Cooperative Task Assignment for Distributed Deployment of Applications in WSNs

Virginia Pilloni*, Pirabakaran Navaratnam†, Serdar Vural†, Luigi Atzori*, Rahim Tafazolli†

*DIEE, University of Cagliari, Italy {virginia.pilloni,laatzori}@diee.unica.it
†CCSR, University of Surrey, Guildford, UK {p.navaratnam,s.vural,r.tafazolli}@surrey.ac.uk

Abstract—Nodes in Wireless Sensor Networks (WSNs) are becoming more and more complex systems with the capabilities to run distributed structured applications. Which single task should be implemented by each WSN node needs to be decided by the application deployment strategy by taking into account both network lifetime and execution time requirements. In this paper, we propose an adaptive decentralised algorithm based on non-cooperative game theory, where neighbouring nodes negotiate among each other to maximize their utility function. We then prove that an increment of the nodes utility corresponds to the same increment of the utility for the whole network. Simulation results show significant performance improvement with respect to existing algorithms.

Index Terms—Wireless Sensor Networks, network lifetime, game theory.

I. INTRODUCTION

The great progress in Wireless Sensor Networks (WSNs) nodes’ technology, both in terms of processing capability and energy consumption reduction, has made them more complex systems capable not only of gathering information about the monitored environment, but also of making decisions and acting upon them. These developments have contributed in considering WSNs as one of the pillars of the Future Internet, where they interface with other technologies in order to create an horizontal ambient intelligent infrastructure wherein the sensing, computing and communicating infrastructure is set with a programmable middleware that allows for quickly deploying different applications running on top of it.

Since nodes in a WSN are mainly battery powered, one of the major issues related to WSNs has always been the maximization of the network lifetime. A great effort has been spent by researchers in this problem, taking into account many different approaches. Only recently the attention has been drawn on the development of algorithms that, starting from the partition of applications into small tasks, are able to find more rational task assignments that could help in improving the network performance, be it the improvement of the network lifetime or the reduction of the completion time to perform the application assigned to the network.

In this paper, an adaptive distributed algorithm for the improvement of the network lifetime and the contemporary reduction of the application completion time is proposed. We consider a heterogeneous hierarchical network, where neighbouring nodes negotiate among each other following the rules of non-cooperative game theory to maximize their own utility function. In the following, we will prove that the game under examination is a potential game, which means that every improvement of the nodes’ utility corresponds to the same improvement in the utility perceived by the whole network, and this implies that the problem has a unique outcome that is reachable in a finite time. Simulation results show significant performance improvement for both overall energy consumption and completion time in a typical reference scenario, with respect to gateway-oriented and DLMA-oriented task assignment approach presented in [1].

This paper is organized as follows. The second section provides the preliminaries; the third section introduces the problem and adopted approach; the fourth section describes the task assignment model whereas section five describes the resulting algorithm; the following section presents some simulation results and conclusions.

II. PRELIMINARIES

A. Past Studies

Since nodes in WSNs are mainly battery powered, a great effort has been spent by researchers in finding strategies to increase the nodes’ battery lifetime. Many different approaches have been studied to come to the same goal: convenient deployment of sensors [2], use of efficient routing techniques [3], use of relay nodes that help in balancing the network energy consumption among nodes [4] are some examples.

The increased processing capabilities of modern nodes have made them suitable for more complex applications, such as those of the forthcoming Future Internet [5]. For this reason, network lifetime optimization is not only centred on the reduction of the transmission power anymore, but convenient processing could also reduce the amount of data to be delivered to the Gateway, thus reducing the transmission energy consumption. This is the principle that lies behind LEACH [6], where sensors serve as Cluster Heads aggregating data and, indeed, decreasing the amount of data sent over the network.

A step forward could be taken considering not only aggregation, but also any possible processing task that could be performed on data, on the basis of the network topology, battery power and node processing capabilities. In [7] the maximization of the clusters lifetime is proposed. This approach has two main limitations: it focuses on homogeneous networks, that are not common in real scenarios, and it considers only communication tasks. An adaptive task allocation that aims at reducing the overall energy consumption by maximizing energy balancing was introduce in [8]. However, this mechanism requires the exchange of messages among all the nodes.
in the network, indeed considerably increasing the overhead. [1] proposes an overlaying framework that determines the distribution of tasks among the nodes in a WSN by means of a distributed optimization algorithm aimed at maximizing the network lifetime. However, one major drawback of the previously cited studies is that they do not take into account the deadline of the application assigned to the network.

Completion time reduction is studied in [9], where an algorithm that improves the network lifetime while contemporary reducing the tasks execution time is proposed for a homogeneous network. The algorithm designed in this work is a centralised one, which thus lacks in scalability and dynamism.

In this paper, we propose a framework for a heterogeneous hierarchical WNS, which aims at maximizing the network lifetime and minimizing the application completion time in a decentralized fashion. The nodes of this network negotiate the best solution following the rules of non-cooperative game theory [10]. In the following, we will prove that the proposed algorithm can come to a good solution in a few simple steps.

B. Energy Consumption

There are several mechanisms that contribute in reducing the amount of residual energy for the nodes in a WSN. Since our goal in this work is the convenient assignment of tasks to the nodes, we focus on the mechanisms that are directly related to the execution (or not) of the tasks, that are processing energy consumption and communication energy consumption.

Processing energy consumption $e_{proc}$ depends on the complexity of a task, that is, the number of instructions $I_{task}$ needed to perform it, and on the average energy consumption per executed instruction $e_{instr}(x)$, that is determined on the basis of node $x$ datasheet

$$e_{proc}(task, x) = I_{task}C_{x}$$

There are two main components that contribute in communication energy consumption: transmission and reception energy consumption [11]. We define the energy per bit necessary to send and receive data from node $x$ to node $y$ (one-hop neighbours) at a constant rate $R$, respectively as

$$e_{tx}(x,y) = \frac{1}{R} \left( P_{0,tx} + \frac{\phi_{xy} + \delta_{xy}}{\eta_{x}} \right)$$

$$e_{rx}(y) = \frac{P_{0,rx}}{R}$$

where: $P_{0,tx}$ and $P_{0,rx}$ are the power consumption components of the transmitting and receiving circuitry, for node $x$ and node $y$ respectively; $\eta_{x}$ is the drain efficiency of the Power Amplifier in $x$; $\phi_{xy}$ is a coefficient proportional to the minimum reception power and the characteristic parameters of the antennas; $\delta_{xy}$ is the distance between transmitter and receiver; $\gamma$ denotes the path loss component. These expressions are described in more detail in [11].

III. PROBLEM FORMULATION

The reference scenario considered in this paper is that of a hierarchical heterogeneous WSN. This WSN needs to accomplish a determined application, given by the execution of a certain number of tasks by some of its nodes. Being a heterogeneous network, some nodes can perform the same task faster than others, or spending less energy. The objective of the algorithm described hereinafter is to get an application performed by the network, reducing the processing time and extending the network lifetime at the same time, by assigning the tasks to the nodes of the network in an adaptive and distributed way: whenever a node receives some data, along with the information of which tasks have to be performed on those data, it initiates a negotiation among its neighbours. In this negotiation, based on a non-cooperative game theory approach, each node decides whether it should perform some tasks or not, depending on the contribution it could give to the network in terms of faster processing time and lifetime improvement.

In our modelling, the network can be described as a Directed Acyclic Graph (DAG) $G_{X} = (X, E_{X})$, where the vertices are the nodes $X = \{1, \ldots, i, \ldots, N\}$, while the connections between each pair of nodes $(i, j)$ is described by the set of edges $E_{X} = (e_{ij})$, where each edge represents a connection from node $i$ to node $j$. Since the network is hierarchically organised, a level is associated to each node $i$ in $X$. At the top of the hierarchy is the Gateway (GW), which collects data received from the Cluster Heads (CHs). CHs reach the GW using a Cluster Based routing protocol [12]. Sensor Nodes (SNs) are at the lowest level of the hierarchy. They are grouped in clusters in such a way that each SN is one hop away from all the other SNs constituting the cluster, and from the CH to which it is associated. No communication is allowed between SNs 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 allows to reduce the overhead of the negotiations on the network: when a negotiation is started, all the nodes involved in this negotiation can directly communicate with each other without intermediaries, thus reducing the number of messages exchanged.

Taking into account what has just been said, each node $i$ in $X$ is characterised by:

- its hierarchical level $L(i)$;
- its CH to which it is associated $CH(i)$, with $CH(i) = i$ in case it is a CH;
- the completion time per single instruction $t_{instr}(i)$, that is the time needed by node $i$ to perform the simplest instruction that its microprocessor can execute;
- the energy per single instruction $e_{ins}(i)$, that is the energy spent by node $i$ to perform the simplest instruction that its microprocessor can execute, as defined in Section II-B;
- the vector $e_{tx}(i) = (e_{tx}(i,j))$, where each element $e_{tx}(i,j)$ is the energy per bit spent to send data from node $i$ to an adjacent node $j = X : (i,j) \in E_{X}$, as described in Section II-B;
- the energy per bit spent to receive data $e_{rx}(i)$
- the residual battery capacity $e_{res}(i)$ of node $i$.

We suppose that a given application is assigned to the WSN. Since every application can be decomposed into a set of tasks, it can be described as a Directed Graph (DG) of tasks $G_{T} = (T, E_{T})$, where $T = \{1, \ldots, \lambda, \ldots, \Lambda\}$ is the set
of tasks, while $E_T = (e_{vw})$ is the set of edges, with each edge $e_{vw}$ representing a unidirectional data transfer from task $v$ to task $w$. We suppose that every node in the network knows which is the required application and the relations among the tasks in which it is decomposed. This information could have been given initially to the nodes by the GW.

As to the tasks, a binary strategy vector $s(i) = (s(i, λ))$ can be assigned to each node $i$ in the network (the meaning of strategy will be better explained in Section IV), where $s(i, λ)$ is the current state of node $i$ with reference to task $λ$, that is $s(i, λ)$ is equal to 1 when node $i$ is performing task $λ$. Since the nodes are heterogeneous, different configurations of strategies of all the nodes in the network correspond to different tasks completion times and energy consumption. We will return to this concept in the following sections.

We can distinguish two groups of tasks: already performed and still to be performed. The set of already performed tasks is defined as $T_{prev} = \{1, \ldots, h, \ldots, H}\}$, where each task has already been assigned to a node. This means that, for each task $h$, there is a node $i$ for which its state $s(i, h)$ is equal to 1, and it cannot be changed. We assume that source tasks, that are the tasks that do not receive any input but return an output, are already assigned to the nodes. Hence, at the initial time $t = t_0$, $T_{prev}$ is only populated with source tasks.

Furthermore, we define the set of tasks to be performed $T_{next} = \{1, \ldots, k, \ldots, K\}$, each of them characterised by:

- the deadline for successfully completing a task $t_d(k)$;
- a number of required single instructions $I(k)$.

Not all the nodes have the same abilities to perform some tasks. For this reason, we define a binary vector $d(i) = (d(i, λ))$, where the element $d(i, λ)$ is equal to 1 if and only if node $i$ is able to perform task $λ$. This means that if node $i$ is not able to perform task $λ$, it cannot be assigned to the node: $\forall i \in X$ and $\forall λ \in T$, $d(i, λ) ≥ s(i, λ)$.

IV. THE TASK ASSIGNMENT MODEL

In this section, our task allocation problem will be defined as a non-cooperative game [13]. With reference to our model described in the previous Sections, a non-cooperative game is defined by the tuple $Γ = (X, \{s(i), u_i\}_{i \in X})$: given the strategy vector $s(i)$ assigned to node $i \in X$, a utility function $u_i : s(i) \to \mathbb{R}$ is associated to it. The goal of each node is to maximize its own utility (payoff) in a selfish and rational way. Therefore, a strategy $s^*(i)$ will be preferred to a strategy $s(i)$ if and only if $u_i(s^*(i)) > u_i(s(i))$. For simplicity of notation, we will often use $S = \bigcup_{i \in X} s(i)$ to refer to the strategy of all the nodes in the network.

A. Task Utility Function

Since the utility of a node is strictly bound to the utility for completing or not some tasks, before defining the node utility function, that is the function to be maximized by each node, we first need to define the utility function for a task $k \in T_{next}$ given the overall strategy $S$

$$u_k(S) = \max_{i \in X} \{\Omega_k(i, k) + \alpha \times \Omega_0(i, k, S)\} \times s(i, k),$$

with $s(i, k) ≤ \prod_{l \in T_{in}(k)} \sum_{j \in X} s(j, l)$  

where: $\Omega_k(i, k)$ is the component of the utility function related to the completion time of task $k$ when it is performed by node $i$; $\Omega_0(i, k, S)$ is the component related to the lifetime of the network when task $k$ is performed by node $i$, according to the nodes’ strategy $S$; $\alpha > 0$ is a weighting factor that takes into account any benefit of improving the network lifetime with respect to the completion time (we will further explain the meaning of $\alpha$ in the following); $T_{in}(k) = \{λ \in E_T : \exists \epsilon_λk\}$ is the set of tasks which output is needed by task $k$ as input (i.e., previous tasks for task $k$). We recall that $s(i, k)$ is the current state of node $i$ with reference to task $k$. The constraint on $s(i, k)$ ensures that if any of task $k$’s previous tasks is not performed by any node, that is if any of its inputs is not available, there is no utility in performing task $k$, and therefore $s(i, k)$ cannot be equal to 1. The maximum ensures that the task can only be performed by the node that maximizes the utility function $u_k(S)$: since the goal of the game is to maximize the utility function, that is maximizing the lifetime due to the execution of some tasks, only the node that ensures the best outcome will be chosen to perform task $k$.

We only define a utility function for the tasks that still need to be performed: since performed tasks are already assigned and this assignment cannot be changed, there is no need to evaluate their utility.

1) Completion Time Component: We express the completion time component as

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

where $t_c(i, k)$ is the completion time if task $k$ is performed by node $i$ and $t_d(k)$ is the deadline for successfully completing task $k$. Calling $I(k)$ the number of single instructions for task $k$ and $t_{instr}(i)$ the completion time per single instruction for task $i$, as defined in Section III, $t_c(i, k)$ is defined as

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

2) Lifetime Component: The lifetime component is defined as follows

$$\Omega_0(i, k, S) = F_p(i, k) + F_{tx}(i, k, S)$$

where $F_p(i, k)$ is the component related to the changing in lifetime due to the processing needed to perform task $k$ in node $i$, while $F_{tx}(i, k, S)$ is related to the changing in lifetime due to transmission (and reception) in case task $k$ is performed in node $i$ with strategy $S$.

We define the processing component as

$$F_p(i, k) = -I(k) \times \frac{e_{ins}(i)}{e_{res}(i)}$$

with $I(k)$ number of single instructions for task $k$, $e_{ins}(i)$ energy per single instruction for node $i$, and $e_{res}(i)$ residual battery capacity for node $i$, as introduced in Section III. Since processing entails an energy consumption, $F_p(i, k)$ needs to be negative in order to decrement the payoff due to the execution of task $k$.

Let $p_{i \to j} = \{(i, a), (a, b), \ldots, (c, f), (f, j)\}$ be the sequence of edges in the path that connects node $i$ to node $j$.
considering a difference in lifetime between the case where $k$ receives data from node $x$ and the task utility function (3) and making it more convenient to perform the task as its strategy. The node utility function is then defined as

$$u_i(s(i), s(-i)) = \sum_{k \in T_{next}} m u_k(s(i), s(-i))$$

From (9) we can infer that the marginal contribution of the node to a task is not null only if the node is contributing to the task. This means that, since the global utility function in (8) is a summation of task utility functions (3), if the node contributes to the task, it follows that it contributes to the global utility. In other words, a change in strategy for node $i$ that increases its utility, entails the same increment in the global utility. Let us suppose that $s(i)$ is the strategy of node $i$ at time $t$ and $s^*(i)$ is the strategy of node $i$ at time $t^* > t$

$$u_i(s^*(i), s(-i)) - u_i(s(i), s(-i)) = \sum_{k \in T_{next}} m u_k(s^*(i), s(-i)) - \sum_{k \in T_{next}} m u_k(s(i), s(-i)) = \sum_{k \in T_{next}} \left( u_k(s^*(i), s(-i)) - u_k(s(i), s(-i)) \right) = u_g(s^*(i), s(-i)) - u_g(s(i), s(-i))$$

This property is particularly desirable because it implies that the game we are focusing on is a potential game, where the potential function is given by the global utility function. A consequence of this is that this game has at least one pure Nash equilibrium [15]: pure Nash equilibria are characterised by a unique outcome, contrary to mixed Nash equilibria where the outcome is stochastically variable. Furthermore, another property of potential games is the Finite Improvement Property (FIP): every sequence of changes in strategy that improves the global utility, converges to a Nash equilibrium in finite time. The FIP ensures that many simple adaptive processes, such as the Distributed Stochastic Algorithm (DSA) [16], converge to Nash equilibria.
V. PROPOSED ALGORITHM

Given the network described in Section III, in order to improve the network lifetime while reducing the completion time for the whole application, the global utility function defined in (8) should be maximized. As demonstrated in Section IV-C, a change in the node utility reflects in the same change in the global utility. This means that maximizing the node utility function for all the nodes in the network, would reflect in a maximisation of the $u_g(S)$. In order to find the best outcome, a greedy local search algorithm such as the DSA is proposed. We use DSA because, with respect to other similar algorithms, it is proved to come to a solution quicker [17].

DSA is a synchronous algorithm; in our case, during the DSA execution the involved nodes negotiate the best strategy to maximize the global utility. At each time step, each node involved in the negotiation, that changed its strategy in the previous time step sends, to all the other involved nodes, a strategy update message with its new strategy [16]. If a node receives any strategy update message, it has some probability $p$ of activation to compute a new strategy that maximizes its utility (10). If the utility is already maximized by the current strategy, no changes occur. The probability of activation $p$ is known as the degree of parallel executions. A particular attention has to be taken in choosing $p$ value: the higher it is, the higher the probability that all the nodes change their strategy at the same time, thus increasing the overhead due to the algorithm, and of course its convergence time; on the other hand, low values of the degree of parallel executions could lead to an unacceptable increment of DSA’s convergence time as well, because changes of strategy would be less frequent.

It seems evident that a negotiation of this type, that involves all the nodes in the network at the same time, would require exceedingly high communication costs and convergence time. For this reason, we propose a suboptimal algorithm, called Task Allocation Negotiation (TAN), where the nodes negotiate only with their neighbours (remember that node $i$’s neighbours are its adjacent nodes, as defined in Section IV-B). More specifically, if the negotiation is started by a SN, its neighbours are all the nodes belonging to its same cluster; if the negotiation is started by a CH, its neighbours are its adjacent CHs. The basic idea is this: at the initial time $t = t_0$, no node is performing any task, hence the global utility is $u_g(S) = 0$; the source tasks are allocated to the appropriate nodes. As soon as a node performs a source task, it has some data to process. Let us suppose that node $i$ has just performed a source task. Remembering that each node knows the relations among the tasks in which the application assigned to the network is decomposed, as said in Section III, if there is any processing task that can be performed on data, node $i$ starts the negotiation’s initialisation by sending an information message to all its neighbours. Information messages include the node’s strategy vector $s(i)$ (since no strategy has been negotiated yet, $s(i)$ is still a null vector), its energy per single instruction $e_{ins}(i)$, its vector $e_{tx}(i)$ of energy per bit spent in transmission, and its residual battery capacity $e_{res}(i)$ (we refer to Section III for the definitions of these parameters), along with the identification code of the task just performed. Node $i$’s neighbours reply broadcasting an information message of their characteristic parameters, as well. The DSA is then run for each node involved in the negotiation. At each time step, if the node has not received a strategy update message, it increments a counter; otherwise, this counter is set to 0, and the node chooses the strategy that maximizes its utility function (10). When this counter reaches $t_{wait}$ since the last received strategy update message, the halting criteria for DSA is satisfied. Since all the nodes involved in the negotiation are connected to each other, as described in Section III, they all know at the same time if a strategy update message has been sent or not; therefore, they all meet the halting criteria at the same time without the need for further synchronization. If the strategy of node $i$ that performed the source task has changed, it performs the assigned processing tasks, and then, if no other node has changed its strategy, it sends a data message to its CH. Data messages contain the identification codes of the last performed tasks, along with the resulting data. If any node is scheduled to perform a processing task on those data, node $i$ sends them a data message. This process is repeated until all the allocated tasks are executed. Eventually, data are sent to the CH, which in turn starts a negotiation with its neighbours. If a negotiation involving this CH has already been started, the CH already knows the characteristic parameters of its neighbours by their information messages. Hence, it broadcasts an information message to let its neighbours know about the identification codes of the data it has just received, but they do not reply to it. The TAN stops when data reaches the GW.

VI. PERFORMANCE ANALYSIS

The proposed algorithm has been tested by means of simulations carried out for a realistic heterogeneous WSN application in a smart city context. The scenario under examination is that of an urban environment, where nodes have been positioned along the streets as shown in Figure 1. Solid markers represent nodes equipped with sensors for speed measurement of vehicles passing through; speed measurements are represented as double numerical values (64-bit long). Every stretch of street is a cluster, where the CHs are represented by the empty markers. They are more capable nodes (with an initial battery charge three times higher than the others), that do not perform any sensing task, but they are able to perform all the other tasks for the appropriate stretches.

![Fig. 1. Example of topology for Scenario B. Solid markers represent nodes equipped with speed sensors, while empty markers are more capable nodes which do not perform any sensing task, but can perform more complex processing tasks](image-url)

We supposed that a driver, placed in the Start point, would like to know which is the fastest way to reach Destination,
based on the speed information collected by sensor nodes. We identified 89 different speed sensing tasks ($\lambda_1 \div \lambda_{89}$), one for each sensing node. Since we need to know the mean travelling time for each stretch of street in order to confront them to each other, we defined 78 mean speed computing tasks ($\lambda_{90} \div \lambda_{157}$), and 11 mean travelling time computing tasks ($\lambda_{158} \div \lambda_{168}$), one for each stretch of street. To find the best path, that is the best combination of stretches of streets that leads from the Start point to Destination, we then need to sum the mean travelling times of all the combination of stretches that can be driven one after the other, and confront them. Therefore, the remaining tasks are: 8 different summation of mean travelling times for different stretches ($\lambda_{179} \div \lambda_{186}$) and 3 different choice of the best path ($\lambda_{187} \div \lambda_{189}$). With reference to Figure 1, solid markers represent nodes that are only allowed to perform the speed sensing and mean speed computing for the stretch of the street where they are placed (8 in total).

Nodes communicate using IEEE 802.15.4 radio interfaces on the 2.4 GHz ISM frequency band. We simulated the described scenario in a MatLab environment, where we implemented TAN algorithm along with two alternative approaches: all the data sent to the GW and processed only by the GW, that is the Start node (that is the centralised mechanism that we refer to with C); data processed according to DLMA algorithm described in [1] (DLMA approach).

The obtained results for energy consumption and completion time have been compared to those obtained using TAN algorithm. Table I shows the percentage of energy conservation and completion time gained when using TAN with respect to the alternative approaches. We refer to these comparisons as TAN-C and TAN-DLMA.

We ran simulations for: $\alpha = 0$ (null lifetime component); $\alpha = \bar{\Omega}_t/\bar{\Omega}_r$ (comparable $\bar{\Omega}_t$ and $\bar{\Omega}_r$, where $\bar{\Omega}_t$ and $\bar{\Omega}_r$ are calculated as the mean values of $\Omega_t$ and $\Omega_r$ when every node is performing every task at the same time); $\alpha = \infty$ (null completion time component).

We observe a marked improvement of energy conservation and a good improvement in completion time gain with respect to the centralised mechanism. Good results are also observable both for energy conservation and completion time gain with respect to TAN-DLMA. It has to be noted that results for $\alpha = 0$ and $\alpha = \bar{\Omega}_t/\bar{\Omega}_r$ in comparison TAN-DLMA have been reported just for completeness, but DLMA algorithm does not take into account completion time, therefore the most significant results are those obtained for $\alpha = \infty$.

As we expected, when $\alpha$ increases, $u_\alpha(S)$ is more over-balanced in favour of its lifetime component, and thus energy conservation percentage increases, while the completion time gain decreases; on the contrary, when $\alpha$ decreases, $u_\alpha(S)$ is over-balanced in favour of its completion time component: the energy conservation percentage decreases, while the completion time gain increases.

VII. Conclusion

The algorithm presented has proven to overcome the results obtained for both C and DLMA mechanisms, and both energy conservation and completion time gain. Since results are slightly different for different values of $\alpha$, it should be chosen according to the application deployment requirements.

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>PERCENTAGE VALUES OF ENERGY CONSERVATION AND COMPLETION TIME GAIN USING TAN, FOR COMPARISONS TAN-C AND TAN-DLMA</th>
</tr>
</thead>
<tbody>
<tr>
<td> </td>
<td>TAN-C [%]</td>
</tr>
<tr>
<td>$\alpha = 0$</td>
<td>$\alpha = \bar{\Omega}_t/\bar{\Omega}_r$</td>
</tr>
<tr>
<td>Energy conservation</td>
<td>73.6</td>
</tr>
<tr>
<td>Completion time gain</td>
<td>22.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REFERENCES</th>
</tr>
</thead>
</table>