Single and Group Agent Migration: Algorithms, Bounds, and Optimality Issues

Nikos Tziritas, Samee Ullah Khan, Member, IEEE, Thanasis Loukopoulos, Spyros Lalis, Cheng-Zhong Xu, Senior Member, IEEE, Petros Lampsas

Abstract— Recent embedded middleware platforms enable the structuring of an application as a set of collaborating agents deployed on various nodes of the underlying wireless sensor network (WSN). Of particular importance is the network cost incurred due to agent communication, which in turn depends on how the agents are placed within the WSN system. In this paper, we present two agent migration algorithms with the aim of minimizing the total network overhead. The first one takes independent single agent migration decisions, while the second one considers groups of agents for migration. Both algorithms work in a fully distributed fashion based on the knowledge available locally at each node, and can be used both for one-shot initial application deployment as well as for the continuous updating of agent placement. We also propose two methodologies to tackle the problem when WSN nodes have limited capacity. We show through theoretical analysis that one of our algorithms (called GRAL*) always results in an optimal placement, while for the rest of the algorithms, we derive approximation ratios pertaining to their performance. We evaluate the performance of our algorithms through a series of simulation experiments. Results show that group migration algorithms are superior compared to single agent migration algorithms with the performance difference reaching 34% for some settings.

Index Terms—Embedded systems, sensor and actuator networks, agent placement, distributed algorithms.

1 Introduction

Wireless sensor networks (WSNs) have been a vibrant field of research during the last several years. Many advances have been made across a wide range, from the design of battery-powered embedded nodes and ad-hoc network protocols to operating systems and middleware offering application-neutral communication and execution support on top of such resource-constrained devices. One of the most challenging issues for hardware, protocol designers and software engineers is to maximize the lifetime of the WSN and the availability of the application(s) running on top of it. Given the relatively high energy cost of wireless data transmission and reception compared to data processing, reducing the frequency of communication and the amount of data that travels through the network is of primary importance [1].

Tackling this challenge, we discuss algorithms that drive the migration of mobile components between nodes so as to reduce the wireless network traffic due to application-level communication. The application model adopted in this work is inspired by the POBICOS [32] platform, and its predecessor ROVERS [12], which supports the deployment and execution of distributed monitoring-and-control applications on top of a WSN. In the spirit of hierarchical control systems [25], the application is structured as a set of cooperating mobile components, called agents, which are organized as a tree. The agent tree is instantiated in a top-down fashion under the control of the application, with the underlying runtime system being responsible for the actual placement of agents on nodes. Given that the tree structure and message traffic between agents is not a priori known to the sys-

Digital Object Indentifier 10.1109/TC.2013.166

tem, the initial placement can be suboptimal in the sense that agents which communicate heavily with each other may be hosted on nodes that are far apart (hop-wise). Moreover, even if the initial placement is good, it may become arbitrarily bad at a later stage due to changes in the agent tree and/or agent communication pattern. Therefore, to maximize the lifetime of the WSN, the system must be able to dynamically change the placement of agents whenever necessary.

In this paper, we assume that the runtime system supports the transparent (to the application programmer) migration of agents, as is the case with POBICOS [35], and focus on the algorithmic aspects of *deciding whether and where to move one or more agents*. More specifically, we introduce the *agent migration problem* (AMP) as: *Given an application that is structured as a hierarchy of mobile cooperating agents and its current deployment on a WSN, perform one or more agent migrations so as to reduce the amount of data exchanged between agents over the links of the WSN*. We discuss two fully distributed algorithms that move node-neutral agents towards their center of (communication) gravity, thereby reducing the total amount of application-level data that traverses through the WSN. We must note that both algorithms are based on tree-structured application and network graphs, and consider only non-leaf agents for migration (leaf agents remain fixed on the nodes where they were initially created). The first algorithm, referred to as single agent migration algorithm (AMA), considers only migrations of individual agents. The second algorithm, referred to as group agent migration algorithm (GRAL), considers migrations of entire agent groups to capture "communication dependencies" between agents. In both cases, it is assumed that the agent tree structure and communication rate between agents change slowly enough for the migration overhead to be amortized. Therefore, these algorithms target relatively "stable" systems, such as buildings and home automation installations where wireless sensors and actuator nodes are added, moved, or removed in an ad-hoc yet infrequent[1] fashion. While such changes may be rare, they can render the current agent placement bad, which in turn can have a significant negative impact on the WSN, precisely because the system may remain in this state for a very long time.

The core design concepts of AMA and GRAL are discussed in [40] and [39], where their performance is studied via simulation experiments. Here, we extend previous work by providing a rigorous theoretical analysis on AMA and GRAL performance, deriving their approximation ratios compared to the optimal algorithm. Results indicate that the worst-case performance of AMA is unbounded, while GRAL is suboptimal. Based on the analysis, we extend GRAL (GRAL*) so as to become provably optimal when there is no capacity limitation on the nodes. (Otherwise, the problem is NP-complete [37]). To our best knowledge, there is no work in the literature that considers groups of agent migrations. Our solution is especially valuable given that we prove that solutions based on single agent migrations are not bounded.

The remainder of this work is structured as follows: In Section 2 we present the system model and rigorously formulate the optimization problem. The proposed agent migration algorithms are described in Section 3, while Section 4 discusses extensions for dealing with capacity constraints and exploiting increased network awareness. Section 5 provides a theoretical analysis about

[1] The design of online algorithms that can handle dynamic systems is discussed in a separate paper [36].

the approximation ratios of our algorithms, along with a modification of GRAL that is provably optimal. The experimental evaluation is described in Section 6. Section 7 reports the related work, and Section 8 concludes the paper.

2 Problem Definition

2.1 Application Model

We focus on sensing and context inference applications that are structured as a hierarchy of cooperating agents. As an example, consider the simplified logic of an application that infers the presence of fire, based on the measurements of temperature and humidity, shown in Fig. 1. One possible structure for this application, using a separate agent for each of the respective sensing and context aggregation tasks, is depicted in Fig. 2. Leaf agents measure temperature and humidity in each room. At the next level, two aggregator agents compute the average room temperature and humidity values, respectively, and report these values to the alarm agent for that room. In turn, each room alarm agent processes these values, and notifies the root if a fire is inferred. The root agent waits to receive a notification from one of the room alarm agents, in which case, it alerts the user.

```
while(true)
  for each room R
    if avg("temp",R)>X && avg("hum",R)<Y
      notify_user();
```

Figure 1. Application code

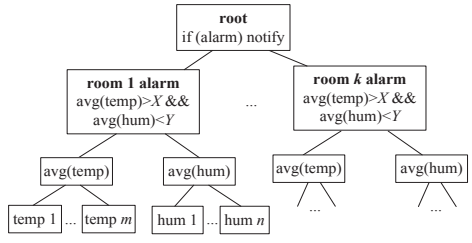

Figure 2. Application structure

Leaf agents, such as the agents measuring temperature and humidity, can only be created on nodes that feature the appropriate sensors and are located in the proper rooms. For this reason, these agents are called "node-specific". Intermediate agents, such as the temperature and humidity aggregators or the room alarm agents, perform higher-level aggregation and context synthesis tasks relying on generic and location-independent computing resources. Since such resources are provided by all nodes, these agents are called "node-neutral". Node-specific agents remain fixed on the nodes where they are initially created. In contrast, a node-neutral agent can be moved from one node to another in order to: **(a)** offload its old host, or **(b)** get closer to the agents that it communicates with more intensely. The latter creates a significant optimization potential in terms of reducing the wireless network traffic due to agent-level communication.

2.2 System Model

We assume a WSN of capacitated nodes with sensing capabilities and applications to be deployed on them. Let n_i denote the i^{th} node, $1 \leq i \leq N$ and $r(n_i)$ its capacity in terms of processing power, memory, and/or even bandwidth. The capacity of a node imposes a general constraint on the number of agents that it is able to host. Nodes communicate with each other via some wireless technology (which we treat as a black box). The underlying routing paths are abstracted as a graph, its vertices representing nodes and each edge representing a bidirectional routing-level link. Here, we consider *tree-based routing*, i.e., there is exactly one path connecting any two nodes. Let h_{ij} be the length of the path between n_i and n_j. Note that $h_{ii}=0$ and $h_{ij} = h_{ji}$. Each applica-

tion has the form of an agent tree, with the leaf agents being node-specific and all of the other agents being node-neutral, as discussed above. Assuming an enumeration of agents whereby node-neutral agents come first, let a_k be the k^{th} node agent, $1 \le k \le A+S$, whereby A is the total number of node-neutral and S the total number of node-specific agents. Let $r(a_k)$ be the capacity requirement of a_k. Moreover, let the agent-level traffic be captured using an $(A+S)\times(A+S)$ matrix C, where C_{km} denotes the load from a_k to a_m, measured in simple data units (bytes) over a time period (seconds). Note that C_{km} need not be equal to C_{mk}, and that the total communication load between a_k and a_m equals $C_{km}+C_{mk}$. Moreover, $C_{km}=0$ when $k = m$.

For the sake of generality, we focus on the case where the application is already deployed on the WSN, with each agent being hosted on some suitable node (however, the current placement may not be optimal). Let P be an $N\times(A+S)$ matrix used to encode the placement of agents on nodes as follows: $P_{ik}=1$ iff n_i hosts a_k, and 0 otherwise. The total network load L incurred by the application for a placement P can then be expressed by Eq. 1. For a placement P to be valid, each agent must be hosted on exactly one node and the node capacities must be respected as denoted by Eq. 2 and Eq. 3. Moreover, a migration is valid only if starting from a valid placement P it leads to another valid placement P' with node-specific agents on their original hosts (see Eq. 4).

$$L = \sum_{k=1}^{A+S}\sum_{m=1}^{A+S} C_{km} \sum_{i=1}^{N}\sum_{j=1}^{N} h_{ij} P_{ik} P_{jm} \qquad \text{Eq. 1}$$

$$\sum_{i=1}^{N} P_{ik} = 1, \forall 1 \le k \le A+S \qquad \text{Eq. 2}$$

$$\sum_{k=1}^{A+S} P_{ik} r(a_k) \le r(n_i), \forall 1 \le i \le N \qquad \text{Eq. 3}$$

$$P_{ik} = P'_{ik}, \forall A < k \le A+S \qquad \text{Eq. 4}$$

The agent migration problem (AMP) can then be stated as: *Starting from an initial valid agent placement P_{old}, perform a series of valid agent migrations, eventually leading to a new valid placement P_{new} that minimizes Eq. 1.* Note that the solution to AMP may be a placement that is actually suboptimal in terms of Eq. 1. This is the case if the optimal placement is "unreachable" due to Eq. 3: when it can be reached only by "swapping" agents but there is not enough free capacity to perform these swap(s). A similar issue is discussed in [31] in a slightly different context. It is also important to stress that Eq. 1 does not take into account the cost for performing the migrations needed to reach a given placement. This is because we target applications whose agent structure and communication pattern is expected to be sufficiently stable so as to amortize the migration cost.

Table I. Notations

n_i	ith node	$r(n_i)$	capacity of n_i
N	total number of nodes	h_{ij}	length path between n_i and n_j
a_k	k^{th} agent	A_G	hyper-agent representing agents belonging into group G
$r(a_k)$	capacity requirements of a_k	$h(a_k)$	hosting node of a_k
A	total number of node-neutral agents	S	total number of node-specific agents
C_{km}	load between a_k and a_m (in bytes)	L	total network load
P_{ik}	captures whether n_i hosts a_k (boolean)	$m_{sd}(k)$	migration of a_k from n_s to n_d
$R_{dk}(i)$	Records whether n_i is used for the communication between a_d and a_k	$dl_{sd}(k)$	*disadvantageous load* when a_k migrates from n_s to n_d (single-hop migration)
$al_{sd}(k)$	*advantageous load* when a_k migrates from n_s to n_d (single-hop migration)	$pg_{sd}(k)$	*partial migration gain (*single-hop migration*)* when a_k migrates from n_s to n_d (the agents communicating with a_k remain on n_s)
$p(a_k)$	parent agent of a_k	$g_{sd}(k)$	*gain* when a_k migrates from n_s to n_d (single-hop migration)
$mdl_{sd}(k)$	disadvantageous load (multi-hop migration)	$mal_{sd}(k)$	advantageous load (multi-hop migration)
$mpg_{sd}(k)$	partial migration gain (multi-hop migration)	$mg_{sd}(k)$	the total gain of migrating a_k along a path starting from some node n_s and ending on some other node n_d.
t_m	group agent node	q_{km}	data exchanged between a_k and a_m

D	network diameter	$B/2$	maximum data an agent can send towards another one
$rl(a_k)$	remote load exchanged between a_k and the agents located on a node different than $h(a_k)$	$ll(a_k)$	local load exchanged between a_k and the agents co-located with a_k
T	maximum number of times the agents can send B/2 data units over the network	$f_{sd}(z)$	represents the z^{th} node along the path starting from n_s and ending on n_d
$cpg_{su}^{sv}(m,k)$	*combined partial gain* after merging t_k and t_m, under the assumption that a_m migrates from n_s at n_v, while a_k from n_s to n_u.		

3 Algorithms

This section presents two agent migration algorithms. The proposed algorithms require that each node knows its immediate (1-hop) neighbors involved in transporting inbound and outbound agent messages. We refer to the above as *1-hop network awareness.* Each node within the system is responsible to determine whether it is beneficial or otherwise to migrate one or more of its hosted agents towards its neighbors. An agent may nevertheless move to distant nodes via consecutive 1-hop migrations.

3.1 Simple Agent Migration Algorithm (AMA)

The proposed algorithm (hereafter called AMA or AMA-1) considers only simple 1-hop agent migrations to reduce the total WSN overhead. Each AMA-enabled node supports a *traffic recording mechanism*. Therefore, each node within the system is able to record the : **(a)** *local traffic* (per communicating agent) that is the amount of data exchanged between its hosted agents and **(b)** *remote traffic* (per communicating agent) associated with the hosted agents and the 1-hop neighboring nodes of the node in question. The above information is adequate for a node to predicate whether it is beneficial to migrate or otherwise a locally hosted agent towards a 1-hop neighboring node. The variable $R_{dk}(i)$ records whether n_i is used or not for the communication between a_d and a_k. Specifically, $R_{dk}(i) = 1$ iff n_i is used for communication between a_d and a_k, else $R_{dk}(i) = 0$. In a special case where a_d and a_k are co-located, then we have $R_{dk}(i) = 0$, irrespective of whether n_i is used for the communication or not.

Specifically, the algorithm works as follows. Each node within the system runs the traffic recording mechanism. In that way, a node is able to identify whether the remote traffic associated with a locally hosted node-neutral agent can be alleviated by migrating the agent in question to a neighboring node. A candidate agent migration is denoted by $m_{sd}(k)$, with a_k denoting the candidate migrating agent, n_s the current hosting node (or *local node*), and n_d the destination node.

$$dl_{sd}(k) = \sum_{m=1}^{A+S} (C_{km} + C_{mk})(1 - R_{km}(d)) \quad \text{Eq. 5}$$

$$al_{sd}(k) = \sum_{m=1}^{A+S} (C_{km} + C_{mk}) R_{km}(d) \quad \text{Eq. 6}$$

$$g_{sd}(k) = al_{sd}(k) - dl_{sd}(k) \quad \text{Eq. 7}$$

$$\sum_{a_k \in G} al_{sd}(k) > \sum_{a_k \in G} dl_{sd}(k) \quad \text{Eq. 8}$$

For a node to decide whether an agent migration is advantageous, it has to evaluate the resulting (new) placement compared to the current (old) one. Let the *disadvantageous load* denote the extra load the application would incur on the WSN assuming the new placement. This is captured by Eq. 5: If a_k migrates from n_s to n_d, the extra load equals the volume of data exchanged between a_k and the agents not using n_d to communicate with n_s (a_k will distance itself by 1 hop from those agents). Similarly, let the *advantageous load* denote the application load that would no longer burden the WSN assuming the new placement. This is captured by Eq. 6: If a_k migrates from n_s to n_d, the load saved equals the volume of data exchanged between a_k and the agents

using n_d to communicate with n_s (a_k will come 1 hop closer to those agents). The difference between the advantageous load and the disadvantageous load reflects the *gain* of the respective migration (Eq. 7). If the gain is positive, then the migration is considered *advantageous*, otherwise it is considered *non-advantageous*.

AMA allows different agent migrations to be performed concurrently, but prevents the swapping of communicating agents. More specifically, before deciding to actually migrate a_k, a control message is sent to the destination node in order to check whether a locally hosted agent that communicates with a_k has been picked for migration to a_k's host. If this is the case, the migration of the child agent is cancelled. This introduces an overhead of two control messages (request-reply) per migration. The space and time complexity of the algorithm is discussed in [40].

3.2 Group Agent Migration Algorithm (GRAL)

GRAL (hereafter also referred to as GRAL-1) is a completely different approach compared to AMA. GRAL considers 1-hop migrations in a group-wise fashion by taking into account the agent dependencies. Conversely, AMA migrates individual agents. GRAL is also equipped with the same recording mechanism as that of AMA. Each node within the system running GRAL is able to identify disjoint application sub-trees hosted locally on the respective nodes (using also partial information of the application tree). For each identified sub-tree, a group of agents is produced that may be a subset of the sub-tree. For each group, a single destination is chosen as a host for all of the agents that are part of the group. Initially, the algorithm performs group agent migrations, and in the sequel single agent migrations. The single agent migrations are identified in the same procedure as that of AMA. The group agent migrations are identified as follows:

Sub-tree Identification. First, one or more disjoint sets of communicating locally hosted node-neutral agents (belonging to the same application) are identified. Each such set corresponds to a part of the application tree; hereafter, referred to as a *sub-tree*. Specifically, the sub-tree identification takes place as follows: **(i)** create a sub-tree rooted on a locally hosted node-neutral agent not belonging to an already identified sub-tree and **(ii)** add to the respective sub-tree each locally hosted node-neutral agent that is adjacent (according to the application graph) to one of the agents belonging already to that sub-tree. Repeat phase (ii) until no agent can expand the respective sub-tree. After the expansion of a sub-tree completes, repeat phase (i) and (ii) accordingly, until all of the node-neutral agents have been considered.

Selection of destination. For each sub-tree, the most *promising 1-hop destination node* is determined by subtracting collectively the disadvantageous loads from the advantageous ones in terms of the group of agents (denoted by G) belonging to the corresponding sub-tree. If the resultant is positive, then the corresponding node is considered as the most promising 1-hop destination node for the group of agents belonging to the respective sub-tree. Specifically, Eq. 8 must hold true to select n_d as the most promising 1-hop destination node for a sub-tree G hosted at n_s.

Partial migration gain calculation. Having chosen the best promising 1-hop destination n_d, the respective *partial migration gain* value is computed for each agent a_k of the corresponding sub-tree as:

$$pg_{sd}(r) = al_{sd}(r) - dl_{sd}(r) \mid root = a_r \quad \text{Eq. 9}$$

$$pg_{sd}(k) = al_{sd}(k) - dl_{sd}(k) + 2\times(C_{km} + C_{mk}) \mid p(a_k) = a_m \quad \text{Eq. 10}$$

The partial migration gain is calculated for each agent in a top-down fashion. Eq. 9 is used to calculate the partial gain of migrating the root a_r of a sub-tree from n_s to n_d. The above partial gain corresponds to the actual gain of that migration, provided that the agents of the corresponding sub-tree communicating with a_r remain on n_s. To calculate the partial migration gain of every other agent a_k of a sub-tree we make use of Eq. 10, given that the parent of a_k is a_m (in terms of the corresponding sub-tree). Specifically, Eq. 10 corresponds to the load impact if both a_k and its parent a_m (denoted by $p(a_k) = a_m$) migrate to n_d, on the premise that the rest of the agents of the corresponding sub-tree that communicate with a_k remain on n_s. The third term in Eq. 10 is justified by the fact that when calculating the partial migration gain of an agent, there is an assumption that the corresponding agent is migrated along with its parent. Therefore, we must add the traffic load between the respective agent and its parent. However, we can observe that: **(a)** the disadvantageous load of a migrating agent belongs to the negative part of Eq. 10 and **(b)** it contains the traffic load associated with the parent of the respective agent. Therefore, we must double the traffic load between the respective agent and its parent. The actual gain for migrating any agent a_k together with all its predecessors (in the path) up to the root a_r is equal to the sum of the respective partial migration gain values.

Sub-tree contraction. The algorithm processes a sub-tree by performing iteratively in a bottom-up fashion prunes/merges of the leaves of the corresponding sub-tree. Specifically, if the partial migration gain of a leaf is negative, then we prune the respective leaf. Otherwise, we choose to merge the leaf with its parent by aggregating their partial migration gains. Each merge produces a so-called *contracted node* with a respective partial migration gain. The sub-tree contraction phase terminates when either all of the tree nodes have been pruned or it results in a single contracted node (called *final contracted node*) that represents a group agent migration.

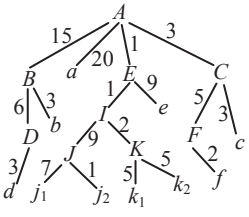

Figure 3. Application structure

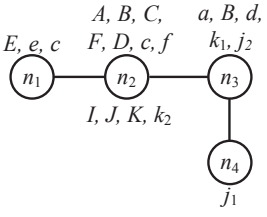

Figure 4. Agent placement

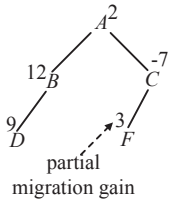

Figure 5. Partial migration gains for the sub-tree (*A*,*B*,*C*,*D*,*F*)

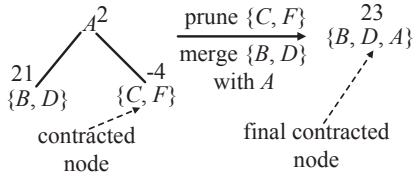

Figure 6. Sub-tree contraction phase for the sub-tree (*A*,*B*,*C*,*D*,*F*)

As an example, consider the application tree shown in Fig. 3, whereby generic agents are denoted in capitals and non-generic agents in small case letters. Edge values stand for the communication load between two agents. Let the application be deployed on a network as illustrated in Fig. 4. Two disjoint sub-trees are hosted at n_2: *(A, B, C, D, F)* and *(I, J, K)*. Therefore, two groupings will be produced, one for each sub-tree. (Note that AMA cannot improve the placement depicted in Fig. 4). By taking into account the communication loads between agents, we identify through Eq. 8 that n_3 is the most promising 1-hop destination for both sub-trees *(A, B, C, D, F)* and *(I, J, K)*. In the sequel, we illustrate in Fig. 5 the calculation of the partial migration gains of the first sub-tree. As stated previously, the above calculation takes place in a top-down fashion. Therefore, we first calculate the *p*artial gain ($pg_{23}(A)$) for the migration of agent *A* from n_2 to n_3. The above partial migration gain equals to two because Eq. 9

must be used for the above calculation (note that agent A is the root of the sub-tree under consideration). We should also note that for the next levels of the sub-tree under consideration, we calculate sub-tree's partial migration gains by making use of Eq. 10. Specifically, for the second level of the tree, we calculate the partial migration gains of agent B and agent C that being 12 and -7, respectively. For the last level, the partial migration gains of agent D and agent F are equal to 9 and 3, respectively. In Fig. 6, we show the tree contraction phase taking place in a bottom-up fashion. Because the partial gains of the leaves D and F appearing in Fig. 6 are positive, we choose to merge each one with its parent (left part of Fig. 6). In the sequel, we observe that the partial gain of the new leaf $\{B, D\}$ is positive, while the other one $\{C, F\}$ is negative. Therefore, $\{B, D\}$ is merged with its parent A, while $\{C, F\}$ is pruned (negative contribution). Finally, we result in the migrating group $\{B, D, A\}$, with its impact (migration gain) within the system equaling 23 (right part of Fig. 6).

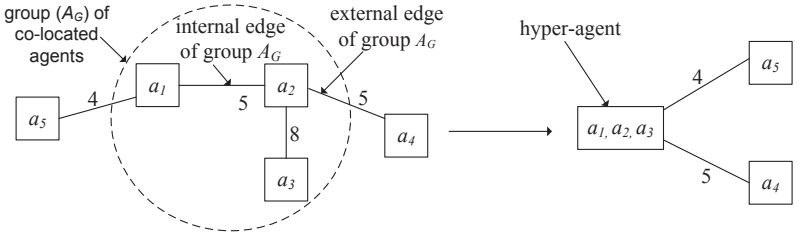

Figure 7. Constructing a hyper-agent from a group of three agents

Hyper-agent: According to the following procedure, a group of co-located node-neutral agents (called G) can be transformed into a hyper-agent (called A_G). The transformation starts by removing all of the edges that connect agents belonging to the targeted group. The above edges are also called *internal edges*, while the edges that connect agents of the targeted group with agents not belonging to that group are called *external edges.* All of the external edges of the targeted group become edges of the hyper-agent to be formed. We must note that GRAL exchanges two control messages per hyper-agent to prevent a swapping of communicating agents. In terms of the space and time complexity of the algorithm, the reader is referred to [39].

4 Algorithmic Extensions

This section discusses how AMA and GRAL can be extended to handle: **(a)** node capacity constraints and **(b)** increased network knowledge.

4.1 Handling Capacity Constraints

Here we discuss two methodologies to tackle node capacity constraints. In a nutshell, three main elements must be added: **(a)** infeasible migrations must be dropped, **(b)** the available free capacity of nodes must be "discovered" dynamically, and **(c)** the capacity reservations must be made before initiating a migration.

Inquire-Lock Before (ILB). Before running the algorithm, a request is sent to *all potential destination* nodes, inquiring about their free capacity and requesting them to reserve up to the amount needed to host the agent(s) being considered for migration. The nodes reply with their available capacity, if any, which they reserve until further notice. Then, the selection of the destination(s) is done as described in the previous subsections, having a *consistent and guaranteed view* of node capacities. Nodes that

are not chosen as destinations are informed to release their reserved capacity. Nodes that are chosen as destinations are certain to be able to host the agent(s) assigned to them and release any "leftover" capacity when migration terminates.

Inquire-Lock After (ILA). The algorithm runs based on a previous, *possibly outdated*, view of free node capacities. Destinations are then contacted to reserve the capacity needed for hosting the agents assigned to them. Initially, all of the nodes are assumed to have an infinite free capacity. This view, along with the nominal capacity of each node, is updated based on the replies received for each request. To avoid excluding destinations due to outdated information, with a certain probability, nodes are assumed to have their full nominal capacity free, independently of the local view. Of course, this means that a migration might be decided based on invalid information, in which case the destination will send a negative reply when contacted to actually reserve capacity (and perform the migration).

Algorithmic Adaptations: When AMA picks a destination for a locally hosted agent, the migration is performed only if that node indeed has sufficient free capacity. GRAL checks the capacity constraint during the grouping phase of a sub-tree. If a leaf contains agents that exceed the capacity of the destination, it is pruned.

4.2 Handling Increased Network Knowledge

This section introduces extensions of AMA-1 and GRAL-1 for the case where a node is assumed to know the routing topology within a M-hop radius. We refer to this as *M-hop network awareness*. We must note that this information may be collected in a lazy fashion, incurring a minimal communication overhead, by piggybacking the M most recent node identifiers when a (small) message travels through the network. In fact, this information comes for free by employing a naming scheme that encodes path information into node identifiers (e.g., as in ZigBee networks with hierarchical routing [27]).

4.2.1 Extending AMA to support multi-hop migrations

The *M-hop agent migration algorithm* (AMA-*M*) is a straightforward extension of AMA-1 that exploits M-hop awareness. The difference is that for each agent a_k hosted at node n_s, AMA-*M* considers as possible candidates all nodes up to *M-hops* away from n_s that are involved in the message traffic of a_k. To identify whether a single agent multi-hop migration is gainful or otherwise we need to redefine Eq. 7. Therefore, we first need to introduce the variable $f_{sd}(z)$ that represents the z^{th} node along the path starting from node n_s and ending on node n_d. We should note that when $z = 0$, then $f_{sd}(z)$ represents n_s, while in case $z = h_{sd}$, then $f_{sd}(z)$ represents the node n_d. Therefore, Eq. 7' specifies the total gain of migrating an agent a_k along a path starting from some node n_s and ending on some other node n_d. In particular, Eq. 7' sums the individual gains of migrating an agent a_k hop-by-hop from a node n_s to a node n_d.

$$mg_{sd}(k) = \sum_{z=1}^{h_{ij}} g_{f_{sd}(z-1), f_{sd}(z)}(k) \qquad \text{Eq. 7'}$$

The algorithm chooses the destination for a_k by iteratively evaluating Eq. 7' for neighbor nodes, starting from 1-hop neighbors and working its way to more distant neighbors, following the most beneficial outbound direction. Each iteration determines

whether it is beneficial to move a_k to a node that is 1 hop further away from n_s, assuming a_k were hosted on the node picked in the previous iteration. The algorithm stops after M iterations or earlier when it is no longer beneficial to migrate a_k. AMA-M is expected to lead to fewer migrations than AMA-1 because an agent can (directly) move on a distant node in a single migration; as opposed to performing several 1-hop migrations to reach the same destination.

4.2.2 Extending GRAL to support multi-hop migrations

By following the same rationale as stated previously, we extend GRAL to exploit increased network knowledge. Specifically, GRAL-M extends GRAL to: **(a)** take advantage of M-hop awareness, and **(b)** potentially assign different parts of a group to different destinations (i.e., suggesting that some agents of the group migrate to different nodes).

For each sub-tree G, all neighbors within M hops of the local host and which are involved in the load associated with G are considered as potential destinations. The respective partial gain values for each destination node n_s are calculated in the spirit of multi-hop migrations. To do the above we need first to redefine Eq. 5 and Eq. 6 in the spirit of Eq. 7'. Therefore, Eq. 5 and Eq. 6 transform into Eq. 5' and Eq. 6', respectively. With the help of the above equations, we are able to transform Eq. 9 and Eq. 10 into Eq. 9' and Eq. 10', respectively. As we can see, the partial gain formulas (Eq. 9' and Eq. 10') are straightforward extensions of Eq. 9 and Eq. 10, respectively, taking into account the distance between n_s and n_d.

$$mdl_{sd}(k) = \sum_{z=1}^{h_{ij}} dl_{f_{sd}(z-1),f_{sd}(z)}(k) \quad \text{Eq. 5'} \qquad mal_{sd}(k) = \sum_{z=1}^{h_{ij}} al_{f_{sd}(z-1),f_{sd}(z)}(k) \quad \text{Eq. 6'}$$

$$mpg_{sd}(r) = mal_{sd}(r) - mdl_{sd}(r) \mid root = a_r \quad \text{Eq. 9'}$$

$$mpg_{sd}(k) = mal_{sd}(k) - mdl_{sd}(k) + 2\times(C_{km} + C_{mk})\times h_{sd} \mid p(a_k) = a_m \quad \text{Eq. 10'}$$

The partial gain values for each agent and destination node can be stored using a single tree structure, where the partial gain of an agent is a vector, where each element indicating the partial gain for a different destination node. The grouping process follows the same principle as in GRAL, but when merging two nodes the best destination for the leaf is selected for each destination option of the next-level node, producing an equal number of combined placements and partial gain values for the resulting group node. Again, the best combination is updated after each merge, and is returned when grouping finishes.

During the grouping phase, the gain values are calculated based on the fact that each merge "links" the parent agent a_k in the leaf node with the parent agent a_m in the next-level node (a_k is the child of a_m, both hosted at n_s). Let t_k and t_m denote the group nodes that contain these agents, and $t_{m,k}$ denote the group node that results after merging t_k with t_m. The combined partial gain after merging t_k and t_m is given by Eq. 11, under the assumption that a_k migrates from n_s at n_u, while a_m from n_s to n_v. In case t_m (and/or t_k) represents more than one agent, then m (and/or k) in Eq. 11 represents the most ancient agent in regards to the hierarchy of the agents participating in t_m (and/or t_k).

$$cpg_{su}^{sv}(m,k) = mpg_{sv}(m) + mpg_{su}(k) - (C_{mk} + C_{km})(-h_{sv} + h_{su} + h_{uv}) \quad \text{Eq. 11}$$

To explain the third term, recall that $mpg_{sv}(m)$ and $mpg_{su}(k)$ are calculated as per Eq. 10', unless a_m plays the role of the root ode where in that case Eq. 9' (instead of Eq. 10') is used to calculate $mpg_{sv}(m)$. Specifically, in case that $v \neq u$, the gain must be

adjusted by: **(a)** crediting the cost $(C_{mk}+C_{km})\times h_{sv}$ assumed in mpg_{sv}^{m} , **(b)** subtracting the benefit $(C_{mk}+C_{km})\times h_{iu}$ assumed in mpg_{su}^{k} , and **(c)** subtracting the load $(C_{mk}+C_{km})\times h_{uv}$ that will actually be incurred between a_m and a_k from their new hosts. Note that the third term disappears when $v=u$ in which case the partial gain of $t_{k,m}$ equals the sum of the individual partial gains, as usual.

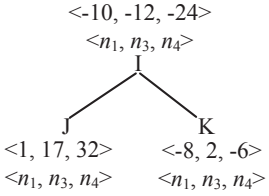

Figure 8. Grouping (I, J, K): initial state

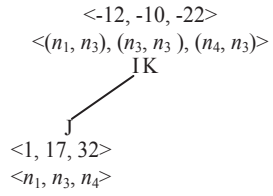

Figure 9. Grouping (I, J, K): after merging leaf node K with I

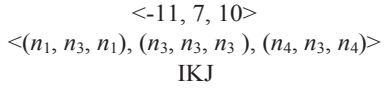

Figure 10. Grouping (I, J, K): after merging leaf node J with IK

We illustrate how the algorithm works by revisiting the previous example (of Fig. 4 and Fig. 6) for network awareness that equals two (M = 2). Assume that n_2 invokes the algorithm for the sub-tree *(I, J, K)*. The candidate destination nodes, involved in the message traffic associated with one or more agents of this sub-tree, are n_1, n_3 and n_4. Fig. 8 depicts the initial state of the sub-tree where each node is associated with three different partial gain values, one for each of the candidate destinations nodes (listed below the respective values). Each merge produces three combinations whereby each agent is separately assigned to a destination. Fig. 9 shows the result of merging group node *K* with *I* into a group node *IK* by considering three cases: **(a)** If *I* migrates on n_1 the best destination for *K* is n_3 yielding a combined partial gain of –12. **(b)** If agent *I* moves to n_3 the best destination for *K* is n_3 with a combined partial gain of –10. **(c)** if *I* migrates on n_4 the best destination for *K* is n_3 with a combined partial gain of –22. Fig. 10 illustrates the result of merging group node *J* with *IK* into a group node *IJK*. As we can see, the most beneficial group migration is that of migrating *I* and *J* on n_4, and *K* on n_3. The above gives a total gain of 10.

5 Optimality Issues and Bounds

We extend our system model to include notations that will be useful for our proofs. S and N denote the number of node-specific and node-neutral agents, respectively. The variable q_{km} captures the data exchanged between a_k and a_m as $q_{km} = C_{km}+C_{km} = q_{km}$. Let Eq. 12 represent the remote load exchanged between a_k and the agents located on a node different than $h(a_k)$. The local load exchanged between a_k and the agents that are co-located with a_k is captured by Eq. 13. Eq. 14 and Eq. 15 capture the remote and local load in terms of the node-neutral agents belonging into hyper-agent A_G, respectively.

$$rl(a_k) = \sum_{\forall m: h(a_k)\neq h(a_m)} q_{km} \quad \text{Eq. 12} \qquad ll(a_k) = \sum_{\forall m: h(a_k)= h(a_m)} q_{km} \quad \text{Eq. 13} \qquad rl(A_G) = \sum_{\forall a_k \in A_G: h(a_k)\neq h(a_m)} q_{km} \quad \text{Eq. 14}$$

$$ll(A_G) = \sum_{\forall (a_k \vee a_m)\in A_G: h(a_k)= h(a_m)} q_{km} \quad \text{Eq. 15} \qquad q_{km} \leq B \quad \text{Eq. 16}$$

Let D be the diameter of the network, while $B/2$ be the maximum data an agent can send towards another one, each time it calls the respective network routine. Therefore, the maximum volume of data can pass through an edge (of the application tree) at any instance of time, must equal B (see Eq. 16). This happens when both of the involved agents simultaneously send data to-

wards one another $B/2$ data units.

We refer to a node as the *center of (communication) gravity* for an agent, if placing the agent on that node minimizes the amount of its communication with other agents over the wireless network, based on the current placement of those agents. We say that an agent or hyper-agent is *individually balanced* if it is placed at its center of gravity; else, the agent is *unbalanced*. Note that in case of GRAL an agent is individually balanced if there is no sub-tree that contains that agent after the contraction phase completes. A placement where all agents or hyper-agents are individually balanced is called a *totally balanced placement* (TBP). For simplicity, and without the loss of generality, henceforth we refer to both simple agents or hyper-agents as "agents".

Lemma 1. *Given a TBP, then any attempt to migrate an agent to any node within the system always leads to a new network cost that is greater than or equal to the old network cost.*

Proof. As per the TBP definition, there is no 1-hop migration of any agent within the system that may reduce the total network cost. Therefore, it only remains to be shown that the above applies for any M-hop migration. Consider the example as depicted in Fig. 11. As we can see, there is a 3-hop migration $m_{14}(k)$ that can be decomposed into three 1-hop migrations ($m_{12}(k)$, $m_{23}(k)$, $m_{34}(k)$). The gain of $m_{14}(k)$ is equal to the sum of the gains of the above three 1-hop migrations: $g_{14}(k) = g_{12}(k) + g_{23}(k) + g_{34}(k)$ (Eq. 17). Recall that the gain of an agent migration can be split into the advantageous ($al_{sd}(k)$) and the disadvantageous ($dl_{sd}(k)$) load. Looking at the advantageous loads of these migrations, we claim that the following holds $al_{12}(k) \geq al_{23}(k) \geq al_{34}(k)$. Recall that $al_{sd}(k)$ represents the communication load between a_k (located on n_s) and the agents using n_d as: **(a)** hosting node to reach a_k or **(b)** a routing. In our example, set A is called the set of agents using n_2 to communicate with a_k (when a_k is located on n_1). Set B is named the set of processes using n_3 to communicate with a_k (when a_k is located on n_2). Set C is named the set of processes using n_4 to communicate with a_k (when a_k is located on n_3). We observe that, set A is a superset of set B, which in turn is a superset of set C. Therefore, we have the following $A \supseteq B \supseteq C \Rightarrow al_{12}(k) \geq al_{23}(k) \geq al_{34}(k)$ (Eq. 18). Following the same rationale as that of advantageous loads, we can say that $dl_{12}(k) \leq dl_{23}(k) \leq dl_{34}(k)$ (Eq. 19). Combining Eq. 7, Eq. 18, and Eq. 19, we obtain $g_{12}(k) \geq g_{23}(k) \geq g_{34}(k)$ (Eq. 20). The combination of Eq. 17 and Eq. 20 gives $g_{14}(k) \leq 3 \times g_{12}(k)$. Note that n_2 is the 1-hop neighbor of n_1, which is used by n_4 to reach n_1. Therefore, the above can be generalized into the following statement: *If we assume a M-hop migration in a tree-structured network from a node n_s to a node n_d, and an 1-hop neighbor (called n_z) used by n_s to reach n_d, then the following holds* $g_{sd}(k) \leq M \times g_{sz}(k)$ (Eq. 21).

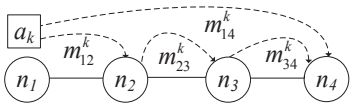

Figure 11. Decomposing a 3-hop migration into three 1-hop migrations

Now getting back to our case, we must recall that there is no 1-hop agent migration that can gain benefit greater than zero. (This is a result of the TBP assumption undertaken previously.) We also must note that $g_{sz}(k)$ in Eq. 21 represents the benefit when migrating a_k by 1-hop. Therefore, according to Eq. 21, in the best case scenario any M-hop migration will have benefit M

times zero. From the above we can conclude that there is no M-hop migration with benefit greater than zero. Consequently, there is no M-hop migration that can further reduce the total network cost. □

Lemma 2. *Given a TBP, then any migration of an agent leaving the total network cost intact or increasing it cannot cause a series of other migrations that may eventually further reduce the total network cost*.

Proof. To prove the lemma by contradiction, we assume that **(i)** a_k is an agent migrating from n_s to a 1-hop neighbor n_d, **(ii)** this migration leaves the total network cost intact (or increases it), but causes other subsequent migrations that eventually reduce the total network cost. The above implies that an adjacent agent of a_k (named a_i) is affected by $m_{sd}(k)$ and becomes unbalanced. Therefore, the following three scenarios must be examined: **(a)** a_i is located on a node that uses n_d to reach a_k. However, the above entails that a_i was not in its center of gravity before $m_{sd}(k)$ takes place, which contradicts TBP. **(b)** a_i is located on a node n_f (other than n_s) that does not use n_d to reach a_k. However, the above implies that a_i was again not in its center of gravity before $m_{sd}(k)$ takes place. The above is justified by the fact that the load associated with a_i and n_f's 1-hop neighbor used to reach n_s remains the same irrespective of $m_{sd}(k)$ (this contradicts TBP). **(c)** a_i is co-located with a_k, which entails that both a_k and a_i form an unbalanced hyper-agent (this contradicts TBP). Therefore, we conclude that there cannot exist a $m_{sd}(i)$. To complete our proof, we also need to prove that a contradiction holds for the case where n_d is a M-hop neighbor of n_s. Because $m_{sd}(k)$ is a M-hop migration leaving the total network cost intact (or increasing it), then it is equivalent of saying that there are M 1-hop migrations from n_s to n_d, where each of them leaves the total network cost intact (or increases it). Applying the above three scenarios for M migrations we end the proof. □

Theorem 1. *A totally balanced placement (TBP) is always an optimal placement.*

Proof. According to Lemma 1 and Lemma 2, if a totally balanced placement has been achieved, then there cannot be any migration that can further reduce the total communication cost. □

5.1 Identifying the Worst-case Bound of AMA

In this section, we identify the worst-case bound between AMA and the optimal algorithm (OPT). First, we will set forth the sketch of our proof and then we will move on towards a more detailed discussion of the proof. We assume that after performing AMA, in our system, there are no individually unbalanced agents. The above is a valid assumption because AMA always results in a placement in which all of the agents within the system are individually balanced. We also assume that there exists at least one unbalanced group of agents (called hyper-agent A_G). We already know that AMA cannot identify unbalanced group of agents to migrate it towards the center of gravity. However, the optimal algorithm will always achieve the aforementioned. Therefore, for the bound to be as less as possible, we demand that A_G be as far away from its center of gravity as possible. As a result, we can achieve the above by placing: **(a)** A_G on a node (hereafter denoted as n_i) that is located on the one end of the network and **(b)** A_G's center of gravity (or adjacent agents) on a node (hereafter denoted as n_j) that is located on the opposite end of the network. In that way, the distance between A_G and its center of gravity equals the diameter of the network D (measured in hops).

Lemma 3. *Assuming that all of the node-neutral agents of a given application are hosted on a single node n_i and participate on an unbalanced hyper-agent A_G, then the worst-case bound between AMA and the optimal algorithm is equal to (N-1)×2B×D.*

Proof. Without the loss of generality, we assume that there are no individually unbalanced agents within the system as AMA always result in a placement where all of the agents are individually balanced. According to the preceding text, for the bound to be as less as possible, we demand that the source of the remote load of A_G must stem from n_j, which is D hops away from n_i (**Condition 1**). In Fig. 12, we set forth an example of m node-neutral agents that are individually balanced but not totally balanced. Each node-neutral agent depicted in Fig. 12 has: **(a)** one or two *internal* edges belonging to A_G, with their load representing a portion of its local load (e.g., a_1 has only one edge q_{12} belonging to A_G), **(b)** one *local abstract* edge representing the remaining local load, which is the local load ($ll(a_k)$) minus the load of the internal edges, and **(c)** one *remote abstract* edge representing the remote load. (In Fig. 12, $rl(a_1)$ is one such edge.)

Because of Condition 1 and of the fact that each node-neutral agent is individually balanced, we conclude that the following must hold $ll(a_k) \geq rl(a_k)$, which we term as **Condition 2**. The best-case scenario for the optimal algorithm is to force each agent to have as less of a local load as possible. Otherwise, the aforementioned load will become remote when the optimal algorithm takes the decision to migrate the unbalanced group. Therefore, the best-case scenario is that the local load of an agent equals the accumulation of the loads between the agent in question and its adjacent agents belonging to the unbalanced group (**Condition 3**). In case of two node-neutral agents (m = 2 in Fig. 12), then according to Condition 3, both $ll(a_1)$ and $ll(a_2)$ must be equal to q_{12}. Consequently, the total remote load cannot be greater than $2q_{12}$. While the total communication cost cannot be greater than $2q_{12}\times D$ (that is the remote load multiplied by the traveling distance in hops). Now due to Eq. 16, we can deduce that the total communication cost cannot be greater than $2B\times D$. If three node-neutral agents exist, then we have $ll(a_1) = q_{12}$, $ll(a_2) = q_{12} + q_{23}$, and $ll(a_3) = q_{23}$. Consequently, the total remote load and the total communication cost cannot be greater than $4B$ and $4B\times D$, respectively. When four node-neutral agents are present, then the total communication cost cannot be greater than $6B\times D$. (We must stress on the fact that the above observations are completely independent of the underlying application tree connectivity.) Therefore, we can observe that for each internal edge of A_G, AMA incurs $2B\times D$ additional network overhead. Because the optimal algorithm is able to migrate A_G towards n_j, the total network overhead will be zero. In principle, our application is structured as a tree, which means that the largest difference (in terms of network overhead) between AMA and OPT will be equal to $(N-1)\times 2B\times D$. The above is because in a tree of N agents, N-1 edges must exist. □

Lemma 4. *Given that: **(a)** there exist N node-neutral agents within our WSN, **(b)** N-1 of them ($a_1 \ldots a_{N-1}$) are on an unbalanced hyper-agent A_G that is hosted on a node n_i, and **(c)** all of the N agents are hosted on n_i, we can conclude that the bound between AMA and the optimal algorithm is less than (N-1)×2B×D.*

Proof. We have already understood that the optimal algorithm will migrate A_G from n_i to n_j. Note that the optimal algorithm cannot migrate a_N to n_j. This is because a_N does not belong to A_G. Without the loss of generality, we can assume that all of the agents are individually balanced. As previously established, the worst-case scenario for AMA (conversely the best-case for the

optimal algorithm) is to have the source of $rl(A_G)$ to be as far away as possible from the hosting node of A_G. Therefore, we have the following two cases: **(a)** it is located on n_i and **(b)** it is located on a different node other than n_i.

Case (a): Note that $rl(a_N)$ cannot stem from n_j, otherwise the OPT would be able to migrate a_N towards n_j. Because a_N does not participate in A_G and due to the fact that A_G must be a collection of agents that cannot be partitioned, we deduce that a_N is connected to only one agent, namely a_k of A_G. For OPT to not migrate a_N towards n_j, we should have q_{Nk} to be less than the local load between a_N and the node-specific agents that it communicates with. Therefore, the worst-case for AMA (conversely, the best-case for OPT) is that the above loads should be as large as possible (with the former load being 0 and the latter one being 1). Consequently, when applying the optimal algorithm, the total network overhead becomes $1\times D$, while in case of AMA, the total network overhead becomes $(N\text{-}2)\times 2B\times D$. Thus, the largest difference between AMA and OPT is $(N\text{-}2)\times 2B\times D$–D. However, we can see that the difference is less than that indicated in Lemma 3.

Case (b): The worst-case scenario for AMA is that a_N is located as far away as possible from n_i. The best-case scenario for OPT is that a_N be located on n_j. Therefore, we assume that a_N is located on n_j, and consequently q_{Nk} should be less than the local load between a_N and the node-specific agents that it communicates with. The worst-case scenario for AMA (conversely, the best-case scenario for OPT) is to have q_{Nk} be as large as possible. The above is because AMA cannot migrate A_G to n_j; however, OPT is able to do so. Consequently, we assume that q_{Nk} is equal to B, with the network overhead in terms of that load being $B\times D$. In terms of the network overhead associated with A_G, it is equal to $(N\text{-}2)\times 2B\times D$. The total network overhead incurred by AMA is equal to $(N\text{-}2)\times 2B\times D+B\times D$, while the total network overhead incurred by OPT is equal to zero as A_G and a_N are co-located. Therefore, the largest difference between AMA and OPT is equal to $(N\text{-}2)\times 2B\times D+B\times D$ (less than that reported in Lemma 3). □

Theorem 2. *The worst-case bound between AMA and the optimal algorithm is $(N\text{-}1)\times 2B\times D\times T$, with T denoting the maximum number of times the agents can send $B/2$ data units over the WSN network.*

Proof. Lemma 3 and Lemma 4 prove that at any time instance, the largest difference (in terms of the network overhead) between AMA and OPT is equal to $(N\text{-}1)\times 2B\times D$. We assume that T denotes the maximum number of times the agents can send $B/2$ data units over the WSN network. Therefore, there can be at most T time instances that the total network overhead equals to $(N\text{-}1)\times 2B\times D$. Summing up all of them, we conclude that the total network overhead equals to $(N\text{-}1)\times 2B\times D\times T$. We also conclude that as T grows to infinity, the total network overhead also grows to infinity, which means that there is no fixed bound between AMA and OPT. □

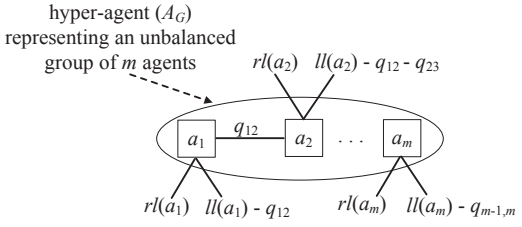

Figure 12. Unbalanced group of m agents

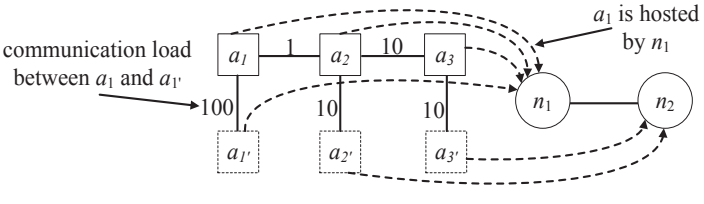

Figure 13. Unbalanced group of 2 agents (a_2, a_3)

5.2 GRAL* an Optimal Algorithm

In this section, we first discuss two important issues that make GRAL identifies sub-optimal placements. Specifically, the first case is that we prove that GRAL chooses the destination node for a potential migrating group in such a way that may lead to a sub-optimal assignment scheme. The other case is attributed to the fact that the choice of the root agent of a sub-tree may lead to a sub-optimal placement. In the sequel, we introduce a modification of GRAL that we name GRAL* along with a mathematical proof that GRAL* always results in an optimal placement. GRAL's sub-optimality and GRAL*'s optimality proofs are discussed in a detailed fashion in the subsequent text.

Lemma 5. *GRAL may result in a sub-optimal placement because of the fact that it chooses only the most promising nodes as the destination for a potential migrating group.*

Proof. Consider the example depicted in Fig. 13, where three node-neutral agents (a_1, a_2, a_3) and three node-specific agents ($a_{1'}$, $a_{2'}$, $a_{3'}$) compose an application. Recall that when GRAL runs on a node, it chooses for each disjoint sub-tree the most promising destination node. In our example, GRAL will identify the most promising node for the sub-tree composed of a_1, a_2, and a_3. The only possible destination for the aforementioned group of agents is n_2. Therefore, we assume that the destination of a_1, a_2, and a_3 is n_2 and calculate the disadvantageous (a_1:101, a_2:11, a_3:10) and the advantageous (a_1:0, a_2:10, a_3:10) loads. By subtracting the disadvantageous loads from the advantageous ones (0-101+10-11+10-10), we find that the resultant equals to -102. Therefore, the most promising destination node is the current hosting node. Because there is no other disjoint sub-tree that GRAL can identify, the current placement is considered GRAL's final placement. As can be seen, the total network overhead of the current placement scheme is equal to 20, which can be deduced from the communication between $\{a_2, a_{2'}\}$ and $\{a_3, a_{3'}\}$. However, we also can observe that if we move both a_2 and a_3 to n_2, then the total network overhead becomes 1 (communication between a_1 and a_2). Consequently, we conclude that there is an unbalanced group of agents (a_2 and a_3) that cannot be identified by GRAL, resulting in a sub-optimal placement. □

Lemma 6. *GRAL may result in a sub-optimal placement because it chooses the root agent of a sub-tree randomly.*

Proof. Consider again the example reported in Fig. 13 and assume that GRAL now possesses the ability to identify that if n_2 hosts some of the agents of n_1, then the placement will be improved. Therefore, GRAL will proceed with the calculation of partial migration gains and the tree contraction phases as shown in Fig. 14 and Fig. 15, respectively. Specifically, we can see in Fig. 14 that if GRAL chooses a_1 as the root agent of the sub-tree $\{a_1, a_2, a_3\}$, then the contraction phase will result in a final contracted node $\{a_1, a_2, a_3\}$ with negative partial migration gain. In principle, a final contracted node with negative migration gain contributes negatively to the total network overhead, which ends up pruning the node in question. However, we can observe in Fig. 15 that if GRAL chooses a_2 as the root agent of the aforementioned sub-tree, then the contraction phase will result in a final contracted node $\{a_2, a_3\}$ with positive partial migration gain. The above means that if we move both a_2 and a_3 to n_2, then the total network overhead will be improved. Consequently, by appropriately choosing the root agent of a sub-tree, we can identify an unbalanced group of agents that GRAL is not able to identify. □

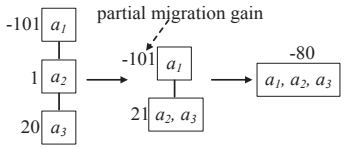

Figure 14. Sub-tree rooted on a_1.

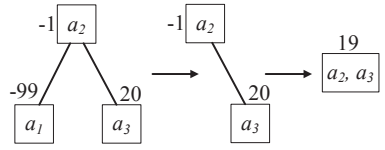

Figure 15. Sub-tree rooted on a_2.

Lemma 7. *If the root agent of an identified sub-tree belongs to an unbalanced group of agents, then after the contraction phase, GRAL will identify the above unbalanced group and identify an optimal migration.*

Proof. Recall that when GRAL calculates the partial migration gains of a sub-tree, each agent belonging to that sub-tree is assigned a partial migration gain for its migration to a chosen destination. If the aforementioned migration concerns a bottom-level agent, then its partial migration gain corresponds to an upper bound gain. The above is justified by the fact that when calculating a partial migration gain of an agent, we consider that the respective agent will migrate along with its parent, resulting in the best-case scenario. To prove our result, we proceed with the following three assumptions.

Assumption 1: Consider that **(i)** GRAL identifies a promising destination node n_d for a sub-tree of only two levels rooted at an agent a_r, and **(ii)** the respective sub-tree represents an unbalanced group of agents (with a_r belonging to that group), with an optimal solution being its migration to n_d. We refer to the group of agents that the optimal algorithm decides to migrate to n_d as the *optimal migrating group*. If there are agents at the bottom level of the sub-tree under consideration that have partial migration gain less than or equal to zero, then GRAL will decide to prune them. It is noteworthy to mention that the optimal algorithm will also decide to remove the respective agents from the optimal migrating group. The above is because the upper bound migration gains of the corresponding agents are not positive. Therefore, GRAL's decision to prune the aforementioned agents is correct. The rest of the bottom-level agents (if any) must have positive partial migration gains. The above means that: **(a)** GRAL will decide to merge the aforementioned agents with their parents and **(b)** if parent a_r of the aforementioned agents migrates along with them to n_d, then the actual migration gain of that group of agents will be positive. Because of the assumption that a_r belongs to an unbalanced group, the hypothesis stated in (b) will be true and OPT will decide to move the corresponding group of agents to n_d. Now it only remains to be shown that GRAL also will decide to move the same group of agents as OPT to n_d. The aforementioned can be justified from the fact that GRAL will perform the contraction phase, which will result in a positive partial migration gain of the respective group of agents. We can conclude that the above is true because: **(a)** by construction, the partial migration gain of the aforementioned group of agents (final contracted node) is equal to the actual migration gain and **(b)** the actual migration gain of the above group is positive because of OPT's decision to migrate the group to n_d.

Assumption 2: The hypothesis is the same as that in Assumption 1 with the difference being only in the levels of the sub-tree. Here we assume a three level sub-tree. In the first step, GRAL will proceed with the merging/pruning of the bottom level of the tree. As stated earlier, each leaf contributing negatively will be pruned as OPT will take the same decision. Because the rest of the bottom-level agents must have positive partial migration gain, they will be merged with the next upper-level agents. Therefore, the process will result in a case that is identical to Assumption 1, with the difference being that some of the bottom-level

nodes may represent hyper-agents instead of individual agents. We must note that by construction, a hyper-agent shares the same properties as that of a simple agent. Combining the above with Assumption 1, we conclude that GRAL will move towards n_d the same agents as OPT.

Assumption 3: The hypothesis is the same as that state in Assumption 2 with the only difference being the depth of the sub-tree, which we consider to be four. Following the same rationale as previously detailed, we conclude that this case is reduced to the case of Assumption 2. Therefore, GRAL again converges to an optimal decision.

By continuing the above assumptions iteratively, we conclude that the general case where the sub-tree is consisted of n levels is always reduced to Assumption 1. As a result, we conclude that if the root agent of a sub-tree belongs to an unbalanced group, then GRAL will identify and move that group in an optimal way. □

Definition 1. GRAL* is a modification of GRAL that circumvents the drawbacks of GRAl as identified in Lemma 5 and Lemma 6. GRAL* becomes: For each possible pair (A_G, n_d) — where A_G is an identified sub-tree and n_d is the potential destination node of that sub-tree — GRAL* constructs as many sub-trees (containing the same agents with A_G) as the number of the agents belonging to A_G, with each such sub-tree being rooted at a different agent.

Theorem 3. *GRAL* always results in an optimal placement.*

Proof. By construction, GRAL* is able to: **(a)** identify each non-partitioned collection of co-located node-neutral agents (or sub-tree) and **(b)** examine all of the possible root agent combinations of a given sub-tree. Thus, for each unbalanced group of agents, GRAL* always identifies a sub-tree with its root being part of the corresponding unbalanced agent group. Combining the above with Lemma 7, we conclude that GRAL* is able to identify all of the unbalanced groups and optimally identify placements. □

The space complexity of GRAL* remains the same as that of GRAL, that is $O(A'N'^2)$. However, the time complexity is a little bit different compared to that of GRAL. The above is due to the fact of the extra computing cost incurred to examine all of the possible root agent combinations of a given sub-tree. Therefore the time complexity of GRAL* becomes $O(A'N'^2M')$, with M' representing the total number of agents participating on the corresponding sub-tree.

5.3 Identifying the Worst-case Bound of GRAL

In this section, we show that there is a fixed bound between GRAL and the optimal algorithm.

Theorem 4. *The worst-case bound between GRAL and the optimal algorithm is 1/*((*N*-2)×2*B*).

Proof. We split our proof into two parts. The first part addresses the case when the number of node-neutral agents is limited to two ($N \leq 2$), while the second part examines the case when $N > 2$.

Part A: In this part, we prove that GRAL is optimal when $N \leq 2$. Because AMA is optimal when $N = 1$ (stems directly from Theorem 1), we deduce that GRAL also is optimal for that particular case. Therefore, we only need to consider the case where $N = 2$. Because the sub-optimality of GRAL is attributed to Lemma 5 and Lemma 6 (GRAL* becomes optimal by overcoming the drawbacks brought out by these lemmas), we only need to show that the aforementioned lemmas do not hold true for the case of $N = 2$. We have to examine three cases to prove the above: **(a)** if both of the node-neutral agents form an unbalanced group, then

GRAL will identify the respective unbalanced group and migrate it optimally, **(b)** if only one of the two node-neutral agents is unbalanced then the above is reduced to the case of a single agent migration (optimal decision), and **(c)** if both of the node-neutral agents are totally balanced, then the optimal solution has already been identified by GRAL. Therefore, Lemma 5 and Lemma 6 do not hold true for the aforementioned cases.

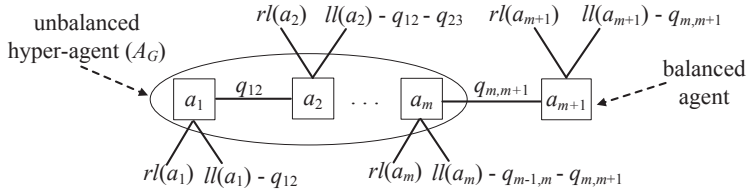

Figure 16. Unbalanced group of agents.

Part B: In this part, we consider the case when $N > 2$. Without the loss of generality, we assume that each of the agent is individually balanced. Recall from Lemma 5 and Lemma 6 that GRAL becomes suboptimal when there is at least one totally balanced agent. Therefore, we extend the case represented in Fig. 12 to a scenario reflecting **Part B** in Fig. 16. In that case, GRAL cannot identify the unbalanced hyper-agent A_G to migrate it towards its center of gravity, while the optimal algorithm is always able to do it. Therefore, for the bound to be as less as possible (for the sake of the proof) we must have A_G located to be as far away as possible from its center of gravity (**Condition 1'**). We can achieve the above by placing A_G and A_G's center of gravity (or adjacent agents) in such a way such that their distance equaling the diameter of the WSN D (measured in hops).

Because of Condition 1' and due to the fact that each node-neutral agent is individually balanced, we conclude that the following must hold $ll(a_k) \geq rl(a_k)$, which we denote as **Condition 2'**. The best-case scenario for the optimal algorithm is to force a_{m+1} to have no remote load, while each agent belonging to the unbalanced group to have as less of a local load as possible (**Condition 3'**). The intuition behind the latter is that the local load will become remote load when the optimal algorithm takes the decision to migrate the unbalanced group. Therefore, we can assume that the only local load of an agent belonging to the unbalanced group will be the load associated with the rest of the node-neutral agents belonging to the unbalanced group. The above means that $q_{m,m+1}$ must equate to zero, which cannot happen as a_{m+1} would not be adjacent to the unbalanced group. The aforementioned in turn means that GRAL would be able to identify the respective unbalanced group, which is deemed undesirable for the sake of the proof. Therefore, we state that $q_{m,m+1}$ must equate to 1 (**Condition 4'**).

Initially, we assume the case where $m = 2$, with a_1 and a_2 belonging into the unbalanced group, while a_3 being totally balanced. According to Condition 3', $ll(a_1)$ and $ll(a_2)$ must be equal to q_{12} and $q_{12}+q_{23}$, respectively. Due to Condition 4', we know that $q_{12}+q_{23}$ must equal $q_{12}+1$. If we are considering applying GRAL to the above case, then we must recall that according to Condition 1' and Condition 2', the cumulative remote and communication load cannot exceed $2q_{12}$ and $2q_{12}\times D$, respectively. Due to Eq. 16, we can deduce that the worst-case for GRAL is to have the total communication cost equal to $2B\times D$. If we apply the optimal algorithm to the above case, then according to the Condition 1' and Condition 4' the cumulative remote and communication load equals to 1 and D, respectively. Following the same rationale, when $m = 3$, we conclude that the total communication load in terms of GRAL and the optimal algorithm becomes $4B\times D$ and D, respectively. Generalizing the above for N

agents, we have $m = N - 1$. Consequently, the total communication cost of the optimal algorithm and GRAL are equal to D and $(N\text{-}2)\times 2B\times D$, respectively. The factor $(N\text{-}2)$ is justified by the fact that the unbalanced group is composed of a tree of N-1 agents (or N-2 edges). Therefore, the ratio becomes $1/(N\text{-}2)\times 2B$. We must note that if the number of the balanced agents is larger than one, then the ratio becomes bigger (undesirable for the sake of the proof). For example, if the number of balanced agents is two, then the unbalanced group is composed of a tree with N-2 agents (or N-3 edges). Therefore, the ratio becomes $1/(N\text{-}3)\times 2B$. □

6 Experimental Evaluation

We conducted the experimental evaluation using ns2 [26]. Each experiment is repeated 25 times, with each plot representing the average of them. The number 25 represents the combination of five different network topologies and five different application graphs. The details of the simulation setup are briefly given below.

Network generation. Two types of networks are considered, with 50 and 20 nodes placed randomly in a plane of 120×120 and 80×80 distance units, respectively. In both cases, nodes are assumed to be in range of each other if their Euclidean distance was less than 30 distance units. The corresponding tree-based routing topology is obtained by constructing a spanning tree, whereby each pair of nodes is connected via a single path.

Application generation. The application tree structure is generated randomly, based on the (given) number of node-specific agents. Three different application structures are generated with (50, 22), (25, 12) and (10, 5) (node-specific, node-neutral) agents, referred to as app-50, app-25 and app-10, respectively. The initial placement of agents on the network is random, unless stated otherwise.

Application traffic. We let each node-specific agent send between one to five messages per simulation time unit to its parent. For the load from a node-neutral agent towards its parent we consider three cases: **(a)** *lavg*: the agent sends the average of the load received from its children, corresponding to a data aggregation scenario, **(b)** *lsum*: the agent sends to its parent the sum of the loads received from its children, corresponding to a forwarding scenario, and **(c)** *lmix*: half node-neutral agents (randomly chosen) generate load according to *lavg* and the other half according to *lsum*. All messages are of equal size. The application-level message traffic pattern remains static to allow the algorithms to converge.

Invocation period. Nodes invoke the algorithm every T time units, each node starting this periodic invocation at a different point in time, randomly set between *0* and T. When a migration attempt fails (because it was impossible to reserve enough capacity), a node backs-off for a randomly chosen time interval, between T and $5T$.

Metrics. In each experiment, the quality of the solution is measured in terms of total network load for the placement produced by GRAL, GRAL* and AMA versus the initial placement. We must note that in capacitated cases the problem becomes NP-complete [37]. Therefore, the optimal solution is not obtained by GRAL* but via an exhaustive search (the latter can be computed only for small-scale experiments). We also record the total number of control messages generated (to avoid concurrent con

Table II. Application traffic *lavg*

Algorithm	Total Load	Agent Migrations	Control Messages	Time to Converge
Initial	11004.2	N/A	N/A	N/A
AMA-1	4711.8	825.4	1650.8	10.6
GRAL-1	4601	957.6	1730.4	9.2
GRAL*-1	4578	961.4	1736.8	9.2

Table III. Application traffic *lsum*

Algorithm	Total Load	Agent Migrations	Control Messages	Time to Converge
Initial	29628.2	N/A	N/A	N/A
AMA-1	6742.4	842.4	1684.8	11.2
GRAL-1	4895.6	1247	1764.4	8.2
GRAL*-1	4432	1281.2	1836.8	8.4

Table IV. Application traffic *lmix*

Algorithm	Total Load	Agent Migrations	Control Messages	Time to Converge
Initial	17964.4	N/A	N/A	N/A
AMA-1	5921.8	738.8	1477.6	10.8
GRAL-1	4843.6	1070	1622.8	8.6
GRAL*-1	4654	1096.2	1686.8	8.6

flicting migrations as well as inquire about, lock and unlock node capacity),the number of migrations performed, and the time (the maximum number of algorithm invocations performed by any node) needed to reach the final placement.

For experiments without capacity constraints, convergence is inferred when all nodes invoke the algorithm without deciding for any migration while, in capacitated cases, the simulation is stopped when each node consecutively invokes the algorithm 4 times without successfully performing any migration.

6.1 Results without Capacity Constraints

In the first experiment, we compare the placements obtained by the GRAL and AMA variants, and the optimal algorithm GRAL* without taking into account capacity constraints. We choose network topologies of 50 nodes, and a mixed application of 5 applications of each application type (app-10, app-25, app-50). Algorithms' network awareness measured in hops ranges between 1 and 4. Table II summarizes the results of the aforementioned algorithms for an initial (random) placement and the *lavg* model. The first observation is that the initial placement is quite bad, incurring more than twice the total load of the optimal. Specifically, GRAL-1 (GRAL with network awareness of 1 hop) achieves almost the same performance with optimal (5‰ difference), AMA-1 following close to optimal (2.9% difference). As expected, both GRAL and GRAL* perform more migrations compared to AMA. As far as the control messages are concerned, we can see that they are twice the number of migrations in case of AMA, while in GRAL and GRAL* they are less than twice because a hosting request/reply may refer to more than one agents. A final observation is that GRAL and GRAL* need less time to converge against AMA. The above is attributed to the fact that GRAL and GRAL* perform agent migrations in groups compared to AMA that performs only single agent migrations.

The same experiment was conducted for the other two load models (*lsum* and *lmix*) and the results show that the performance difference between GRAL and GRAL* increases compared to the previous experiment (see Table III and Table IV). Specifically, GRAL* achieved 9.4% and 3.9% better performance against GRAL over *lsum* and *lmix*, respectively. However, it is interesting to notice that compared to GRAL, AMA is even worse in *lsum* and *lmix* load models, yielding a noticeably inferior performance of 34.2% and 21.4% against GRAL*. The aforementioned difference is attributed to the fact that in these models the dependency among adjacent agents becomes stronger, especially when the load between node-neutral agents and their parents is

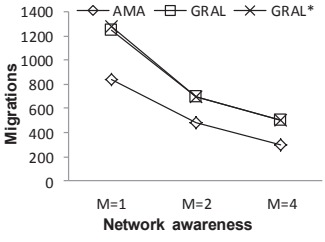

Figure 17. Number of migrations when increasing network awareness

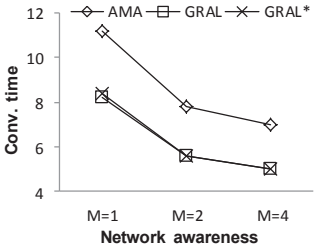

Figure 18. Convergence time when increasing network awareness

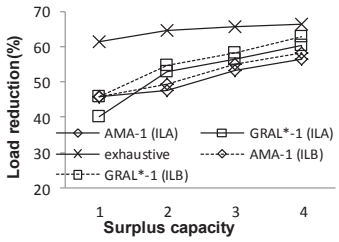

Figure 19. Load reduction when increasing surplus capacity (1-hop variants, small scale)

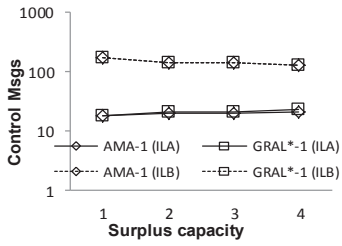

Figure 20. Num. of control msgs when increasing surplus capacity (1-hop variants, small scale)

relatively heavy. The above stresses the importance of considering group migrations instead of single ones. Hereafter, in the experiments we will always be using the *lavg* model.

We repeated the above mentioned experiments with increased network awareness. We omit to show the results in regards to the total load because there were only rudimentary fluctuations when varying the network awareness.

In Fig. 17 and Fig. 18 we show the number of migrations performed and the time to converge when increasing network awareness. As observed, the number of migrations performed decreases as the network awareness increases, which is expected since the final agent destinations can be reached through multi-hop jumps. The above is also reflected in the convergence time of algorithms.

6.2 Results with Capacity Constraints

In this section, we have conducted two major set of experiments. Both of them consider nodes that are able to host only a limited number of agents due to capacity constraints. The first part concerns small-scale network topologies and applications, thus giving us the ability to compare our algorithms against the optimal. In the second part, involving larger simulation setups we exclude the exhaustive algorithm due to its prohibitive execution time.

6.2.1 Small-scale experiments

Here we compare the performance of the AMA, GRAL, and GRAL* algorithms for the ILA and ILB schemes versus the optimal solution obtained by exhaustive search. We must note that the reason we used exhaustive search to derive optimal placements, is that the problem becomes NP-complete [37] in capacitated cases. To reduce simulation time (for the exhaustive algorithm) the experimental setup consists of a 20 node network and app10. The evaluation is performed for varying levels of "tightness" of the capacity constraint. More specifically, we start with the nodes having just enough capacity to store the agents defined in the initial placement and add additional capacity to hold 1, 2, 3, or 4 extra agents at each node. We first evaluate the performance of algorithms for 1-hop awareness, while in the sequel we repeat the experiment to see the behavior of algorithms when increasing network awareness. Because we noticed that in small-scale experiments GRAL's performance is identical to that of GRAL*, for clarity reasons we don't show GRAL's performance.

Fig. 19 presents the percentage of load reduction achieved by the 1-hop variants. All algorithms reduce significantly the load

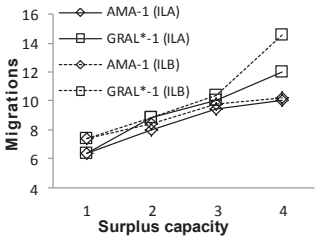

Figure 21. Number of migrations when increasing surplus capacity (1-hop variants, small scale)

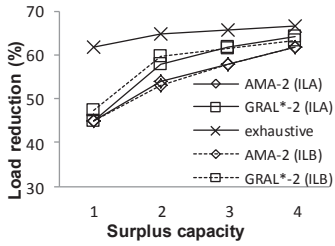

Figure 22. Load reduction when increasing surplus capacity (2-hop variants, small scale)

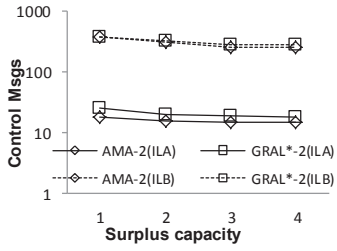

Figure 23. Num. of control msgs when increasing surplus capacity (2-hop variants, small scale)

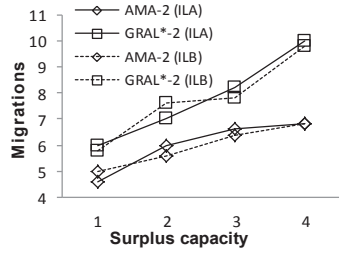

Figure 24. Number of migrations when increasing surplus capacity (2-hop variants, small scale)

by more than 40%, even in the case where the capacity constraint is tight. Compared to exhaustive search, we notice that the performance difference between the algorithms and the optimal solution rapidly decreases as more capacity becomes available. For instance, with a surplus capacity of one agent, the difference between "GRAL-1 (ILB)" and the optimal result is more than 15%, while with a surplus capacity of four agents it is less than 5%. This is because when capacity is scarce it is also more likely that nodes will be filled. A filled node essentially acts asbottleneck separating the tree network into two parts. For 1-hop algorithms, this means that these network parts cannot exchange agents; therefore, it is more likely to reach a suboptimal solution. Obviously, exhaustive search does not perform any real migrations to find the optimal solution, thus does not suffer from the effects of bottleneck nodes. The increase in load reduction for exhaustive search should be attributed to the "generally improved" optimization potential as the capacity of nodes increases and the setting gradually shift towards the unconstrained case.

Concerning the relative performance of the 1-hop algorithms in Fig. 19, we can observe the following: **(a)** ILB achieves better placements than ILA in both AMA-1 and GRAL-1, and **(b)** GRAL-1 consistently outperforms AMA-1 except in the case of ILA and surplus capacity of 1. Both observations are due to the fact that ILA works with estimates about the free capacity of nodes, thus it may be impossible to perform the decided migrations. Each failed migration and capacity reservation attempt forc es ILA to update the capacity information it keeps and back-off. Such delay might prove vital because in the meantime a bottleneck node could be created. This particularly affects "GRAL-1 (ILA)" because group migrations (two or more agents destined for the same node) are more likely to fail due to outdated capacity information when capacity is tight. "AMA-1 (ILA)" is less vulnerable to this effect because it only considers single agent migrations. However, it is worth noting that the negative performance impact of ILA in both AMA and GRAL applies only when capacity is scarce and diminishes when capacity increases.

The performance of the ILB scheme comes at a non-negligible cost. Fig. 20 plots the control messages generated by the algorithms. It can be seen that ILB requires roughly one order of magnitude more messages compared to ILA. This is because ILB greedily attempts to obtain locks from all neighbors, before running the actual algorithm that determines the destinations for agent migrations. On the contrary, ILA tries to lock capacity only at the nodes that have been selected as destinations for one or more agent migrations. It is also worth noting that AMA-1 and GRAL-1 generate roughly the same amount of control messages. Another interesting observation is that the number of control messages for ILB tends to decrease as capacity increases. This is

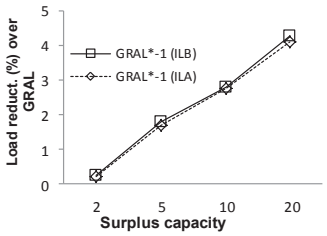

Figure 25. GRAL vs GRAL* when increasing surplus capacity (1-hop variants, large scale)

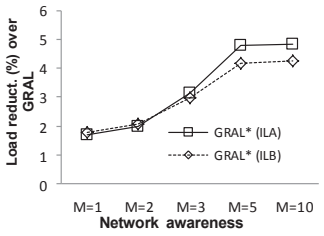

Figure 26. GRAL vs GRAL* when increasing net. awareness (1-hop variants, large scale)

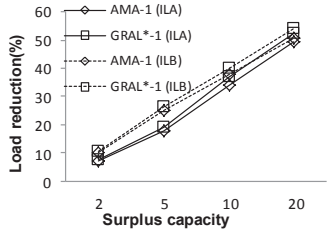

Figure 27. Load reduction when increasing surplus capacity (1-hop variants, large scale)

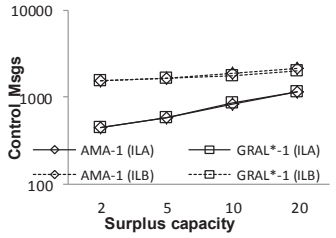

Figure 28. Number of control msgs when increasing surplus capacity (1-hop variants, large scale)

attributed to the fact that with larger free capacity a larger number of migrations will succeed without experiencing back-offs or lock-ins due to filled nodes. This is in line with Fig. 21, which plots the number of migrations. As it can be seen, the number of migrations rises as capacity increases. It can also be seen that when capacity becomes abundant, the GRAL algorithms are able to perform more migrations than the AMA algorithms, which suffer from capacity reservation conflicts.

Fig. 22, Fig. 23, and Fig. 24 show the results for 2-hop variants, i.e., AMA-2 and GRAL-2. Most of the general trends discussed for 1-hop variants hold here too, i.e., load reduction (Fig. 22) and the number of migrations (Fig. 24) rise as capacity increases , whereas the number of control messages (Fig. 23) experiences a slight drop. However, we must note that the performance difference between ILA and ILB becomes minimal for both AMA and GRAL (Fig. 22). This is a very encouraging result considering the fact that ILB is very expensive in terms of control messages (Fig. 23). To explain the above, notice that with a 2-hop network awareness, the number of capacity reservation conflicts for ILB increases as a node can receive requests from a larger number of nodes. Thus, it becomes more likely that some agents will not migrate to the destination(s) assigned to them but rather to a (less optimal) one hop neighbor of it, or not at all. This induces a similar effect to the one observed for ILA for 1-hop awareness. Namely, once back-offs occur and agent migrations are delayed, node capacity may be filled with other agents thereby hindering migrations that would have been more beneficial overall. In fact, notice that ILA quickly closes on and eventually overtakes ILB for GRAL-2 as capacity increases, approaching an optimal result.

Comparing Fig. 19 to Fig. 22, we notice that the 2-hop aware algorithms are superior compared to 1-hop variants. This is because they are in a better position to tackle bottlenecks. The merit of increased network awareness is also verified by comparing Fig. 21 to Fig. 24. It can be seen that 2-hop algorithms perform less migrations than their 1-hop counterparts as they are able to move agents on more distant nodes directly, without having to make consecutive 1-hop migrations which might be eventually hindered due to bottlenecks. Last, comparing Fig. 20 to Fig. 23, note that ILA incurs comparable control message overhead for 2-hop and 1-hop variants, while ILB leads to significantly more control message traffic in 2-hop variants. The reason for this is, once again, the eager nature of ILB, which contacts all neighbor nodes to reserve capacity. As the number of neighbors rises with network awareness, so does the number of control messages generated by ILB. ILA scales significantly better because it relies on information collected as a side effect of previous capacity reservation attempts and ignores with high probability nodes

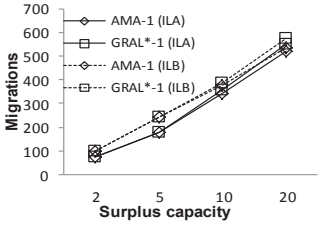

Figure 29. Number of migrations when increasing surplus capacity (1-hop variants, large scale)

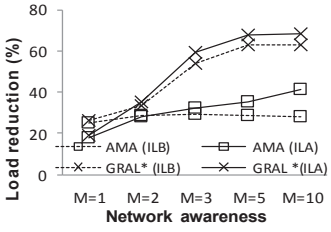

Figure 30. Load reduction when increasing network awareness (capacity=5, large scale)

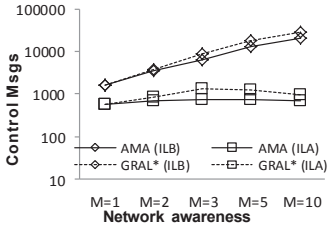

Figure 31. Num. of control msgs when increasing netw. awareness (capacity=5, large scale)

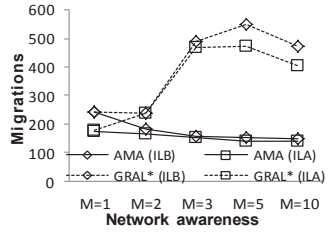

Figure 32. Number of migrations when increasing netw. awareness (capacity=5, large scale)

that have reported no or little free capacity.

6.2.2 Large-scale experiments

In the third set of experiments, we compare the performance of the AMA, GRAL, and GRAL* algorithms in a larger setup. A 50-node network is used in which five instances of app10, app25 and app50 applications are randomly placed (15 applications in total). The load model is *lavg*. We measure the performance of the algorithms as free node capacity increases from two up to 20 agents.

Initially, we show the difference between GRAL and GRAL*. Specifically, Fig. 25 reflects the improvement of GRAL* over GRAL with regards to both locking schemes, as a function of surplus capacity on nodes. As we can see, “GRAL* (ILA)” and “GRAL* (ILB)” achieve up to 5% better performance against their GRAL counterparts. In a next experiment (Fig. 26), we demonstrate the aforementioned improvements, by keeping fixed the surplus capacity at five agents and varying network awareness between one and ten. As demonstrated, “GRAL* (ILA)” and “GRAL* (ILB)” achieve up to 5% and 4% better performance against “GRAL (ILA)” and “GRAL (ILB)”, respectively. It is also observed, that “GRAL-ILA” and “GRAL-ILB” have comparably the same performance against their GRAL’s counterparts when the network awareness is kept at low levels. However, the difference becomes noticeable, when the network awareness becomes more than three. We attribute this to capacity reservation conflicts, which become more likely for ILB as hop awareness increases. A more detailed explanation about this behavior is given by the end of this section.

Fig. 27, Fig. 28, and Fig. 29 depict the performance of the 1-hop algorithms against the three metrics: load reduction, control messages and number of migrations. Similar trends to the ones observed in the smaller experiments are also seen here. One thing worth noticing, is that the increase in the number of control messages of ILA is steeper in the larger setup (Fig. 28) compared to the smaller one (Fig. 20). This is explained by the fact that the number of migrations is also accordingly larger. ILA generates control messages when attempting to reserve capacity after a migration is defined hence the number of control messages naturally increases to the number of successful migrations. In fact, as capacity increases the number of control messages generated by ILA approaches twice the number of (successful) migrations performed, hinting to a dropping number of back-offs (or failed reservation attempts).

In the last experiment, we investigate the effects of *M*-hop awareness. The basic experimental setup is the same as in the previous experiment, but we fix the surplus capacity to five agents per node and instead vary network awareness from one to ten hops. From Fig. 30, we can observe that GRAL* performs significantly better than AMA for both ILA and ILB schemes, clearly demonstrating the superiority of the grouping approach. More specifically, for *M*=5 GRAL* achieves roughly double the load reduction of AMA. The reduction itself is also quite impressive (about 65% of the random initial placement).

Concerning the capacity reservation schemes, ILA clearly outperforms ILB for both AMA and GRAL*, with the difference becoming more pronounced as network awareness increases. Fig. 31 also shows that ILB exhibits an exponential trend with regards to control messages rendering this scheme inherently non-scalable. Looking at Fig. 32, which plots the number of performed migrations, note that GRAL* exhibits a rapid increase as hop awareness increases from 1 to 3, then stabilizing and dropping afterwards. The trend up to 3-hop awareness is due to the fact that increased hop awareness enables the flexible placement of even more agents at even better destinations. Once a good placement is reached, a further increase in hop awareness does not considerably enhance placement quality (see plateau in Fig. 30) but only has the effect of decreasing the number of performed migrations (or more precisely, the consecutive migrations an agent must do to reach a good destination; a trend which is more clear for AMA). The above indicate an essential property of *M*-hop aware algorithms, namely that significant load reduction can be achieved with a relatively small value for *M*. Even larger *M*-hop awareness is not entirely without a positive effect, because it results in a reduced number of migrations and a smaller number of control messages for ILA.

A final remark concerns the fact that the larger number of migrations performed by GRAL*(ILB) compared to GRAL*(ILA) for *M*=3,4,5, actually leads to inferior solutions. We attribute this to capacity reservation conflicts, which become more likely for ILB as hop awareness increases. Such conflicts may lead to the suboptimal mapping of agents on nodes that are further away from their ideal destinations (and closer to their original hosts). In turn, this may create bottlenecks that hinder more beneficial migrations, without necessarily blocking them completely. On the other hand, as hop-awareness increases, ILA can miss migration opportunities due to outdated capacity information, but for the same reason it is also less vulnerable to reservation conflicts, thus, its migrations are more beneficial/effective on average compared to those of ILB. The net effect seems to be in favor of ILA even when performing fewer migrations compared to ILB.

7 Related Work

Placement/migration problems have been tackled in the past under various contexts, e.g., virtual machine placement, task allocation etc. In [20] the authors consider the problem of mapping communicating tasks to homogeneous computing nodes to minimize execution time. In [1] the authors tackle task allocation in an underlying torus network with the target of reducing both task communication and network congestion. Ref. [38] deals with the problems of agent acceptance and maximizing network longevity in WSNs. The data migration problem is tackled in [16] and [17] to rescue utilized bandwidth across multiple data

centers. Task allocation problems are addressed in [9], [23], [30] targeting Network-on-Chip architectures. Last, the service placement problem is tackled in [4] to minimize the total delay experienced by the clients. The above works differ from ours either in the network and application structure assumed [1], [20] as well as in scope [4], [9], [16], [17], [30], [38].

In terms of the problem of virtual machine placement, in [13] the authors tackle the problem of deciding the location of communicating virtual machines to minimize the network congestion. Ref. [5] and [6] propose VM consolidation algorithms to reduce the energy consumption within cloud environments, taking into account SLA violations. The aspect of minimizing the inter-data center latencies by deciding the location of VMs is tackled in [3]. The authors in [24] propose an algorithm that minimizes the total network traffic incurred by the communication between VMs. The difference of the aforementioned work with ours is that their proposed algorithm is based on multi-tier network structures. The most similar work to ours is that of [31] that discusses an algorithm migrating virtual machines towards their center of gravity of the communication load. However, the proposed algorithm does not consider group of migrations and works almost identically as AMA. For the above reason, in the experimental evaluation we present only AMA.

In the context of WSNs, [34] discusses task allocation with the aim to minimize energy consumption. The application model assumed however differs from ours. In [14], the authors study the problem of data dissemination along a tree network. The problem is to define an optimal dissemination tree, which is quite different from the target in this paper. In [42], the authors also consider agent migrations but with the aim of defining optimal paths to collect data. The mobile agent problem is addressed in [18] to balance energy consumption. The authors in [11] and [8] tackle the multi-agent itinerary planning problem. An WSN architecture is proposed in [33] to enable group agent migrations, which is complementary to our work. Last, a novel traffic modeling scheme for capturing network dynamics for WSNs is proposed in [41].In the above papers, the proposed algorithms are not directly applicable here because they differ on their assumptions.

Our work is in the context of agent-based embedded middleware, specifically, POBICOS [32]. Many systems, such as, Rovers [12], Olympus [29], SensorWar [7], Mobile-C [10], MagnetOS [19], Pleiades [15], and DFuse [28], provide environments supporting mobile code and migrations, however enough of them do not support automatic code placement, assigning the complex task of manual code placement to the application programmer. This is a main drawback, because there are environments where the network topology changes dynamically, rendering these systems completely unsuitable to those environments. Therefore, our proposed algorithms could be adopted from such systems to address the problem of automatic, transparent, and dynamic code placement. Even though, there are some above-mentioned middleware initiatives (MagnetOS, Pleiades, and DFuse) supporting dynamic agent placement; to the best of our knowledge none of them considers migrating group of agents, taking in that way into account agent dependencies, which turns out to be an important achievement in terms of network load reduction.

8 Conclusions and Future Directions

In this paper, we tackled the problem of placing the agents comprising an embedded application to the available nodes. The problem is especially important in the context of agent-based embedded middleware design. We proposed distributed asynchronous algorithms to tackle both uncapacitated and capacitated versions of the problem, considering single and group agent migrations. The proposed solutions were extended to tackle cases where WSN nodes have limited capacity. We also provided extended versions of our algorithm to handle increased network awareness. We put extra emphasis to provide solutions such that the underlying WSN is burdened by only a small amount of control messages. It is shown, through an extensive analysis, that AMA's performance is not bounded against an optimal solution. The aforementioned analysis is also accompanied with the maximum performance difference between AMA and the optimal algorithm per time instance. A detailed proof about the approximation ratio of the GRAL algorithm is also given together with the optimality proof of GRAL*. To further evaluate the performance of the proposed algorithms, we conducted a series of experiments. As a future work we will consider to extend our algorithms to work for general application and network graphs. Another consideration is the difficulty of performing both node-specific and node-neutral agent migrations.

References

[1] Z. Abrams and J. Liu, "Greedy is Good: On Service Tree Placement for in-network Stream Processing," *in Proc. ICDCS,* 2006.

[2] T. Agarwal, A. Sharma, A. Laxmikant and L.V. Kale, "Topology-aware Task Mapping for Reducing Communication Contention on Large Parallel Machines," *in Proc. IPDPS*, 2006.

[3] M. Alicherry, T. V. Lakshman, "Network Resource Allocation in Distributed Clouds," *in Proc. INFOCOM*, 2012.

[4] D. Arora, M. Bienkowski, A. Feldman, G. Schaffrath, S. Schmid, "Online Strategies for Intra and Inter Provider Service Migration in Virtual Networks," *in Proc. IPTcomm*, 2011

[5] A. Beloglazov, J. Abawajy, R. Buyya, "Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing," *in Journal FGCS*, 2012.

[6] A. Beloglazov, R. Buyya, "Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers Under Quality of Service Constraints," *IEEE Transactions on Parallel and Distributed Systems*, 2012.

[7] A. Boulis, C.-C. Han, R. Shea, M.B. Srivastava, "SensorWare: Programming Sensor Networks beyond Code Update and Querying", *Pervasive and Mobile Computing Journal*, vol. 3 (4), pp. 386-412 2007.

[8] W. Cai, M. Chen, T. Hara, L. Shu, T. Kwon, "A Genetic Algorithm Approach to Multi-Agent Itinerary Planning in Wireless Sensor Networks," ACM/Springer MONET, vol. 16 (6), pp. 782-793, 2011.

[9] S. Chai, Y. Li, J. Wang, and C. Wu, "An Energy-efficient Scheduling Algorithm for Computation-Intensive Tasks on NoC-based MPSoCs," *Journal of Computational Information Systems*, vol. 9, no. 5, pp. 1817-1826, 2013.

[10] B. Chen, H.H. Cheng, J. Palen, "Mobile-C: A Mobile Agent Platform for Mobile C-C++ agents,"*in Journal Software Practice and Experience*, vol. 36 (15), pp. 1711-1733, 2006.

[11] M. Chen, L. T. Yang, T. Kwon, L. Zhou, M. Jo, "Itinerary Planning for Energy-Efficient Agent Communications in Wireless Sensor Networks," *IEEE Transactions on Vehicular Technology*, vol. 60(7), pp. 3290-3299, 2011.

[12] J. Domaszewicz, M. Roj, A. Pruszkowski, M. Golanski and K. Kacperski, "ROVERS: Pervasive Computing Platform for Heterogeneous Sensor-Actuator Networks," *in Proc. WoWMoM*, 2006.

[13] J. W. Jiang, T. Lan, S. Ha, M. Chen, M. Chiang, "Joint VM Placement and Routing for Data Center Traffic Engineering," *in Proc. INFOCOM*, 2012.

[14] H.S. Kim, T.F. Abdelzaher and W.H. Kwon, "Dynamic Delay-Constrained Minimum-Energy Dissemination in Wireless Sensor Networks," *in ACM Trans. on Embedded Computing Systems*, vol. 4 (3), pp. 679-706, 2005.

[15] N. Kothari, R. Gummadi, T. Millstein, R. Govindan, "Reliable and Efficient Programming Abstractions for Wireless Sensor Networks", *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2007.

[16] N. Laoutaris, M. Sirivianos, X. Yang, P. Rodriguez, "Inter-Datacenter Bulk Transfers with NetStitcher," *in Proc. SICGOMM*, 2011.

[17] Y. Li, H. Wang, P. Zhang, J. Dong, and S. Cheng, "D4D: Inter-datacenter Bulk Transfers with ISP Friendliness," in *IEEE International Conference on Cluster Computing (CLUSTER),* Beijing, China, September 2012, pp. 597-600.

[18] K. Lin, M. Chen, S. Zeadally, J. J. P. C Rodrigues, "Balancing Energy Consumption with Mobile Agents in Wireless Sensor Networks," in FGCS, vol. 28(2), pp. 446-456, 2012.

[19] H. Liu, T. Roeder, K. Walsh, R. Barr, E.G. Sirer, "Design and Implementation of a Single System Image Operating System for Ad Hoc Networks", *in Proc. MOBISYS*, 2005.

[20] V.M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," *in IEEE Transactions on Computers*, vol. 31 (11), pp. 1384-1397, 1988.

[21] T. Loukopoulos and I. Ahmad, "Static and Adaptive Data Replication Algorithms for Fast Information Access in Large Distributed Systems," *in Proc. ICDCS*, 2000.

[22] T. Loukopoulos, N. Tziritas, P. Lampsas and S. Lalis, "Implementing Replica Placements: Feasibility and Cost Minimization," *in Proc. IPDPS*, 2007.

[23] M. Mandelli, L. Ost, E. Carara, G. Guindani, T. Gouvea, G. Medeiros, F. G. Moraes, "Energy-aware Dynamic Task Mapping for NoC-based MPSoCS," *In Proc. ISCAS*, 2011.

[24] X. Meng, V. Pappas, and L. Zhang, "Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement," *in Proc. INFOCOM*, 2010.

[25] M.D. Mesarovic, "Multilevel Systems and Concepts in Process Control," *in Proc. of the IEEE*, vol. 58 (1), pp. 111–125, 1970.

[26] Network Simulator2 (ns2), http://www.isi.edu/nsnam/ns/.

[27] Pan M.-S., Fang H.-W., Liu Y.-C., Tseng Y.-C., "Address Assignment and Routing Schemes for Zigbee-based Long-thin Wireless Sensor Networks," *in Proc. IEEE International Conference on Vehicular Technology*, 2008.

[28] U. Ramachandran, R. Kumar, M. Wolenetz, B. Cooper, B. Agarwalla, J. Shin, P. Hutto, A. Paul, "Dynamic Data Fusion for Future Sensor Networks," *ACM Transactions on Sensor Networks*, vol. 2 (3), pp. 404-443, 2006.

[29] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R.H. Campbell, M.D. Mickunas, "Olympus: A High-Level Programming Model for Pervasive Computing Environments," *in Proc. PERCOM*, 2005.

[30] O. Sinnen, A. To, and M. Kaur, "Contention-Aware Scheduling with Task Duplication," *in JPDC*, vol. 71, no. 1, pp. 77-86, 2011.

[31] J. Sonnek, J. Greensky, R. Reutiman, A. Chandra, "Starling: Minimizing Communication Overhead in Virtualized Computing Platforms Using Decentralized Affinity-Aware Migration," *in Proc. ICPP*, 2010.

[32] STREP/FP7-ICT: Platform for Opportunistic Behaviour in Incompletely Specified, Heterogeneous Object Communities (POBICOS), http://www.ict-pobicos.eu/index.htm.

[33] S. Suenaga, N. Yoshioka, S. Honiden, "Group Migration by Mobile Agents in Wireless Sensor Networks," *The Computer Journal*, vol. 54 (3), pp. 345-355, 2011.

[34] Y. Tian, J. Boangoat, E. Ekici and F. Özgüner, "Real-time Task Mapping and Scheduling for Collaborative In-network Processing in DVS-enabled Wireless Sensor Networks," *in Proc. IPDPS*, 2006.

[35] N. Tziritas, G. Georgakoudis, S. Lalis, T. Paczesny, J. Domaszewicz, P. Lampsas T. Loukopoulos, "Implementation and Evaluation of Agent Mobility for Wireless Networks of Home Objects," *in Proc. SCUBE*, 2012.

[36] N. Tziritas, S. Lalis, S. U. Khan, T. Loukopoulos, C.-Z. Xu, P. Lampsas, "Distributed Online Algorithms for the Agent Migration Problem in WSNs," *in ACM/Springer MONET*, 2013, DOI:10.1007/s11036-013-0452-0.

[37] N. Tziritas, P. Lampsas, S. Lalis, T. Loukopoulos, S. U. Khan, C.-Z. Xu, "Introducing Agent Evictions to Improve Application Placement in Wireless Distributed Systems," *in Proc. ICPP*, 2012.

[38] N. Tziritas, T. Loukopoulos, S. Lalis, P. Lampsas, "Agent placement in wireless embedded systems: memory space and energy optimizations," *in Proc. Int. Workshop on Performance Modeling, Evaluation and Optimization of Ubiquitous Computing and Networked Systems* (*PMEO-UCNS*), 2010.

[39] N. Tziritas, T. Loukopoulos, S. Lalis and P. Lampsas, "GRAL: A Grouping Algorithm to Optimize Application Placement in Wireless Embedded Systems," *in Proc. IPDPS*, 2011.

[40] N. Tziritas, T. Loukopoulos, S. Lalis and P. Lampsas, "On Deploying Tree Structured Agent Applications in Embedded Systems," *in Proc. EUROPAR*, 2010.

[41] P. Wang, I. F. Akylidiz, "Spatial Correlation and Mobility-Aware Traffic Modeling for Wireless Sensor Networks," IEEE/ACM Transactions on Networking (TON), vol. 19 (6), pp. 1860-1873, 2011.

[42] Q. Wu, N. S. V. Rao, J. Barhen, et al., "On Computing Mobile Agent Routes for Data Fusion in Distributed Sensor Networks," *in IEEE Transactions on Knowledge and Data Engineering*, vol. 16 (6), pp. 740–753, 2004.

[43] Zigbee Alliance: Zigbee specification (2006), http://www.zigbee.org.

Biography

Dr. Nikos Tziritas received his B.Sc. degree from the Technological Educational Institute of Serres, Greece, in 2004, and a M.Sc. and a Ph.D. degree from the University of Thessaly, Greece, in 2006 and 2011, respectively. He is currently a postdoctoral researcher in Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His research has been on scheduling, load-balancing and replication in CDNs, as well as energy optimization and resource management in WSNs and cloud computing systems.

Dr. Samee Ullah Khan is an Assistant Professor of Electrical and Computer Engineering at the North Dakota State University, Fargo, ND, USA. Prof. Khan has extensively worked on the general topic of resource allocation in autonomous heterogeneous distributed computing systems. As of late, he has been actively conducting cutting-edge research on energy-efficient computations and communications. A total of 165 (journal: 61, conference: 72, book chapter: 15, editorial: 6, edited proceedings: 2, technical report: 34, technical blogs: 2) publications are attributed to his name. For more information, please visit: http://sameekhan.org

Dr. Cheng-Zhong Xu received the PhD degree in computer science from the University of Hong Kong in 1993. He is currently a professor in the Department of Electrical and Computer Engineering of Wayne State University, and the director of Cloud and Internet Computing Laboratory (CIC) and Sun's Center of Excellence in Open Source Computing and Applications (OSCA). His research interest is mainly in scalable distributed and parallel systems and wireless embedded computing devices. He has published two books and more than 160 articles in peer-reviewed journals and conferences in these areas. He is a senior member of the IEEE.

Dr. Thanasis Loukopoulos received his Diploma in Computer Engineering and Informatics from the University of Patras, Greece, in 1997. He was awarded a PhD degree in Computer Science by the Hong Kong University of Science and Technology (HKUST) in 2002. Currently, he is an Assistant Professor in the Dept. of Informatics at the Technological Educational Institute (TEI) of Lamia, Greece. He has published 25 papers, has been in the program committee of 15 conferences and had the best paper award in ICPP 2001. His current research interests include: green computing, resource management and scheduling problems in distributed systems with emphasis in cloud and wireless environments.

Dr. Spyros Lalis is Associate Professor at the Computer and Communication Engineering Department, University of Thessaly, Greece, and Research Associate at the Center for Research & Technology Thessaly (CERETETH). His main research interests are programming environments, operating systems, software engineering, parallel & distributed systems, and ubiquitous computing systems. He received his Diploma and PhD in Computer Science from ETH Zurich.

Dr. Petros Lampsas graduated from the Department of Computer Engineering and Informatics, University of Patras, Greece in 1994, and he received his PhD degree from the same department in 2001. His main research interests include Distributed Systems, Pervasive Computing and Web Technologies. He has over 40 publications in international journals and refereed conferences. Dr. Lampsas has participated in various European projects and also in projects co-funded by EU and the Greek State. He was senior engineer in Network and Telematics Bureau and IT Projects for Public Sector for six years (1996-2001) in Computer Technology Institute (CTI), Patras. He is currently Associate Professor at the Dept. of Informatics at the Technological Educational Institute of Lamia, Greece.