed into a set of tasks. This application can be described as a DAG of tasks $\mathcal{G}_T = (T, E_T)$, where $T = \{1, \ldots, \lambda, \ldots \Lambda\}$ is the set of tasks, and $E_T = (e_{vw})$ is the set of edges, with each edge e_{vw} representing a unidirectional data transfer from task v to task w. An example of the task DAG is depicted in Figure 2. The gateway initially sends the application graph $\mathcal{G}_T$ to nodes, so that each node learns the relations and dependencies between the tasks.

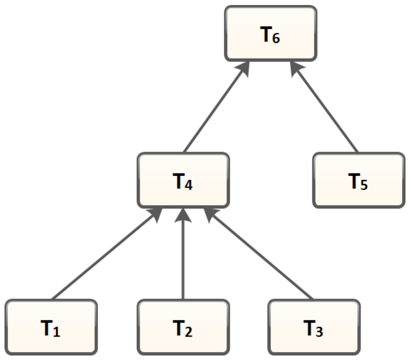


Fig. 2. Example of a task DAG.

A binary vector $\boldsymbol{s}(i) = [s(i, \lambda)]$, for $\lambda \in T$, can be assigned to each node i in the network, where $s(i, \lambda)$ is a boolean value representing the current state of node i corresponding to task λ, i.e. $s(i, \lambda) = 1$ when node i is performing task λ. The vector $\boldsymbol{s}(i)$ is called the *task assignment strategy* of node i. The state $s(i, \lambda)$ can only be set to 1 if and only if all its predecessor tasks have been assigned to a node, i.e. $s(i, \lambda) = 1$ if $s(j, l) = 1, \ \forall l \in T_{in}(\lambda)$, where $T_{in}(\lambda)$ are the ingress tasks for task λ. With reference to Figure 2, $T_{in}(T_1) = T_{in}(T_2) = T_{in}(T_3) = T_{in}(T_5) = \{\}$, hence they are *source tasks*, $T_{in}(T_4) = \{T_1, T_2, T_3\}$, and $T_{in}(T_6) = \{T_4, T_5\}$.

Since not every node may be able to perform each and every task, a binary vector $\boldsymbol{d}(i) = (d(i, \lambda))$ is defined, where $d(i, \lambda) = 1$ if node i is able to perform task λ. Note that $s(i, \lambda) = 1$ is possible only if $d(i, \lambda) = 1$, which means that $d(i, \lambda) \geq s(i, \lambda)$.

During the execution of the proposed algorithm, two sets of tasks are formed: the set of tasks $T_{prev} = \{1, \ldots, h, \ldots, H\}$ that have been already assigned to nodes and the set $T_{next} = \{1, \ldots, k, \ldots, K\}$ of those tasks that are yet to be assigned. This means that, for each task $h \in T_{prev}$, there is a node i for which its state $s(i, h)$ is equal to 1. We assume that source tasks are already assigned to the nodes by the gateway according to the output required by the application. For example, if the application requires the temperature measurement for a certain area, source tasks correspond to the temperature sensing for that area. In this case, the gateway assigns these source tasks to the sensor nodes (SNs) that are located in that area. Hence, initially, the set T_{prev} is only populated with source tasks, i.e. with reference to Figure 2, $T_{prev} = \{T_1, T_2, T_3, T_5\}$ and $T_{next} = \{T_4, T_6\}$.

3.3 Energy Consumption

In this paper, two types of energy consumption at network nodes are considered, namely processing energy consumption and communication energy consumption.

Processing energy consumption e_{proc} depends on two quantities: the number of instructions, I_λ, needed to execute a task, and the average energy consumption per instruction $e_{instr}(i)$ at node i. Hence:

$$e_{proc}(\lambda, i) = I_\lambda \cdot e_{instr}(i). \tag{1}$$

There are two main components that contribute to communication energy consumption: transmission and reception energy consumption [20]. We define the energy per bit necessary to transmit data at rate R from node i to node j (one-hop neighbours) as $e_{tx}(i,j)$, and the per-bit energy consumed for reception of data at node j as $e_{rx}(j)$, which are calculated by:

$$\begin{aligned} e_{tx}(i,j) &= \frac{1}{R}\left(P_{T0_i} + \frac{\varphi_{ij} \cdot \delta_{ij}^{\gamma}}{\eta_i}\right), \\ e_{rx}(j) &= \frac{P_{R0_j}}{R}, \end{aligned} \tag{2}$$

where P_{T0_i} and P_{R0_j} are the power consumption components of the transmitting and receiving circuitry for node i and node j, respectively. η_i is the drain efficiency of the power amplifier in i; φ_{ij} is a coefficient that is proportional to the minimum reception power and depends on antenna characteristics; δ_{ij} is the distance between the transmitter and the receiver; and finally, γ denotes the path loss component. These expressions are described in more detail in [20].

4 THE TASK ASSIGNMENT MODEL

The aim of the optimization problem is to finding the trade-off that best fits the requirements of both minimizing the application completion time and maximizing the network lifetime. However, this is an NP-hard problem [2][3], which could be exceedingly time and energy consuming. For this reason, in this section, we use a non-cooperative game model [21] for the task allocation problem in WSNs. Neighbouring nodes negotiate in order for each node i to choose a task assignment strategy $\boldsymbol{s}(i)$ that maximizes its own utility function.

4.1 Definitions

A non-cooperative game is defined by the tuple $\Gamma = \langle X, \{\boldsymbol{s}(i), u_i\}_{i \in X}\rangle$, where a utility function u_i is assigned to node $i \in X$ for the given strategy vector $\boldsymbol{s}(i)$. The goal of each node is to maximize its own utility in a rational way. Therefore, a strategy $\boldsymbol{s}^*(i)$ is preferred to a strategy $\boldsymbol{s}(i)$ if and only if $u_i(\boldsymbol{s}^*(i)) > u_i(\boldsymbol{s}(i))$. For simplicity, we denote $\boldsymbol{S} = \bigcup_{i \in X} \boldsymbol{s}(i)$ to refer to the strategy of all the nodes in the network.

In the following, we consider each node i as characterised by five parameters, which are described in Table 1.

TABLE 1
Node parameters

Parameter	Description
$t_{instr}(i)$	Time needed by node i to perform a single instruction
$e_{instr}(i)$	Energy spent by node i to perform a single instruction
$\boldsymbol{e}_{tx}(i) = (e_{tx}(i,j))$	Each element $e_{tx}(i,j)$ of this vector is the energy spent per bit to send data from node i to an adjacent node j
$e_{rx}(i)$	Per bit reception energy at node i
$e_{res}(i)$	Residual energy of node i

4.2 Task Utility Function

The TAN algorithm decides which particular node should execute a given task k by maximizing a *task utility function* over the set T_{next} of unassigned nodes. The task utility function for a generic task depends on the node that is assigned to it. If no node is assigned to task k, its utility value is 0, consistently with the fact that, if task k is not assigned to any node to be executed, there is no utility associated with it. If task k is assigned to node i, its utility value is proportional to the utility that the network has if task k is executed by node i. In particular, this utility is in inverse proportion with the time needed by node i to complete task k, and with the reduction in lifetime due to this execution. Therefore, the task utility function is given by:

$$u_k(\boldsymbol{S}) = \max_{i \in X}\left\{\left[\Omega_t(i,k) + \frac{\alpha}{NF(k)} \cdot \Omega_\tau(i,k,\boldsymbol{S})\right] \cdot s(i,k)\right\}, \tag{3}$$

where $\Omega_t(i,k)$ is the *task completion time component* of task k when it is performed by node i, $\Omega_\tau(i,k,\boldsymbol{S})$ is the *network lifetime component* when task k is performed by node i according to the strategy $\boldsymbol{S}$, $NF(k)$ is a normalization factor that eliminates the difference in magnitude between the task completion time and the network lifetime values, and $\alpha > 0$ is a weighting factor.

4.2.1 *Task Completion Time Component*

As defined in Section 3.2, the set T_{next} is formed of those tasks that are to be assigned to nodes for execution. Each task $k \in T_{next}$ is characterized by two parameters:

1) the deadline for successfully completing a task, $t_d(k)$;
2) the number of required instructions, $I(k)$.

We express the task completion time component as:

$$\Omega_t(i,k) = \frac{t_d(k) - t_c(i,k)}{t_d(k)}, \tag{4}$$

where $t_c(i,k)$ is the completion time if task k is performed by node i, which is defined as:

$$t_c(i,k) = \begin{cases} I(k) \cdot t_{instr}(i), & \text{if } t_c(i,k) \le t_d(k) \\ t_d(k), & \text{if } t_c(i,k) > t_d(k) \end{cases}.$$

4.2.2 Network Lifetime Component

This component is defined as follows:

$$\Omega_\tau(i,k,\boldsymbol{S}) = F_p(i,k) + F_{tx}(i,k,\boldsymbol{S}), \tag{5}$$

where $F_p(i,k)$ is the component related to the change in network lifetime due to the processing cost needed to perform task k in node i. This component is defined as:

$$F_p(i,k) = -I(k) \cdot \frac{e_{instr}(i)}{e_{res}(i)}. \tag{6}$$

Note that, since data processing entails a certain amount of energy consumption, $F_p(i,k)$ cannot increase task k's utility, and thus it needs to be negative in order to decrement the utility of node i when task k is executed at node i.

The term $F_{tx}(i,k,\boldsymbol{S})$ in Equation 5 is related to the change in network lifetime due to the transmission (and reception) of the necessary data for task k to node i, i.e. it represents communication costs in the task utility function. Let $\boldsymbol{p}_{i\to j} = \{(i,a),(a,b),\ldots,(e,f),(f,j)\}$ be the routing path that connects node i to node j, and HL be a hierarchically higher-level node for nodes i and j. The transmission term is then:

$$\begin{aligned} F_{tx}(i,k,\boldsymbol{S}) &= -C_{tx}(p_{i\to HL},k) \\ &+ \sum_{l\in T_{in}(k)} \sum_{j\in X} \Big\{ C_{tx}(p_{j\to HL},l) - C_{tx}(p_{j\to i},l) \Big\} \cdot s(j,l), \end{aligned} \tag{7}$$

where

$$C_{tx}(p_{i\to j},k) = \sum_{(x,y)\in p_{i\to j}} \left(\frac{e_{rx}(y)}{e_{res}(y)} + \frac{e_{tx}(x,y)}{e_{res}(x)} \right) \cdot n(k). \tag{8}$$

In Equation 8, $C_{tx}(p_{i\to j},k)$ is the cost to transmit (and receive) data from node i to node j, and $n(k)$ is the number of output bits for task k. The difference operation is due to the fact that the difference in lifetime between two cases is considered: in the first case, task k is not performed, and input data for task k are sent directly to the higher-level node. In the second case, input data for task k are sent to node i where task k is performed, and then the output is sent to the hierarchically higher-level node. Note that, contrary to the processing component $F_p(i,k)$ (Equation 6) that is always negative, the transmission component $F_{tx}(i,k,\boldsymbol{S})$ (Equation 7) may also be positive, increasing the task utility function (Equation 3) and making it more convenient to perform task k at node i rather than delegating the processing job to the higher-level node.

4.2.3 Normalization factor $NF(k)$

In Equation 3, the normalization factor $NF(k)$ is introduced so as to make the magnitudes of the network lifetime and task completion time components comparable to each other. Since $NF(k)$ is independent from the task assignment strategy, its value can be computed offline before the negotiation starts. The normalization factor is computed by:

$$NF(k) = \frac{\overline{\Omega}_\tau(k)}{\overline{\Omega}_t(k)}, \tag{9}$$

where $\overline{\Omega}_\tau(k)$ and $\overline{\Omega}_t(k)$ are the mean values of $\Omega_\tau(i,k,\boldsymbol{S})$ and $\Omega_t(i,k)$, computed over all nodes $i \in X$.

4.2.4 Weighting factor α

The parameter α introduced in Equation 3 is a coefficient of the network lifetime component in the task utility function. If α tends to 0, the utility function is mostly influenced by the task completion time component; the higher its value is, the more the changes in the network lifetime component are reflected by the utility function. Clearly, if α tends to 1, the task completion time component $\Omega_t(i,k)$ and the network lifetime component $\Omega_\tau(i,k,\boldsymbol{S})$ have comparable magnitudes, and therefore comparable influences on the utility function.

4.3 Network Utility Function

Given the task utility function defined in Equation 3, the network utility function is defined as the sum of all task utility functions for those tasks that are not yet assigned to nodes:

$$u_g(\boldsymbol{S}) = \sum_{k\in T_{next}} u_k(\boldsymbol{S}). \tag{10}$$

This equation implies that the network utility function is maximized when the sum is maximized. However, this is only possible if all the nodes in the network could communicate with each other. The communication overhead resulting from a negotiation of this type would entail an additional transmission cost that would counter the benefit of the maximisation itself, particularly for large networks. For this reason, we choose to let each node negotiate only with its neighbours, achieving a sub-optimal but computationally more efficient solution. Hence, the definition of a *node utility function* is required.

4.4 Node Utility Function

The node utility function must be defined in such a way that any increment in its value must correspond to an equivalent increment in the network utility function of Equation 10. In doing so, an approximation based on local negotiations can be obtained, without the need for network-wide negotiations. The node utility function u_i can then be written as an aggregation of the *marginal* contributions $u_k^{mar}(\boldsymbol{s}(i))$ of node i to each task k (and therefore to the network utility function) given by:

$$u_i(\boldsymbol{s}(i)) = \sum_{k\in T_{next}} u_k^{mar}(\boldsymbol{s}(i)). \tag{11}$$

The marginal contribution is defined as a *Wonderful Life Utility* (WLU) in [22]. WLU is the difference between the task utility for a given node strategy $\boldsymbol{s}(i)$ and the task utility for the null strategy $\boldsymbol{s}_0(i)$, in which all the elements are equal to 0, i.e. the node is not contributing to any task. The marginal utility u_k^{mar} of node i to task k is then computed by:

$$u_k^{mar}(\boldsymbol{s}(i)) = u_k(\boldsymbol{s}(i)) - u_k(\boldsymbol{s}_0(i)). \qquad (12)$$

Note that marginal utility is null when the node is not contributing to the task as a part of its strategy.

From Equation 12, we can infer that the marginal contribution of node i to task k is not null only if the node is contributing to the task. Furthermore, since the network utility function in Equation 10 is a summation of task utility functions (Equation 3), a node contributes to the network utility when it contributes to at least one task. Therefore, a change in node i's strategy $\boldsymbol{s}(i)$ that increases its utility entails the same increment in the network utility. This property is particularly desirable because it implies that the game under consideration is a *potential* game [9], where the *potential function* is given by the network utility function. A consequence is that this game has at least one pure Nash equilibrium [9][1]. Furthermore, potential games have the *Finite Improvement Property*: every sequence of changes in the strategy that improves the network utility converges to a Nash equilibrium in finite time[2].

5 TAN Algorithm

The aim of the TAN algorithm is to maximize the network utility function given by Equation 10. To achieve this, individual node utility functions are maximized by means of a negotiation accomplished by neighbouring nodes. The negotiation is based on a greedy search algorithm, called Distributed Stochastic Algorithm (DSA) [23], which is proved [24] to provide a solution more quickly than existing algorithms, such as Distributed Breakout Algorithm (DBA) [23] and Maximal Gain Messaging (MGM) [25]. Before introducing TAN in detail, it is essential to explain DSA, which is next.

5.1 DSA

DSA belongs to the family of well known greedy local search algorithms for the solution of distributed constraint optimization problems. These kind of algorithms do not require any global control mechanisms, and agents involved in the optimization only need information about themselves and their neighbours. This ensures a low computational complexity and communication cost. Past studies compared local greedy search algorithms, and proved that DSA converges to a solution faster than existing algorithms such as DBA [23] and MGM [25].

1. Pure Nash equilibria are characterised by a unique outcome, contrary to mixed Nash equilibria where the outcome is stochastically variable.

2. This property ensures that many simple adaptive processes, such as the Distributed Stochastic Algorithm (DSA) [23], converge to Nash equilibria.

DSA pseudo-code is provided in Algorithm 1. At each time step t, each node that changed its strategy in the previous time step $t-1$, sends to all other involved nodes a *strategy update* message (SUM) which contains its new strategy $s^*(i)$. A random number generator $rand()$ assigns a value to parameter v (see Algorithm 1). If a node receives a SUM and if the value v is higher than a given probability p, then a new strategy $\boldsymbol{s}(i)$ that maximizes its own utility $u_i(\boldsymbol{s}(i))$ (Equation 11) is computed. The probability value p is known as the *degree of parallel executions*. The value of p affects the performance of DSA: the higher it is, the higher the probability that all nodes simultaneously change their strategies, which increases the communication costs and the algorithm's convergence time. On the other hand, low values may lead to a large convergence time, due to less frequent changes in node strategies. Once the utility is computed, if it is already maximized by the current strategy, no further changes occur at step t. DSA terminates when no new SUMs are received within the last t_{wait} seconds. This occurs when the value of the *counter* variable of Algorithm 1 exceeds t_{wait}. *counter* represents the number of seconds since the last SUM message.

Algorithm 1 DSA for node i

```
1:  while $counter < t_{wait}$ do
2:      if $\boldsymbol{s}^*(i)$ was found at $t-1 < t$ then
3:          broadcast a SUM to all neighbours
4:      end if
5:      if a SUM has been received then
6:          $counter \leftarrow 0$
7:          $v \leftarrow rand()$
8:          if $v > p$ then
9:              find $\boldsymbol{s}(i)$ that maximizes $u_i(\boldsymbol{s}(i))$
10:             if $u_i(\boldsymbol{s}(i)) > u_i(\boldsymbol{s}^*(i))$ then
11:                 $\boldsymbol{s}^*(i) \leftarrow \boldsymbol{s}(i)$
12:             end if
13:         end if
14:     else
15:         $counter \leftarrow counter + t_{step}$
16:     end if
17: end while
```

5.2 TAN

The TAN algorithm consists of the whole procedure to assign the tasks in T_{next} to the nodes in X, in order to maximize the network utility function $u_g(\boldsymbol{S})$. In fact, the highest value of the network utility function corresponds to the highest utility that the network has with reference to the strategy of the network, i.e. the task assignment to network nodes. In other words, TAN finds the network strategy that ensures the best outcome in terms of task completion time and network lifetime.

At each step of the algorithm, some nodes in the network hold some data, on which some tasks in T_{next}

need to be performed. These nodes first exchange the information required for DSA with their neighbouring nodes. Then, all the nodes in the involved neighbourhoods run DSA until a suitable strategy for all nodes is found. Finally, tasks are performed according to the strategy determined with DSA, and processed data are sent to the higher-level node.

The pseudo-code of TAN is provided in Algorithm 2. The algorithm starts as soon as source tasks are performed, and runs until the set T_{next} is empty, i.e. there are no remaining tasks to be performed. Let X_{data} be the set of nodes that have some output data. Initially, X_{data} is the set of nodes that have just performed the source tasks. If the set T_{next} is not empty, i.e. if there are some remaining processing tasks that can be performed on the data, then each node $i \in X_{data}$ sends an *information* message (INFO) to its neighbours, $n(i)$. All the nodes in $n(i)$ reply by sending an INFO message with their own information to their neighbours.

An INFO message includes:

1) $\boldsymbol{s}(i)$, $e_{instr}(i)$, $e_{tx}(i)$, $e_{res}(i)$ (see Table 1).
2) The subset $T'_{prev} \subseteq T_{prev}$ of tasks that are already performed on the data that node i currently holds.

Note that initially the only tasks that are already performed are source tasks, i.e. T'_{prev} is made of the source tasks.

After all nodes exchange INFO messages with their neighbours, the DSA algorithm is initiated at each node. Once DSA converges, each node will have chosen the strategy that maximizes the network utility function $u_g(\boldsymbol{S})$ (Equation 10).

Once that the strategy is chosen, tasks need to be executed by nodes according to their strategy, before sending data to the higher level node. Let $X_{proc} = \{x^1_{proc}, \dots, x^k_{proc}, \dots\}$ be the sequence of nodes in $n(i)$, for which the strategy of node x^k_{proc} entails the execution of processing task t^k_{proc}, and let $T_{proc} = \{t^1_{proc}, \dots, t^k_{proc}, \dots\}$ be the related sequence of processing tasks to be performed. Here, the nodes in set X_{proc} are ordered according to the order in which the tasks in set T_{proc} need to be performed, i.e. task t^k_{proc} is in the set $T_{in}(t^{k+1}_{proc})$. Since a node can perform more than one task, it is possible that two consecutive tasks t^k_{proc} and t^{k+1}_{proc} are executed by the same node, i.e. $x^k_{proc} = x^{k+1}_{proc}$. Each node x^k_{proc} first performs the task t^k_{proc}, and updates the sets T_{next} and T_{prev} accordingly. Then, if there are no other tasks assigned to it, node x^k_{proc} sends a *data* message (DATA) to node x^{k+1}_{proc}, to perform task t_proc^{k+1}. DATA messages contain the set T_{prev}, along with the output data resulting from task t_proc^k.

When all the tasks in T_{proc} are performed, a DATA message is sent to the higher-level node.

Figure 3 shows the sequence of the steps to perform TAN, for a reference neighbourhood of 3 sensor nodes and a cluster head. Each sketch corresponds to one of the basic steps necessary to TAN execution. Node 2 is the node that has task 1 output data. After the DSA execution, tasks 2 and 3 are assigned to nodes 3 and 1, respectively. Data is then sent to the cluster head, which negotiates with their neighbours for the assignment of task 4 that is still to be performed.

Algorithm 2 TAN

```
 1: Source tasks are performed
 2: while $T_{next} \neq \{\}$ do
 3:     for all $i \in X_{data}$ do
 4:         send INFO to all nodes in $n(i)$
 5:         for all $j \in n(i)$ do
 6:             send INFO to all nodes in $\{\{i, n(i)\} \backslash \{j\}\}$
 7:         end for
 8:         for all nodes in $\{i, n(i)\}$ do
 9:             execute DSA
10:         end for
11:         for all $x^k_{proc} \in X_{proc}$ do
12:             perform $t^k_{proc}$
13:             $T_{next} \leftarrow \{T_{next} \backslash \{t^k_{proc}\}\}$
14:             $T_{prev} \leftarrow \{T_{prev}, \{t^k_{proc}\}\}$
15:             if $x^{k+1}_{proc} \neq x^k_{proc}$ then
16:                 send DATA to $x^{k+1}_{proc}$
17:             end if
18:         end for
19:         send DATA to the higher-level node
20:         update $X_{data}$
21:     end for
22: end while
```

6 TAN Complexity Analysis

In this Section, TAN algorithm complexity will be analysed in terms of computational complexity and message complexity.

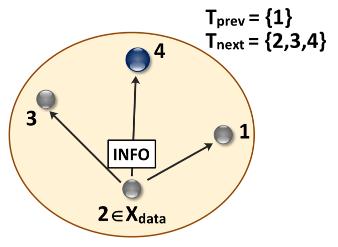
6.1 Computational Complexity of the Node Utility Function Maximization

The maximization of the node utility function $u_i(\boldsymbol{s}(i))$ defined in Equation 11 is a Mixed Integer Linear Programming problem (MILP) [26]. It is well known that MILP problems are optimally solved using branch-and-bound algorithms [27], which, in the worst case, have a complexity that grows exponentially with respect to the number of variables. Hence, the complexity of the node utility function is related to the number of variables a node uses to compute the function.

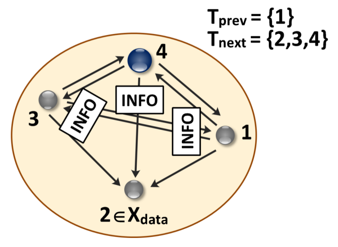
Note that the tasks in $\boldsymbol{d}(i)$ are node i's variables used for executing TAN. Hence, the number of variables $N_{vars}(i)$ of node i is equal to the number of tasks it needs to execute, $N_{task}(i)$. [3] The number of node i's tasks is found by:

$$N_{vars}(i) = \sum_{\lambda \in T_{next}} d(i, \lambda).$$

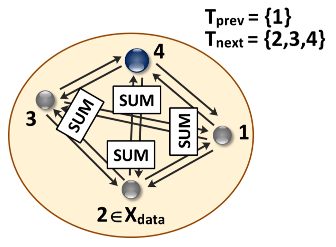
3. A node i can only perform the tasks that are set in its vector $\boldsymbol{d}(i)$, but not all tasks in T_{prev}.



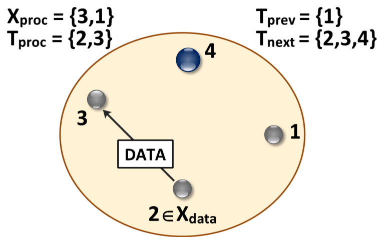
(a) Node $2 \in X_{data}$ sends INFO



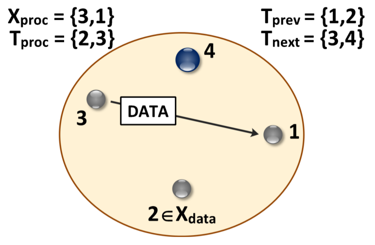
(b) Neighbours send INFO



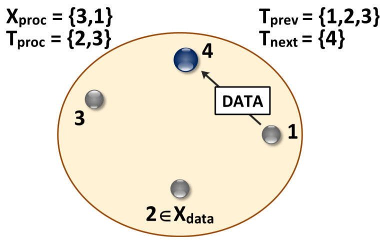
(c) DSA execution



(d) Node 2 sends DATA to $x^1_{proc} = 3$



(e) Node $x^1_{proc} = 3$ sends DATA to $x^2_{proc} = 1$



(f) Node $x^2_{proc} = 1$ sends DATA to the CH

Fig. 3. Sequence of steps to perform TAN. Small nodes represent sensors, whereas the bigger ones are cluster heads (CHs).

In the worst case, node i is able to perform all the tasks in T_{next}, and T_{next} is equal to the number of processing tasks in which the application is sub-divided. This means that, in the worst case, N_{vars} is bounded by $|T|$. In the worst case, the complexity of maximizing $u_i(\boldsymbol{s}(i))$ is equal to the complexity of performing an exhaustive search among all possible combinations of processing assignments, i.e. $2^{N_{vars}} \cdot N_{operations}$, where $N_{operations}$ is the number of operations needed to evaluate one node utility function $u_i(\boldsymbol{s}(i))$ ($N_{operations} \sim 30$). Note that computational complexity can be considerably reduced using sub-optimal heuristic algorithms, such as genetic or tabu-search algorithms.

6.2 Message Complexity of the Node Utility Function Maximization

In this section, the message complexity caused by the exchange of INFO, SUM, and DATA messages is analysed.

INFO: The INFO message is broadcast within the cluster by each node only once. Let $N_{cluster}(i)$ be the number of nodes within the cluster that node i belongs to, and let $n(\text{INFO})$ be the number of bytes required for an INFO message [4]. The number of transmitted bytes due to the exchange of INFO messages for each negotiation is therefore $N_{cluster} \cdot n(\text{INFO})$. Typically, $N_{cluster}$ is around $5-10$ [28].

SUM: During the negotiation, a SUM message, which consists of $|\boldsymbol{s}(i)|$ boolean values, is broadcast within the cluster whenever node i changes its strategy. Let N_{steps} be the number of steps taken until the negotiation converges, and let $n(\text{SUM})$ be the number of bytes of a SUM message. In the worst case, at every time step, each node in the cluster changes its strategy. Hence, the worst case number of transmitted bytes for SUM messages during each negotiation is $N_{steps} \cdot N_{cluster} \cdot n(\text{SUM})$.

DATA: After the negotiation process has converged, a DATA message of $n(\text{DATA}, k)$ bytes is sent to each node $x^k_{proc} \in X_{proc}$ (see Section 5.2), and then to the higher-level node. The dependence of $n(\text{DATA}, k)$ on k is due to the fact that the number of bytes of the output data varies based on which task t^k_{proc} is executed. $n(\text{DATA}, k)$ is proportional to both the amount of output data generated by the processed tasks and the number $|T_{prev}|$ of identification values. In the worst case, each node $x^k_{proc} \in X_{proc}$ is different from the following one x^{k+1}_{proc}. Therefore, considering that the last DATA message is sent to the higher-level node (HL), in the worst case, the number of transmitted bytes due to DATA messages for each negotiation is $\sum_{k \in \{X_{proc}, HL\}} n(\text{DATA}, k)$.

As a result, the worst case number of transmitted bytes sent during a single negotiation is:

$$\begin{aligned} n(negotiation) = &N_{cluster} \cdot n(\text{INFO}) + \\ &N_{steps} \cdot N_{cluster} \cdot n(\text{SUM}) + \\ &\sum_{k \in \{X_{proc}, HL\}} n(\text{DATA}, k). \end{aligned} \tag{13}$$

Considering the typical case of a network with a number of nodes per cluster $N_{cluster} = 10$ [28], and a number of steps for the negotiation convergence $N_{steps} = 10$ [23], for 30 tasks $n(\text{INFO}) \sim 137$, and $n(\text{SUM}) \sim 16$. Therefore, a typical value for the worst case amount of transmitted data per negotiation is $n(negotiation) \simeq 2.9\text{ kB} + \sum_{k \in \{X_{proc}, HL\}} n(\text{DATA}, k)$.

The number of negotiations to perform TAN mainly depends on the number of hierarchical levels of the network. Suppose that, given an application, all the source tasks are assigned to sensor nodes within the

4. $n(\text{INFO})$ consists of: $|\boldsymbol{s}(i)|$ boolean values, $|\{e^{ins}(i), \boldsymbol{e}^{tx}_i, e^{res}_i\}|$ numeric values, and $|T'_{prev}|$ identification values.

same cluster. After each negotiation, data are sent to the higher level node. Calling L the number of hierarchical levels, the number of negotiation is at most equal to $L-1$. On the other hand, if, given an application, all the source tasks are assigned to sensor nodes in different clusters, the number of negotiations depends on the number of sensing tasks as well. In the worst case, for each sensing task a negotiation is performed in each hierarchical level except the gateway. Therefore, calling M the number of sensing tasks, the worst case number of negotiations is $(L-1) \cdot M$.

7 Performance Analysis

The proposed algorithm is tested on heterogeneous WSN scenarios by MATLAB simulations. The main node parameters are listed in Table 2. More specifically, the processing cost parameters found in [3][20], the radio frequency parameters specified in [29], and IEEE $802.15.14$ parameters listed in [30] are used. CHs and sensors have different characteristics: CHs have a maximum processing speed of 206 MHz and a residual energy of 4 kJ, while sensor nodes have a maximum processing speed of 133 MHz and a residual energy of 2 kJ. The randomly assigned heterogeneous node characteristics are:

- non-uniform energy consumption rates: energy consumption parameters are selected randomly in a range from 60% to 140% of the mean values given in Table 1,
- non-uniform initial energy levels: battery energy level is assigned randomly from 20% to 100% of the maximum battery capacity,
- non-uniform processing speeds at each node: processing speed is set randomly from 60% to 100% of the highest possible processing speed.

TABLE 2
Simulation parameters

Parameter	Value
RF frequency	2400 MHz
Bit rate	250 kbps
Output power range	$\sim [-24\ 0]$ dBm
Receiver sensitivity	-94 dBm
P_{R0}	59.2 mW
P_{T0}	26.5 mW
η	50%
e^{instr}	1 nJ
Packet header	12 bytes
Maximum payload	125 bytes
Processing speed of sensors	133 MHz
Processing speed of CHs	206 MHz
Initial energy of sensors	2 kJ
Initial energy of CHs	4 kJ

In the following, the energy consumption model is presented. Then, the performance of the TAN algorithm for two different scenarios are presented. Provided figures present average values over all network nodes.

7.1 Smart City Scenario

The scenario under examination is that of an urban environment, where nodes are positioned along the streets as shown in Figure 4. Solid markers represent nodes equipped with sensors that measure the speeds of passing-by vehicles, where each speed measurement is represented by a double numerical value (64-bit long). Each street segment is a cluster (as enclosed by a dashed ellipse), where CHs are represented by empty markers. CHs are more capable nodes than ordinary sensors (with an initial battery charge twice higher than that of an ordinary sensor), and are able to perform the necessary tasks in their clusters, such as local management of sensors as well as traffic aggregation. CHs do not collect raw data, but process incoming data from the sensors in their clusters.

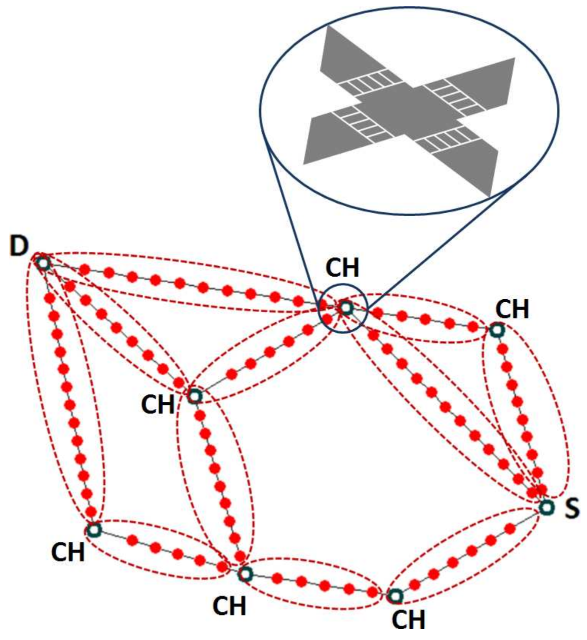


Fig. 4. Sample topology for the Smart City Scenario. Solid markers represent nodes equipped with speed sensors, while empty markers are cluster heads. Clusters of sensors are enclosed by dashed ellipses.

Suppose that a driver at point S in Figure 4 would like to know the fastest route to point D, based on the collected speed information from sensor nodes. For each of the 89 sensing nodes, one *speed sensing* task is assigned. Since the mean travelling time for each street segment is needed, the collected speed values from sensors in the same segment are used as input for a *mean speed computing* task. Then, the output of the mean speed computing tasks, all of which correspond to the same segment, are given to as input to another task, called the *mean travelling time*[5] *computing task*, which provides the mean value for that segment. To find the best path, the sum of the mean travelling times of all

5. The mean travelling time is the ratio between the mean speed value and the length value related to a segment street.

the combinations of segments that can be consecutively driven is required. This is done by the *summation* tasks. Finally, the *choice of best path* task takes the summation values related to two merging paths and returns the lower value, i.e. the quicker path. The ordinary nodes in Figure 4 are able to perform speed sensing and mean speed computing tasks, whereas the CHs are able to perform mean speed computing, mean travelling time computing, summation, and choice of best path tasks. Table 3 summarizes the tasks for this scenario.

TABLE 3
Tasks for the Smart City Scenario

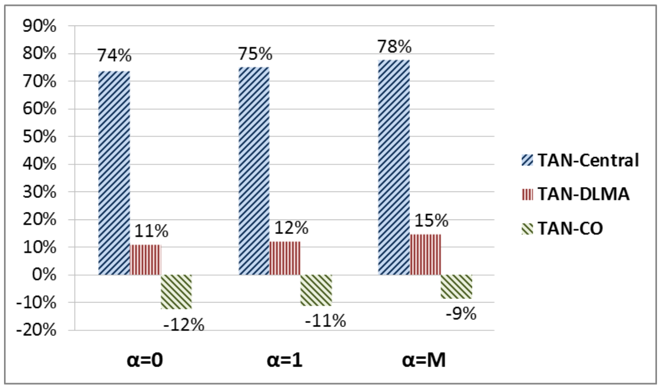
Task	Description	Nodes
Speed sensing	Senses the vehicles' speed. One speed sensing task is assigned to each sensor node	Sensors
Mean speed computing	Computes the average of the speed values collected by speed sensing tasks	Sensors, CHs, node S
Mean travelling time computing	Takes the mean speed value related to a single segment of a street and returns the mean travelling time for that segment	CHs, node S
Summation	Calculates the sum of the mean travelling times of two consecutive segments of a street	CHs, node S
Choice of best path	Takes two summation values related to two merging paths and chooses the fastest one	CHs, node S

We simulated the described scenario and compared the performance of the proposed algorithm with three alternative approaches:

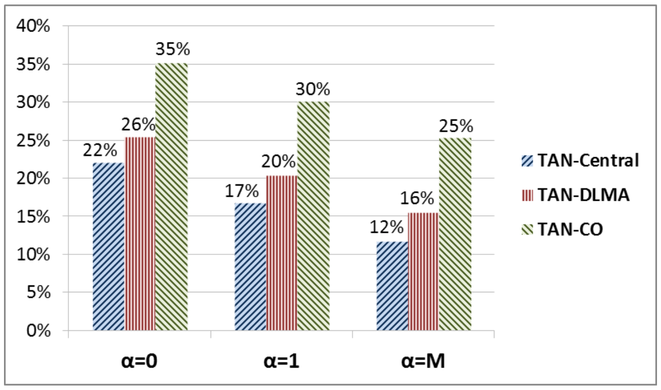
- *Mechanism Central:* All data is forwarded to and processed by the gateway (Node S in Figure 4).
- *Mechanism CO:* Data are processed according to a centralized lifetime optimization algorithm, described in [2]. In this algorithm, the gateway knows all the network and application characteristics. According to this information, the gateway assigns tasks to nodes with the aim of reducing the network energy consumption.
- *Mechanism DLMA:* Data are processed according to the DLMA algorithm described in [6]. DLMA is a distributed sub-optimal task assignment algorithm which aims at improving network lifetime. Contrary to TAN, it only takes into account network lifetime, ignoring the application execution time.

In this way, TAN's performance is evaluated with respect to static task assignment (mechanism Central), dynamic centralized optimal task assignment (mechanism CO), and dynamic distributed sub-optimal task assignment (mechanism DLMA). Note that, with reference to energy consumption, Mechanism Central and Mechanism CO are borderline cases. Even though in Mechanism Central energy consumption is expected to be higher than TAN, and no sub-optimal mechanisms could results in lower energy consumption than Mechanism CO, results are reported and compared to those cases with the aim of evaluating how much TAN's performance differs from theirs.

The obtained results of these mechanisms for energy consumption and application completion time are compared to those obtained using TAN. Figure 5 shows the percent gain of TAN over Central, CO, and DLMA. We refer to these comparisons as TAN-Central, TAN-CO, and TAN-DLMA.



(a) Energy conservation



(b) Completion time gain

Fig. 5. Percentage values of mean energy conservation and completion time gain using TAN, as compared to TAN-Central, TAN-DLMA, and TAN-CO, for different values of α.

Simulations are run for:

- $\alpha = 0$: null network lifetime component;
- $\alpha = 1$: comparable Ω_t and Ω_τ;
- $\alpha = M$, with $M \gg 1$: null task completion time component.

Significant improvement (around 75%) in energy conservation is observed compared to Central. TAN also provides up to 22% gain over Central in application completion time. When compared to DLMA, TAN shows moderate gains (up to 15%) in energy conservation, and up to 26% gain in application completion time. Despite the fact that CO is more energy-efficient than TAN ($\sim 12\%$), TAN achieves its highest gains in application completion time with respect to CO (up to 35%).

Note that, contrary to TAN, the Central, CO, and DLMA task assignment mechanisms do not take into account the task completion time[6], so their performance are the same irrespective of the value of α. For this reason, the most significant results are those obtained when the task completion time (see Equation 4) is negligible, i.e. when $\alpha = M$. Nevertheless, comparisons for non-infinitesimal values of the task completion time ($\alpha = 0$ and $\alpha = 1$) are reported for completeness.

As expected, when α increases, which means that the utility function is more in favour of the network lifetime component, the energy conservation percentage increases, while the completion time gain decreases. On the contrary, when α decreases, the utility function is in favour of the task completion time component, i.e. the energy conservation percentage decreases, while the completion time gain increases.

7.2 Random Network Scenarios

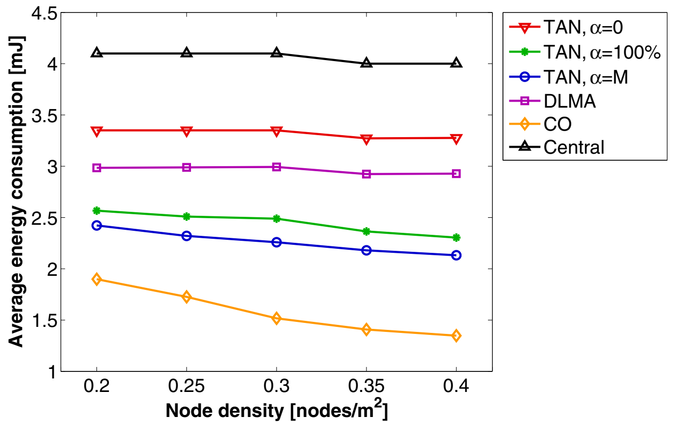
Apart from the Smart City Scenario of Figure 4, TAN is also evaluated for networks with randomly deployed sensors. Sensor locations are uniformly distributed over a rectangular topology, with two key parameters: node deployment density and average number of nodes per cluster.

The DAG of the application assigned to each network is obtained by starting with a set of tasks and then randomly appending the remaining tasks, while ensuring the structure is still a DAG. While doing so, the total number of tasks is fixed. The number of instructions required to perform each task and the number of output bits for each task are set randomly according to a uniform distribution from $2.7 \cdot 10^5$ to $3.3 \cdot 10^5$, and from 720 bits to 880 bits, respectively [31]. The application deadline is fixed and is set to 80 ms [3]. The ability of each node to perform a task[7] (the values of the elements corresponding to the tasks of each vector $\boldsymbol{d}(i)$ described in Section 3.2) is assigned according to a *task distribution parameter*. This parameter defines the probability of each node to be able to perform a given task[8]. Table 4 summarises the evaluation parameters.

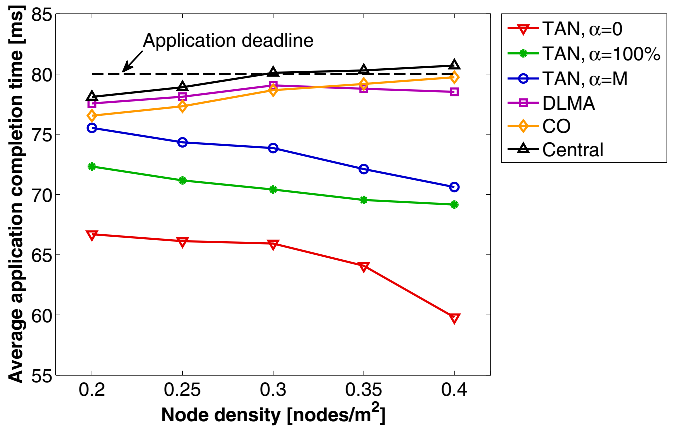
TABLE 4
Characteristic Parameters Values for Realistic Random Scenarios

Parameter	Min	Default	Max
Node density [nodes/m^2]	0.2	0.3	0.4
Number of nodes per cluster	5	15	25
Number of tasks	10	20	30
Distribution of tasks	10%	50%	100%

Note that the Smart City Scenario corresponds to a network with a low node density of around 0.15 nodes/m^2, a low number of nodes per cluster, which is around 8 nodes/cluster, a high number of tasks equal to 100, and a very low task distribution parameter of about 6.9%. In the following, the performance of TAN algorithm when these parameters change is discussed and compared to C, CO and DLMA. The results also include a comparison between TAN and the mechanism in which all the data are sent to the gateway and processed only by the gateway (mechanism Central in the Smart City Scenario).



(a) Average energy consumption



(b) Average application completion time

Fig. 6. Average energy consumption and application completion time for TAN, DLMA, CO, and Central task assignment mechanisms for different node densities.

7.2.1 The Effect of Node Density

In this set of simulations, the aim is to observe the algorithm performance when the inter-node distances change due to the change in node deployment density. Figure 6 shows how the averages of overall energy consumption and application completion time change when node density is altered, for different values of the weighting factor α of Equation 3.

In Figure 6(a), since the network lifetime component (Equation 5) depends on the hop distance to the higher level node due to multihop data transmission, we observe an improvement in energy consumption when α increases. For non-zero values of α, slight improvement

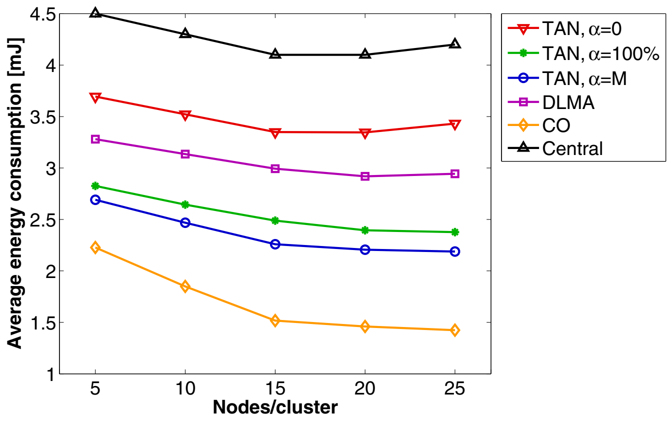
6. Recall that the task completion time is considered by TAN by means of the task completion time component in Equation 3

7. In the following, the word task is referred to as tasks in T_{next} at the initial instant. Since sensing tasks do not affect TAN performance, they are not considered as a critical parameter.

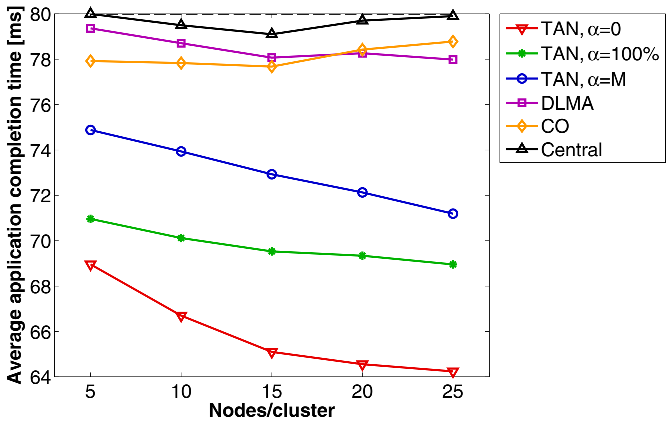
8. Note that this is different from $\boldsymbol{d}(i)$, which is a vector of boolean values denoting the ability of node i to perform individual tasks, but not the probability of the ability to execute the tasks.

in energy conservation is observed for increasing node density. This is due to the fact that, when node density increases, the energy consumption due to the message exchanges of negotiations is lower. Note that when $\alpha = 0$, the network lifetime component is not taken into account for choosing the best strategy, and this can be observed in the flatness of the related curve.

In Figure 6(b), some improvement in application completion time is observed with increasing node density. Although TAN (and all the other mechanisms) is outperformed by the optimal CO algorithm in terms of energy consumption, a gain of up to $\sim$20 sec is achieved by TAN when compared to CO, with respect to application completion time.



(a) Average energy consumption



(b) Average application completion time

Fig. 7. Average energy consumption and application completion time for TAN, DLMA, CO, and Central task assignment mechanisms for different number of nodes per cluster.

7.2.2 The Effect of the Number of Nodes in Clusters

The second set of simulations evaluates the effect of the average number of nodes in a cluster on TAN's performance. In Figure 7, the overall energy consumption and application completion time variations are shown with respect to different values of α and different number of nodes per cluster. It can be observed that the effect of having more nodes in a cluster on TAN's performance is similar to what is observed for the effect of node density increase. This can be attributed to the fact that TAN is mostly affected by how many nodes fall in a cluster's boundary, which directly affects its task allocation strategy. Note that TAN's negotiations are performed among neighbouring nodes within the same cluster. An increment in density or nodes per cluster is equivalent to an increasing node degree. This helps TAN refine the quality of its negotiation results at each algorithm step and eventually reduces algorithm convergence time and the total energy consumption.

7.2.3 The Effect of the Number of Tasks

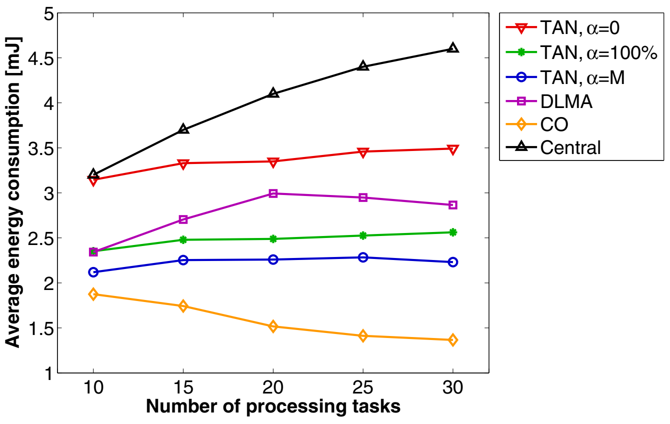
In order to evaluate TAN's performance at processing applications with different complexities, the number of tasks in an application is varied, while keeping the application deadline fixed. By doing so, the available time per task is effectively reduced. This in turn affects the application completion time component of Equation 1.

Figure 8 demonstrates the effect of the application complexity. We observe a noticeable improvement in both energy consumption and completion time when the number of tasks increases from 10 to 20. This is due to the fact that, for a low number of tasks such as 10, the results for an optimized strategy can not be much different from the case where all tasks are assigned to the gateway. One observation in Figure 8(b) is that TAN provides the shortest application completion times for all studied cases of application complexity. TAN with $\alpha = 0$ has a high consumption due to the fact that when $\alpha = 0$ the network lifetime component is not considered in Equation 3. Among all, CO has the least consumption, thanks to its ability to find the optimal solution with respect to the average energy consumption.

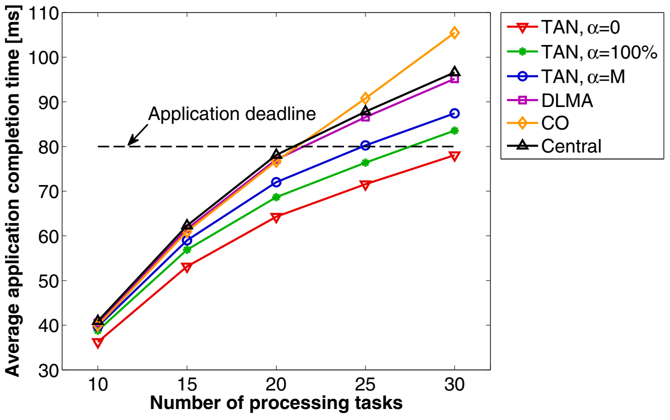
7.2.4 The Effect of the Task Distribution Parameter

In the last set of simulations, the effect of the task distribution parameter is studied. Recall that this value corresponds to the probability of a node to be able to perform a task. Figure 9 shows TAN performance of the algorithms. Noticeable improvement both in overall energy consumption and in application completion time using TAN is noticed when it is possible to assign more tasks to nodes, and the best results are obtained when the network is able to perform every task.

In the Smart City Scenario, the task distribution parameter is set to 6.9%, which is extremely low when compared to those of the random network scenarios. Nevertheless, the performance of TAN in random networks is worse compared to that in the Smart City scenario. This is due to the fact that, for the Smart City scenario, the ability of each node to perform a task is strictly related to the task (as opposed to being randomly assigned). Therefore, only the nodes that are more suitable to perform a given task have the ability to execute that task.

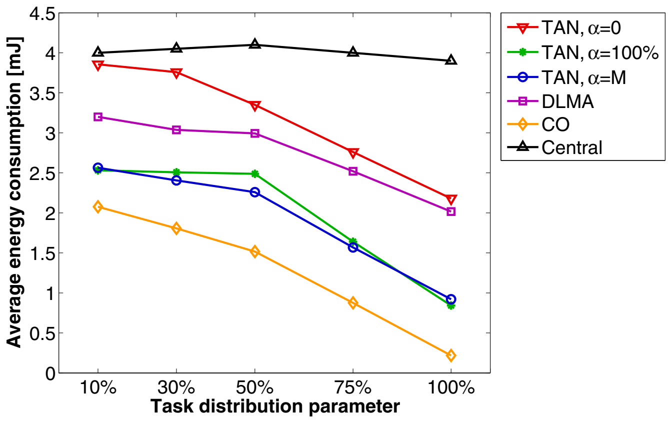


(a) Average energy consumption

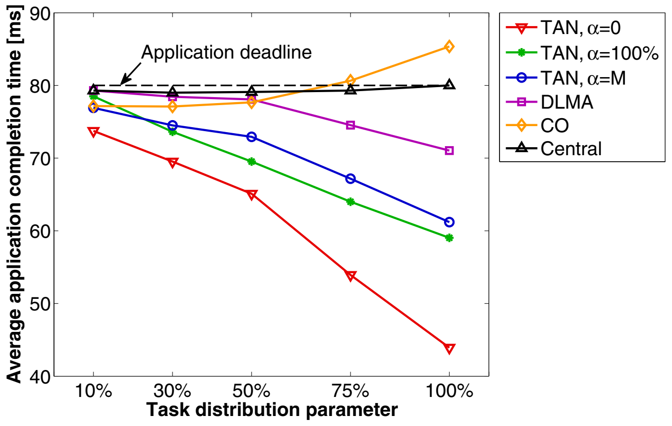


(b) Average application completion time

Fig. 8. Average energy consumption and application completion time for TAN, DLMA, CO, and Central task assignment mechanisms for different number of tasks.



(a) Average energy consumption



(b) Average application completion time

Fig. 9. Average energy consumption and application completion time for TAN, DLMA, CO, and Central task assignment mechanisms for different values of the task distribution parameter.

8 Conclusions

In this paper, we propose TAN, a distributed algorithm for dynamic task assignment in heterogenous WSNs, which aims at maximizing network lifetime while ensuring that application completion time is as short as possible. TAN is tested in two reference scenarios: (i) a Smart City environment, and (ii) a set of random scenarios with different characteristics. TAN has shown to outperform gateway-oriented and DLMA-oriented mechanisms in terms of both energy conservation and application completion time. TAN is only outperformed in terms of energy conservation by the tested centralized algorithm, which however is considerably more complex and cannot meet application deadlines due to its long application completion time.

In all scenarios, the average node energy consumption using TAN is found to be inversely proportional to a weighting factor α that strikes the balance between two conflicting goals: a long network lifetime and a short application completion time. In particular, the α value is found to be more critical for scenarios with complex applications, where the application deadline is more likely to be missed.

TAN characteristics make it perfectly suitable not only for WSNs, but also for multi-technology IoT scenarios. Using TAN, the adaptive IoT network components cooperate dynamically in order to achieve optimal performance. Furthermore, with little changes to the task utility function, other requirements such as Quality of Service (QoS) may be optimized.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010.
[2] V. Pilloni and L. Atzori, "Deployment of distributed applications in wireless sensor networks," *Sensors*, vol. 11, no. 8, pp. 7395–7419, 2011.
[3] Y. Jin, J. Jin, A. Gluhak, K. Moessner, and M. Palaniswami, "An intelligent task allocation scheme for multihop wireless networks," *Parallel and Distributed Systems, IEEE Transactions on*, pp. 444–451, 2012.
[4] J. Zhu, J. Li, and H. Gao, "Tasks allocation for real-time applications in heterogeneous sensor networks for energy minimization," in *Proceedings of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, vol. 2, 2007, pp. 20–25.

[5] S. Abdelhak, C. Gurram, S. Ghosh, and M. Bayoumi, “Energy-balancing task allocation on wireless sensor networks for extending the lifetime,” in *Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium on*, 2010, pp. 781 –784.

[6] V. Pilloni, M. Franceschelli, L. Atzori, and A. Giua, “A decentralized lifetime maximization algorithm for distributed applications in wireless sensor networks,” in *Communications (ICC), 2012 IEEE International Conference on*, 2012, pp. 1392–1397.

[7] Y. Shen and H. Ju, “Energy-efficient task assignment based on entropy theory and particle swarm optimization algorithm for wireless sensor networks,” in *Green Computing and Communications (GreenCom), 2011 IEEE/ACM International Conference on*, 2011, pp. 120 –123.

[8] K. Ritzberger, *Foundations of non-cooperative game theory*. Oxford University Press, 2002.

[9] D. Monderer and L. Shapley, “Potential games,” *Games and economic behavior*, vol. 14, pp. 124–143, 1996.

[10] D. Wang, B. Xie, and D. Agrawal, “Coverage and lifetime optimization of wireless sensor networks with gaussian distribution,” *Mobile Computing, IEEE Transactions on*, vol. 7, no. 12, pp. 1444–1458, 2008.

[11] J. N. Al-karaki and A. E. Kamal, “Routing techniques in wireless sensor networks: A survey,” *IEEE Wireless Communications*, vol. 11, pp. 6–28, 2004.

[12] J. Luo and J. P. Hubaux, “Joint mobility and routing for lifetime elongation in wireless sensor networks,” in *INFOCOM 2005. Proceedings IEEE*, vol. 3, 2005, pp. 1735–1746.

[13] J. Luo and J.-P. Hubaux, “Joint sink mobility and routing to maximize the lifetime of wireless sensor networks: the case of constrained mobility,” *IEEE/ACM Transactions on Networking (TON)*, vol. 18, no. 3, pp. 871–884, 2010.

[14] X. Xu and W. Liang, “Placing optimal number of sinks in sensor networks for network lifetime maximization,” in *Communications (ICC), 2011 IEEE International Conference on*, 2011, pp. 1–6.

[15] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “An application-specific protocol architecture for wireless microsensor networks,” *IEEE Transactions on Wireless Communications*, vol. 1, pp. 660–670, October 2002.

[16] D. Wei, Y. Jin, S. Vural, and R. Tafazolli, “An energy-efficient clustering solution for wireless sensor networks,” *IEEE Transactions on Wireless Communications*, vol. 10, no. 11, pp. 3973–3983, Nov. 2011.

[17] Y. Yu and V. K. Prasanna, “Energy-balanced task allocation for collaborative processing in wireless sensor networks,” *Mobile Networks and Applications*, vol. 10, pp. 115–131, 2005.

[18] N. Edalat, W. Xiao, C. Tham, E. Keikha, and L. Ong, “A price-based adaptive task allocation for wireless sensor network,” in *Mobile Adhoc and Sensor Systems, 2009. MASS’09. IEEE 6th International Conference on*, 2009, pp. 888–893.

[19] Q. Li, J. Aslam, and D. Rus, “Hierarchical power-aware routing in sensor networks,” in *In Proceedings of the DIMACS Workshop on Pervasive Networking*, 2001.

[20] Q. Wang and W. Yang, “Energy consumption model for power management in wireless sensor networks,” in *IEEE Sensor, Mesh and Ad Hoc Communications and Networks (SECON) 2007*, 2007, pp. 142 –151.

[21] A. C. Chapman, R. A. Micillo, R. Kota, and N. R. Jennings, “Decentralised dynamic task allocation: A practical game-theoretic approach,” in *Proc. of the 8th International Conference on Autonomous Agents and Multiagent Systems*, vol. 2, 2009, pp. 915–922.

[22] G. Arslan, J. R. Marden, and J. S. Shamma, “Autonomous vehicle-target assignment: A game-theoretical formulation,” *Journal of Dynamic Systems, Measurement, and Control*, vol. 129, no. 5, pp. 584–596, 2007.

[23] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg, “Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks,” *Artificial Intelligence*, vol. 161, no. 1-2, pp. 55 – 87, 2005.

[24] M. Vinyals, J. Rodriguez-Aguilar, and J. Cerquides, “A survey on sensor networks from a multiagent perspective,” *The Computer Journal*, vol. 54, no. 3, pp. 455–470, 2011.

[25] J. P. Pearce and M. Tambe, “Quality guarantees on k-optimal solutions for distributed constraint optimization problems,” in *Proceedings of the 20th international joint conference on Artifical intelligence*, 2007, pp. 1446–1451.

[26] M. Conforti, G. Cornuéjols, and G. Zambelli, “Polyhedral approaches to mixed integer linear programming,” *50 Years of Integer Programming 1958-2008*, pp. 343–385, 2010.

[27] E. Lawler and D. Wood, “Branch-and-bound methods: A survey,” *Operations research*, vol. 14, no. 4, pp. 699–719, 1966.

[28] N. Vlajic and D. Xia, “Wireless sensor networks: to cluster or not to cluster?” in *World of Wireless, Mobile and Multimedia Networks, 2006. WoWMoM 2006. International Symposium on a*. IEEE, 2006, pp. 9–pp.

[29] Chipcon, “Smartrf cc2420 datasheet,” 2.4GHz IEEE 802.15.4/ZigBee-ready RF transceiver.

[30] IEEE, “Std 802.15.4^{TM},” 2006, wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs).

[31] Y. Tian and E. Ekici, “Cross-layer collaborative in-network processing in multihop wireless sensor networks,” *Mobile Computing, IEEE Transactions on*, vol. 6, no. 3, pp. 297–310, 2007.