Resource Management in Heterogeneous Wireless Sensor Networks

H.E. Baarsma, M.G.C. Bosman, J.L. Hurink *

July 7, 2008

Abstract

We propose a first approach in the direction of a general framework for resource management in wireless sensor networks (WSN). The basic components of the approach are a model for WSNs and a task model. Based on these models, a first version of an algorithm for assigning tasks to a WSN is presented. The models and the algorithm are designed in such a way that an extension to more complex models is possible. Furthermore, the developed approach to solve the RM problem allows an easy adaptation, to fit more complex models. In this way, a flexible approach is achieved, which may form the base for many RM approaches.

The possibilities and limitations of the presented approach are tested on randomly generated instances. The aim of these tests is to show that the chosen models and algorithm form a proper starting point to design RM tools.

1 Introduction

In recent years wireless sensor networks (WSN) have been attracting more and more attention. These networks normally consist of many small battery driven devices, which all by themselves have only limited capacities (resources like memory, processors, and energy). They usually are able to perform one or several types of sensing, and can communicate wireless over small distances. A big part of the research on WSNs concentrates on developing control and communication methods (MAC, routing protocols, ...) tailored for these types of networks. The main challenge in this area results from the low resources of the devices. Solving these issues leads to methods and tools, which allow properly working WSNs. These WSNs may be used for specific applications that are programmed in the network.

A next step in the development of WSNs is to allow not only one or a few predefined applications to run on a network, but to set up WSNs on which

*The work presented in this paper was partially funded by the FP6 EU project e-SENSE, Capturing Ambient Intelligence for Mobile Communications though Wireless Sensor Networks, Contract Number: IST-4-027227-IP, www.ist-e-sense.org

applications can be loaded and where the network does not have to know the applications already at the time of the design of the network. In such a setting a new functionality has to be added to the network: a broker like function. This broker has to handle requests of applications that want to be loaded on the WSN, and it has to decide if and how these applications may be loaded on the WSN. In this sense, the broker acts as some sort of resource manager of the WSN.

In this work we discuss issues involving such a broker function. Our aim is to make a first step in the direction of a general framework for resource management in WSNs. More precisely, we want to design this function in the framework of the e-SENSE project. This mainly implies, that the approach has to be integrated with a publish/subscribe mechanism.

We assume that the basic functionality of the resource management (RM) is implemented on a node that forms a gateway of the WSN to the outside world. The RM, on the one hand, needs to be aware of the current status of the WSN and the available functionalities and resources within the WSN. On the other hand, the RM needs to know for all applications which functionalities and resources are requested. This asks for a proper modeling of WSNs and applications. Based on these models, general methods and approaches for resource management in WSNs can be developed.

There has been other research in this area, but a lot of this research has been focussed on homogeneous networks [7]. Some approaches also take heterogeneous networks into account. In [6] a 'query layer' is introduced that focusses on a place where requests for the network can be sent to. These queries can request certain data, but are not as versatile as complete programs would be. In [4] a decentralized agent based approach is used where nodes can offer their resources on a kind of market. This has as an advantage that the approach is decentralized. However, for applications that require the use of several nodes, this approach will most likely not come with good solutions. Having nodes negotiating the best solution among themselves would require too much communication. In [1] a network with only two kinds of nodes is considered, while we focus on a model that allows for an infinite amount of different sensors. Other papers like [2] do not focus on sensor networks, but more on networks in general. Because of this, it is does not focus on reducing network traffic. Our approach is aimed at a RM that can be incorporated in a framework to easily implement task graph based programs in a sensor network [3]. This frameworks is able to deal with many different kinds of (sub-)tasks, nodes and resources and implements tasks in a fast and efficient way.

The paper is organized as follows. In the next section we introduce models for the nodes (devices) of a WSN, for the network structure of a WSN and for the tasks (applications) to be loaded on a WSN. These models form the base of a general definition for the resource management problem (RMP), described in Section 3. We present the constraints and restrictions of the RMP and possible objectives. In Section 4, a heuristic approach for solving the RMP is given. It is based on an iterative assignment of subtasks using priority rules and a backtrack functionality to cope with infeasibility within the assignment process. This backtrack functionality can also be used to develop local search methods. Afterwards, in Section 5 we introduce a scenario with fire detection algorithms.

In Section 6 we test the performance of our algorithm in this scenario, as well as in randomly created test instances.

2 WSN and task model

The aim of this paper is to present methods for task allocation in WSNs. For this, first a clear understanding of WSNs and tasks has to be present. In this section we give a specification of these two main ingredients of the input for RM in WSNs. In Subsection 2.1 we present a model for describing a WSN and in Subsection 2.2 we present a model for the tasks.

2.1 WSN model

The main elements of a WSN are devices and communication possibilities between these devices. Therefore, a WSN in general is modeled as a graph $G = (N, E)$, where the nodes $N = \{N_1, \ldots, N_{NrN}\}$ represent the devices and each existing communication possibility between two devices is modeled by an edge in E.

However, this graph-representation gives only the basic structure of the WSN. On top of this, more detailed information on the devices and the communication possibilities has to be given. The specification of the features and possibilities of the devices asks for a **node model**. For a given device this node model may contain e.g.

- the processing power
- the available memory
- the (remaining) battery capacity
- the geographical location
- the reliability
- the available sensors and their accuracy
- (in case of mobile devices) how mobile a device is
- the current state
- ....

For all of these items, one has to find a proper way to specify these resources. In Table 1 a possible way to represent the availability of the above mentioned resources is given.

Formally, we assume that a set $R = \{R_1, \ldots, R_{NrR}\}$ of resources is given. This set is divided in two parts: resources that can be shared, RS, and resources that must be divided, RD. For each resource $i \in R$ the amount/quality that resource R_i is available at node N_j is given by a non-negative parameter ρ_{ij}. If

Resource	representation and data
Processor	Processor type
Memory	Amount of RAM available
Battery	Fully charged power, percentage empty
Location	Coordinates or group name or object
Reliability	Indicator ranging $[0 \dots 1]$
Sensors	Indication of quality, ranging $[0 \dots 1]$
Staticity	Indicator ranging $[0 \dots 1]$
Current state	Binary Indicator (awake/asleep)

Table 1: Possible elements of a node model

necessary, further parameters can be added to the resources, which e.g. describe the reliability or the quality at which a resource is available.

Besides the nodes, also more specific information is needed on the communication possibilities. The edges E in the graph only indicate between which pairs of nodes communication is physically possible. However, in WSNs in general communication is multi-hop and, therefore, a routing scheme has to be present, that specifies how (i.e. on which path in the graph G) the communication between two nodes is done. We assume that the routing is done via a cluster based routing protocol. This implies, that the node set N is partitioned into a set $C = \{C_1, \dots, C_{NrC}\}$ of (disjoint) clusters and that each node belongs to one of the clusters. Furthermore, each cluster has assigned one of its nodes as cluster head. We assume that all communication from and to nodes is done via the cluster head of the cluster the node belongs to, i.e. a communication between two nodes consists of the direct communication of the two nodes with their respective cluster heads, and a (multihop) communication between the two cluster heads. This implies that only the cluster heads need some sort of routing table and that the 'normal' nodes only need to know their cluster head. For the latter, some extra information has to be added to the node model: the index Cl_j denotes the cluster C_{Cl_j} the node N_j belongs to and $COST_j$ denotes the communication costs per unit (e.g. byte) for node N_j to reach its cluster head.

Besides the cluster heads, which locally control a part of the network and form a backbone of the WSN, also at least one node with global responsibility is given. This node forms the gateway of the WSN and also runs the RM algorithms. In this node a global view on the given WSN has to be available. Thus, the gateway node needs to known which resources are currently available in the different clusters and how the communication is organized between cluster heads (communication paths and costs). We assume that the communication costs are specified by the parameter $ClusterCost_{kl}$, indicating the communication cost per unit between cluster C_k and C_l. Based on these costs the communication cost per unit between two nodes N_i and N_j are given by $Cost_{i,j} = COST_i + ClusterCost_{Cl_iCl_j} + COST_j$.

In this paper we assume, that the clustering scheme is not part of our decision: the partition into clusters and the corresponding routing scheme belong to a

different layer of the WSN and are treated as fixed input to RM. Furthermore, if a WSN uses a different approach for communication, our model is still able to estimate costs well. For example, by assuming every node is the cluster head of its own private cluster, we have the equivalent of a non-clustered multi-hop network.

2.2 Task Model

Tasks are actions, which have to be performed by the wireless sensor network. They consist of subtasks, which may be performed by different nodes in the network. Each subtask has a certain need for resources within the network and the subtasks may be dependent on each other. Furthermore, some extra characteristics of subtasks may be given (e.g. if a subtask can be preempted).

One way to represent tasks is by **task graphs**. A task graph TG consists of a set of nodes $\tau = \{\tau_1, \ldots, \tau_{NrT}\}$ and arcs A, as described in [3]. The nodes represent subtasks, like e.g. using some sensor or adding up different measurements. The arcs represent dependencies between the subtasks that result in a communication between the subtasks.

We assume that for each subtask τ_j the following characteristics are given:

- a non-negative resource requirement α_{ij} of resource R_i; $i = 1, \ldots, NrR$
- a binary indicator ϵ_j taking value 1 if τ_j is a priority task (priority tasks have to start immediately if requested)
- a binary indicator e_i taking value 1 if τ_j can not be preempted

Furthermore, if necessary, information on the energy cost associated with a subtask (e.g. related to the amount of calculations needed) or requests on the reliability or quality which is needed for certain resources to execute a subtask may be given. This subtask model can be improved upon by allowing the resource usage and energy consumption to be dependent of the node a subtask is executed on.

Another element of a task graph is the set A of dependencies between subtasks. A dependency $(i, j) \in A$ expresses that subtask τ_j needs input from the subtask τ_i. We denote by θ_{ij} the amount of data that has to be interchanged between these two subtasks.

3 The resource management problem

In the previous section we have specified the two main ingredients of resource management: the infrastructure given by the WSN and the applications given by task graphs. The resource management problem (RMP) now is to decide how the applications are carried out by the WSN. More precisely, it has to be decided on which nodes which subtasks are executed. In this section we give a precise definition of the RMP. In Subsection 3.1 we introduce the decision

variables of the problem and the constraints on them and in Subsection 3.2 we discuss possible objective function for the RMP.

3.1 Mathematical model-constraints

We have given a WSN modeled by a graph $G = (V, E)$ as given in Section 2.1 and a set of tasks modeled by a task graph $TG = (\tau, A)$ as described in Section 2.2 (note, that we may model a set of tasks also by a single task graph, by introducing for each task one component in the graph).

The only decision which has to take place within the RMP is to decide to which node N_j a subtask τ_i is assigned. For this we introduce binary variables x_{ij} which take the value 1 if and only if task τ_i is assigned to node N_j. A feasible assignment of tasks to nodes has some constraints. These constraints deal with the question whether it is allowed or not to place a task on a specific node; these are the placement constraints.

- The first placement constraint is that each subtask has to be assigned to exactly one node:

$$\sum_{j=1}^{NrN} x_{ij} = 1 \text{ for all } i = 1, \ldots, NrT$$

- The second placement constraint deals with priority subtasks and non-preemption subtasks. Priority subtasks cannot be placed on the same node with other priority subtasks or with subtasks that cannot be preempted. This leads to the following constraint:

$$\sum_{i=1}^{NrT} \epsilon_i x_{ij} + 1/NrT \left(\sum_{i=1}^{NrT} e_i x_{ij} \right) \leq 1 \text{ for all } j = 1, \ldots, NrN.$$

- The third and fourth placement constraints are resource constraints. For the shareable resources the amount of a resource available in a node has to be at least the amount of the resource required by each individual subtask assigned to the node:

$$x_{ij}\alpha_{ki} \leq \rho_{kj} \text{ for all } i = 1, \ldots, NrT, j = 1, \ldots, NrN, k = 1, \ldots, NrR.$$

  For the dividable resources the sum of the required resources of all subtasks assigned to a node must be less or equal than the available resources:

$$\sum_{i} x_{ij}\alpha_{ki} \leq \rho_{kj} \text{ for all } j = 1, \ldots, NrN, k = 1, \ldots, NrR.$$

The above constraints guarantee that each subtask is assigned to a node, that the set of subtasks assigned to a node is consistent and that the nodes have enough resources to run the subtasks. We did not take into account restrictions

on the amount of data which has to be send between the nodes, i.e. we assume that the communication does not form a bottleneck, which is reasonable for many WSNs, especially since our objectives will usually encourage minimal communication. However, the communication is of importance for judging the quality of an assignment, since communication is the biggest factor in energy consumption in WSNs. Therefore, we incorporate the communication into the objective function.

3.2 Mathematical model-objectives

One important aspect of RM is a good definition of the objective. Depending on the specific setting and use of a WSN, several objectives are possible, some of which are presented here. Each of the objectives is explained, together with a brief description of the important resources and the essential information needed. Afterwards, a concrete mathematical description of one of the objectives is given. The other objectives may be formalized in a similar way.

- **Minimizing the power usage** tries to minimize the total battery power needed to run the given tasks. Essential elements are the power costs for each subtask (dependent on the type of (sensor) node) and the network traffic costs (which can influence the selection of nodes on different locations).
- **Maximizing the lifetime of the network** tries to maintain the battery power level above a certain lower bound as long as possible, such that no node fails in an early stage. In addition to the necessary power and traffic costs in "Minimizing the power usage" also the remaining battery power is needed to develop a scheduling method.
- **Maximizing the reliability** maximizes the reliability of the completion of the tasks. This depends on the reliability of the sensors (whether sensors can collect information or not), the reliability of the nodes (which can fail, such that the sensors on this node cannot be reached anymore) and the reliability of the traffic in the network (which effects in some packets that are lost or cannot be sent). Maximizing the reliability assigns subtasks to the nodes, such that the expectation that all tasks are completed is as high as possible.
- **Optimizing objectives with a lower bound on the reliability** tries to optimize a certain objective (such as minimizing the power usage) while guaranteeing a certain lower bound on the reliability. Information regarding both objectives is essential to do useful resource management.
- **Minimizing network traffic** minimizes the total generated traffic through the network (traffic load). In this case not the costs of the traffic but the amount of data sent through the network is important for the objective.
- **Maximizing the quality** schedules the tasks, such that the subtasks get assigned to nodes, which provide the needed resources with a high quality. The quality of execution may depend on the quality of sensors

(for example precision), but also on the number of sensors used and the sensor type or combination of sensor types used.

In the following, we give a more formal description of the first objective in this list. Minimizing energy consumption is very important in WSNs, and although in theory less important than network lifetime, it is an important factor in maximizing network lifetime. We consider two main cost factors: the communication costs and the costs for nodes to be awake.

The communication costs per unit for each node to reach its cluster head and the communication costs per unit between all cluster heads are given by parameters $Cost_{ij}$ for a pair (i, j) of nodes. For each arc $(i, j) \in A$ we introduce a non-negative variable y_{ij} representing the communication costs between the subtasks τ_i and τ_j resulting from the assignment of these two subtasks to nodes of the WSN. Theses costs can be bounded from below by the following inequality:

$$y_{ij} \geq Cost_{kl}\Theta_{ij}(x_{ik} + x_{jl} - 1) \text{ for all } k, l = 1, \ldots, NrN; (i, j) \in A.$$

Since in the objective function these costs will always be minimized and since both subtasks τ_i and τ_j are assigned to precisely one node, the value y_{ij} takes the correct value.

The second part of the cost deals with the status of the nodes (awake or asleep). We introduce a binary variable O_i, which should take value 1 if node N_i is awake and value 0 if node N_i is asleep. First, we have to ensure, that all cluster heads are awake:

$$O_i = 1 \text{ for all nodes } N_i \text{ which are clusterheads.}$$

Furthermore, a node must be awake if a subtask is placed on that node:

$$O_i \geq x_{ji} \text{ for all } i = 1, \ldots, NrN;\ j = 1, \ldots, NrT.$$

Using these variables a simple variant of the objective function is given by:

$$\min \sum_{(i,j)\in A} y_{ij} + \sum_{i=1}^{NrN} OnCost_i O_k,$$

where $OnCost_i$ denotes the costs resulting from node N_i to be awake.

In our objective function we neglect the costs for the use of resources, we only count the costs for keeping a node awake and the use of communication. This might not be very accurate, but on the other hand it is often difficult to get precise estimates of the power use for every resource. If we scale $OnCost_i$ properly we can still obtain a good estimate of the actual costs, especially if the costs of using a certain resource are equal for all nodes.

The constraints mentioned in the previous subsection together with the above objective function leads to an Integer Linear Progam (ILP) model. We have used an ILP solver package to solve this ILP for several test scenarios. Unfortunately, the large number of integer variables may result in long computation times (up to days) to find the optimal solution for some scenarios. This is of course not acceptable if we want to use on-line resource management. Therefore, in the following section we describe a heuristic approach to find solution for the considered problem.

4 A heuristic approach

This section describes an efficient heuristic to find a solution to the RMP. In general, such a heuristic should aim at

- minimizing the network traffic,
- minimizing the number of nodes that have to be awake,
- planning as many subtasks as possible on a node.

We propose an iterative heuristic assigning the subtasks one by one to the nodes. The basic structure of the method is as follows:

1. Determine for every subtask a priority and sort the subtasks based on this priority in a list L.
2. **WHILE** L contains still some unassigned subtasks **DO**
   (a) Select the first unassigned subtask from L (let τ_j be this subtask).
   (b) **IF** no feasible assignment of τ_j exists (i.e. a deadlock) **THEN**
      i. give task τ_j a higher priority in L,
      ii. withdraw all assignments and resort L,
      iii. start again with Step 2.
   (c) Assign subtask τ_j to a node.

The above heuristic forms a sort of framework, where special elements still have to be specified in more detail: the chosen priorities in Step 1, the increase of priority in a deadlock situation in Step 2(b), and the concrete calculation of the assignment criteria in Step 2(c). In the following we describe how we have realized these issues in our implementation.

4.1 Priority in Step 1

Besides a random priority we used three different priority rules.

- **Hard to place**
  Subtasks, which have only a small number of possible nodes to be assigned to, are in general more difficult to handle than subtasks which have a large number of possible nodes. Therefore, we sort L based on non-decreasing number of nodes it can be assigned to.
- **Link importance**
  This priority is based on the assumption that communication should take place within the clusters as much as possible. In order to do so, we sort L in three steps. First we sort L based on non-increasing number of links to other subtasks. Next we sort the subtasks with equal numbers of links on non-increasing amount of required resources. Finally, subtasks belonging

to the same task graph are put right after each other, such that a list sorted on task graph origin is formed. The order of the task graphs depends on the first appearance of a subtask of the corresponding graph.

- **Following the task graph**
  This ordering is tiered like the previous one, but works slightly different. We first pick the subtask with the most links to other subtasks. After this initialization we look every time for the subtask which has most of its linked subtasks placed. In case of a tie we pick the subtask with the most links to other subtasks. This way we assign subtasks in an order that ensures we have more knowledge of the placement of its linked subtasks, allowing us to minimize communication costs more efficiently.

4.2 Increase of priority in Step 2(b)

If in a given iteration a certain subtask cannot be assigned to a node anymore, the assignments done in previous iterations have taken away possible assignments of the current subtask. Therefore, this subtask should be assigned earlier. We have chosen for the extreme to put the subtask in first position of L. It may be worth, to investigate less extreme reactions in future research.

4.3 Assignment criteria in Step 2(c)

We propose two different ways to assign subtasks to nodes.

- **Version 1**
  Since we deem communication costs the most important factor in power usage, we choose to first try to minimize these costs. As a secondary criterion we concentrate on the sleep mode. Finally, we take into account that we might save communication costs by giving room for other subtasks to be assigned to the same node as the current node. The assignment thus looks as follows:

  1. If subtasks with a dependency with τ_j (i.e. subtasks τ_i with $(i, j) \in A$ or $(j, i) \in A$) have already been assigned, assign subtask τ_j such that the communication costs with these already assigned dependent subtasks is minimized.
  2. If in (1) the minimal communication costs are achieved for several assignments, choose among these assignments one which assigns τ_j to a node which already has to be on due to the current partial assignment.
  3. If due to (1) and (2) still more than one node is left, assign τ_j to the one with maximal remaining resource capacity.

  The criteria (1) to (3) can be evaluated easily for each alternative, to assign a subtask. To reduce the overhead for these calculations, it is worthwhile to keep track of some data (e.g. the set of already active nodes and the remaining resource capacities on the nodes). For criteria (3) one has

to define, how the remaining resource capacities are compared. Possible choices are to concentrate on one particular resource (e.g. the battery or memory capacity) or take a weighted sum of different resource capacities. In our implementation, we take the sum of all resources with equal weights.

- **Version 2**
  The second version looks ahead more than the first one. It tries to reduce future communication costs by placing subtasks on clusters with a lot of resources available. The influence of already active nodes is smaller in the second version. The concrete assignment is the following:

  1. If subtasks with a dependency with τ_j (i.e. subtasks τ_i with $(i, j) \in A$ or $(j, i) \in A$) have already been assigned, assign subtask τ_j such that the communication costs with these already assigned dependent subtasks is minimized.
  2. If in (1) the minimal communication costs are achieved for several assignments, choose among these assignments one which assigns τ_j to a node in the cluster that has the most resources available.
  3. If due to (1) and (2) still more than one node is left, assign τ_j to the one with maximal remaining resource capacity.
  4. If due to (1), (2) and (3) still more than one node is eligible, we try to assign τ_j to a node that is already active.

  In the criteria (2) and (3) we take again the sum of all resources (in the cluster or on the node respectively).

The above discussion shows, that the presented heuristic is a flexible approach which can be adapted to the specific situation to be treated. Furthermore, the heuristic may also form the base of a more advanced approach. If we consider each possible ordering of the list L of subtasks as input for the above heuristic, we may get a lot of different solutions for the RMP. Since a complete enumeration of all orderings is surely not a suitable approach, one can incorporate the above heuristic in a local search framework (e.g. simulated annealing) and search on the set of possible orderings. Since for some orderings of L the algorithm might not find a feasible solution, it is useful to check the feasibility for different orderings if this occurs. If during our algorithm a certain subtask can not be assigned to a node, we move this subtask to the front of L and start our algorithm again. This way "difficult" subtasks are placed first. This approach makes it more likely we find a feasible solution, although it can lead to a deadlock if this process is repeated indefinitely. Therefore we recommend that when implementing this algorithm, the number of reruns is restricted, for example to a certain percentage of the number of subtasks.

The above heuristic may also be used as a 'repair' method. If for example a node fails, we can simply rerun the algorithm with all the tasks that were on that node. Furthermore, to improve a given solution, we may also decide to reimplement some subtasks, for example a subtask that leads to large communication costs or a subtask which is the only task running on a certain node.

5 Example: a fire detection system

For testing the presented approach, next to random generated instances, also some instances are used which represent a simple fire detection system running on a WSN using different sensor types. The different sensor types may include temperature measurement, smoke detection and humidity. These instances are chosen to get some more insight in the way the develloped approach works. The random instances are used to see how the method scales on larger and more complex instances.

Based on the information given in [5], two possible types of task graphs to implement a fire detection system on a WSN are designed. In one implementation each individual node decides on its own to raise the alarm or not. In this case, the task graph has the simple structure given in Figure 1. A second implementation assumes that the node can also use the sensor data of several other nodes (e.g. the nodes with the same type of sensor or nodes in its neighborhood) to make its decision (Figure 2).

Another approach for a fire dection system may be to use sensed abnormal differences (in temperature, humidity or smoke) to initiate more accurate measurements before an alarm is given. This means that there are three task levels:

1. standart mode (once in a while a measurement is done)
2. detecting mode (detecting if abnormal measurements are due to fire or due to measurement faults)
3. high alert mode (if a fire is discovered)

A system like this makes use of priority tasks, like described in our task model. The task graph for this approach looks much like Figure 1, but with a second round of sensing after a node moves the system in the detecting mode.

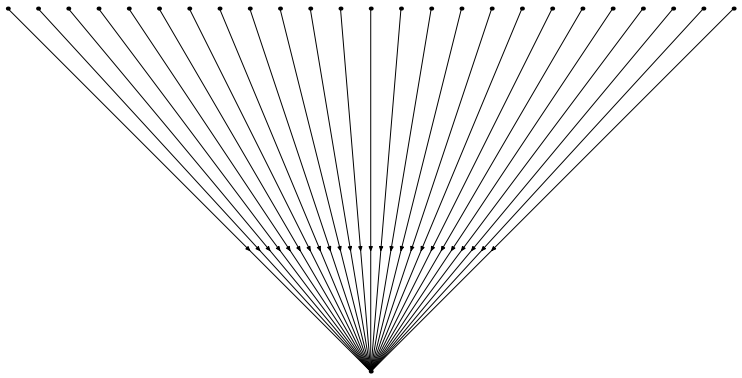

Figure 1: Task graph of fire detection

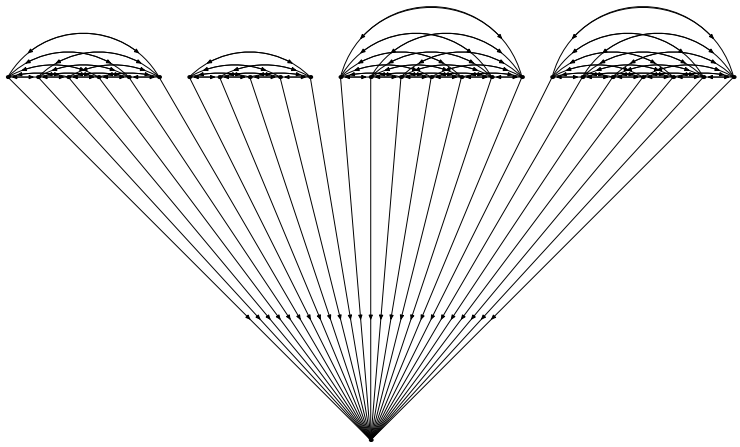

Figure 2: Task graph of fire detection with neighborhood correction

6 Testing

In this section we evaluate the effectiveness of the presented heuristic. First, we use the fire detection scenario and compare the solutions achieved by the heuristic with solutions achieved by a greedy algorithm and with the optimal solutions calculated via the ILP formulation. After that we compare different versions of the develloped heuristic on large randomly generated data sets, to see how the different versions scale.

6.1 The fire detection scenario

For the fire detection scenario, we map task graphs consisting of 5-10 subtasks and using two resources (temperature measurement and smoke detection) on networks with 10-20 nodes, uniformly divided over 4 clusters. The instances of the first type (Figure 1) are denoted by *Fire1* and the instances of the second type (Figure 2) are denoted by *Fire2*. For the later instances, each temperature node collects measurements from all other nodes that measure temperature and each smoke node collects measurements from all other nodes that are assigned to detect smoke. The results of the applied approaches are given in Table 2. The table contains the costs of

- the optimal solution: 'ILP costs', (if solving the ILP to optimality took too long, only the best found solution by the solver (UB) is given);
- the solution achieved by the presented algorithm, where the subtasks are ordered by the sum of their required resources and subtasks are assiged as described in Version 1: 'Version 1 costs';
- the solution achieved by the presented algorithm, where the subtasks are ordered by Link Importance and the subtasks are assigned as described in Version 2: 'Version 2 costs';
- the solution achieved by a greedy approach: 'Greedy costs'.

Case	Subtasks	Nodes	ILP costs	Version 1 costs	Version 2 costs	Greedy costs
Fire1	5	10	12383	12383	12383	23033
	8	15	12180	12180	12180	25632
	10	20	(23221 UB)	23221	23221	28848
Fire2	5	10	20136	26065	26065	44214
	8	15	(43066 UB)	43066	43066	53179
	10	20	(80489 UB)	111846	80489	89306

Table 2: Test Results for the fire detection scenario

We can see that our algorithm performs close to or at the optimal costs. Unfortunately in several cases the ILP solver was unable to give us an answer that was a guaranteed optimal solution, even for these small instances. This shows the need for good heuristics.

6.2 Randomized Test Instances

For further testing we used randomly created instances with a number of fixed properties. We tested the following things:

- How well does the algorithm solve instances with an increasing number of nodes and subtasks
- How well does the algorithm solve instances with an increasing number of subtasks
- How fast does our algorithm work with an increasing number of nodes and subtasks
- How fast does our algorithm work with an increasing number of subtasks

We designed our test instances in the following way. Nodes and subtasks have 3 possible resources. There is a 33% probability that a certain resource is not present on a node or required by a subtask. The average node has 5 times more resources than required by the average subtask. These parameters were chosen to create a good representation of an actual sensor network with many heterogeneous nodes.

Communication costs from nodes to their cluster heads are picked from a uniform distribution between 2 and 8. Communication costs between cluster heads are chosen between 15 and 45. These costs are symmetrical. To assign resource requirements to tasks , we pick a number from a uniform distribution between 0 and 15 and then subtract 5 and if the result is a negative number we change it to 0. To assign resource availabilities to nodes, we pick a number from a uniform distribution between 0 and 75 and then subtract 25 and if the result is a negative number we change it to 0.

We then have to assign nodes to clusters. We start with assigning a node to every cluster. These nodes will be the cluster heads. Every other node is assigned to a cluster, based on uniform distribution. The next step is to define

our task graphs. We divide our tasks into k task graphs, by randomly picking $k-1$ distinct numbers between 0 and the number of sub asks. This way we define k non-empty intervals. The lengths of these intervals give us the sizes of the task graphs. To generate the dependencies between the subtasks of a task graph, we number all subtasks. Then if a task graph consists of subtasks τ_n until and including τ_m, for every subtask τ_i with $n \leq i < m$ we have a 50% probability for each subtask τ_j $(i < j \leq m)$ to depend on τ_i. We check every subtask except τ_m has at least one other subtask that depends on it. This way we ensure connectivity within the task graphs. The tests were performed on a Pentium 3 type desktop computer, with our algorithm implemented in C++.

We first tested the algorithm to see how the costs of assigning increase as we increased the number of nodes and subtasks. This way we can compare our assignment and ordering methods. Per instance, we used the same number of nodes as subtasks, ranging from 100 to 1000 nodes. We ran every scenario ten times to get good averages. The results are given in Figure 3. This figure clearly shows that the algorithm with Link Importance ordering in combination with the Version 2 method of placing subtasks is superior. It is around 20% better in all cases. In Figure 4 the average runtimes of the different versions of our algorithm are given. All algorithms show similar runtimes. We can see that for instances of 100 nodes and smaller, the algorithms finish within a second (making on-line applications possible) and for networks as big as 1000 nodes, our algorithm still finishes within a reasonable amount of time.

For the results in Figure 5 we used the same parameters as in Figure 4 but this time with a smaller probability of having a dependency between two subtasks. We use 10% instead of 50%. We see that with these sparser task graphs there is less difference in the performance of our algorithm.

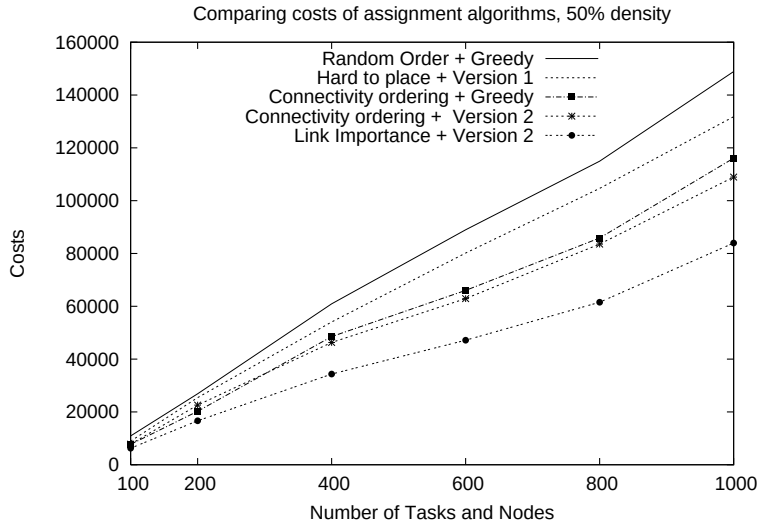

Figure 3:

Another test was seeing how increasing the number of subtasks per node influences the performance of our algorithms. We used the fixed number of 200

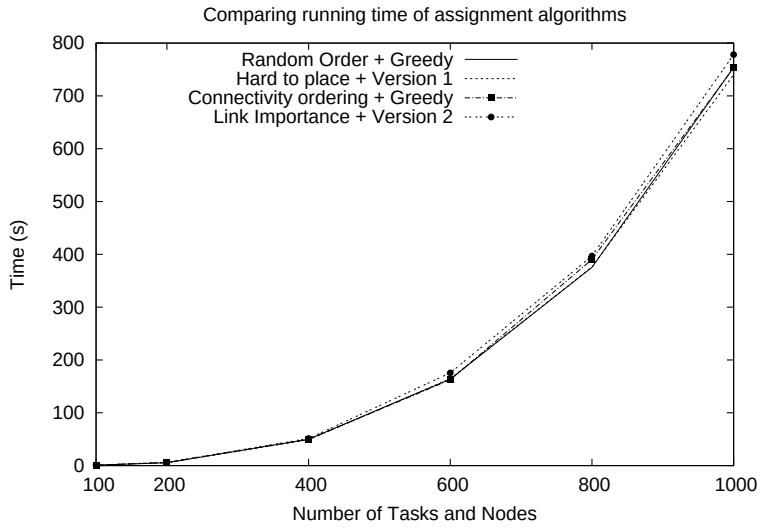

Figure 4:

nodes and increased the number of subtasks from 100 to 650. Adding more subtasks would lead to too many infeasible problems. We reran our algorithm a maximum of 10 times if it encountered an infeasible ordering of the subtasks. Figure 7 we compare the running times of our algorithms for the second series of tests. Running times rise sharply around 600 subtasks. This is because more reruns of the algorithms are required. We see that the Hard to Place + Version 1 algorithm has excellent performance, in terms of running time. This is not surprising since it tries to place the most difficult tasks first. It also has better performance in the number of solved problems as can be seen in Table 3. In Figure 6 we show the costs of average costs of assigning the problems. We see that like in Figure 3 Comparing costs of assignment algorithms the Link Importance + Version 2 algorithm has the best performance. Near 650 subtasks, the costs appear to go down. This is caused by the fact that we deleted insolvable problems. Because of this the rest of the problems might be easier on average. We also tested if the number of task graphs has any effect on the relative performance of different methods, but performance remains mostly the same.

7 Conclusion

We have developed a basic model to make resource management decisions for heterogeneous WSNs. The model is designed in such a way that an extension to more complex models is possible. Furthermore, the developed heuristic approach to solving the RM problem allows an easy adaptation within the different steps, to fit more complex models. In this way, we believe that the presented model is sufficiently flexible to form the base model for future RM approaches. To get some insight into the quality of the solutions achieved by developed approach, we compared for a small-scale test set the calculated optimal solution to the presented heuristic. The solutions the heuristic gave are quite close to opti-

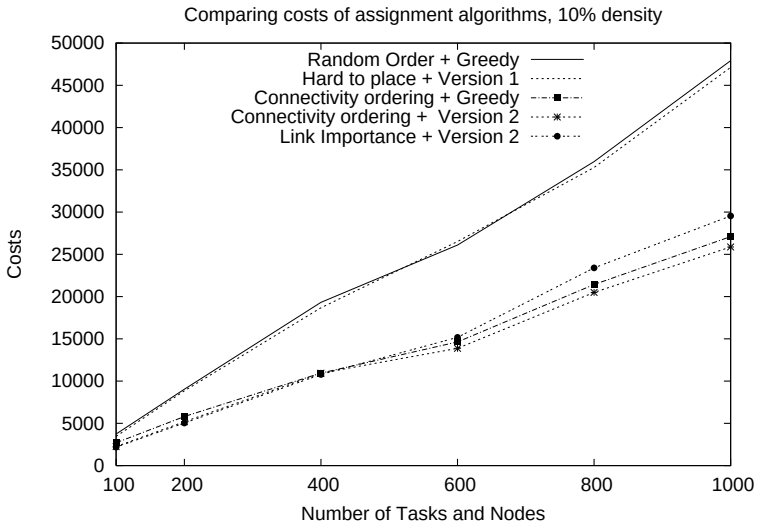

Figure 5:

mal and, thus, can be considered as useful. Furthermore, they are significantly better than using a dumb greedy approach and the computation times are not very large. The letter implies that the heuristic is fast enough to make online decisions about RM. Since within the heuristic we can use several realizations of the different steps, we have a certain degree of flexibility. For example, we may use the Link Importance + Version 2 algorithm as basic approach, but if we do not find a solution after a number of iterations, we can use the Hard to Place + Version 1 algorithm since this version is more robust. All in all, the tests indicate that the presented framework for RM in WSN forms a promising approach and can act as base for developing specific approaches for concrete real world settings.

References

[1] Enrique J. Duarte-Melo and Mingyan Liu. Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks. In *GLOBECOMM02*, 2002.

[2] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of the International Workshop on Quality of Service*, 1999.

[3] Clemens Lombriser, Daniel Roggen, Mathias Stäger, and Gerhard Tröster. Titan: A tiny task network for dynamically reconfigurable heterogeneous sensor networks. In *15. Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, pages 127–138, February 2007.

[4] Geoffrey Mainland, David C. Parkes, and Matt Welsh. Decentralized, adaptive resource allocation for sensor networks. In *NSDI'05: Proceedings of the*

Algorithm	Subtasks	Feasible problems
Random order + Greedy	400	10
	500	9
	550	8
	600	3
	650	1
	700	1
Hard to place + Version 1	400	10
	500	10
	550	10
	600	10
	650	9
	700	8
Connectivity ordering + Greedy	400	10
	500	10
	550	7
	600	2
	650	1
	700	0
Link Importance + Version 2	400	10
	500	10
	550	7
	600	5
	650	1
	700	1

Table 3: Number of feasible instances

2nd conference on Symposium on Networked Systems Design & Implementation, pages 315–328, Berkeley, CA, USA, 2005. USENIX Association.

[5] M. Marin-Perianu and P. J. M. Havinga. D-fler: A distributed fuzzy logic engine for rule-based wireless sensor networks. Technical Report TR-CTIT-07-54, Enschede, 2007.

[6] Y. Yao and J. Gehrke. Query processing in sensor networks. In *Proceedings of Conference on Innovative Data Systems Research*, 2003.

[7] Yang Yu and Viktor K. Prasanna. Energy-balanced task allocation for collaborative processing in wireless sensor networks. *Mob. Netw. Appl.*, 10(1-2):115–131, 2005.

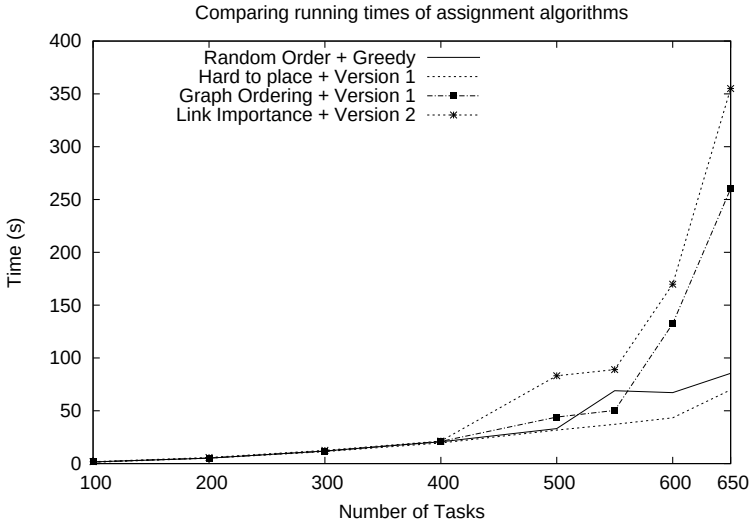

Figure 6:

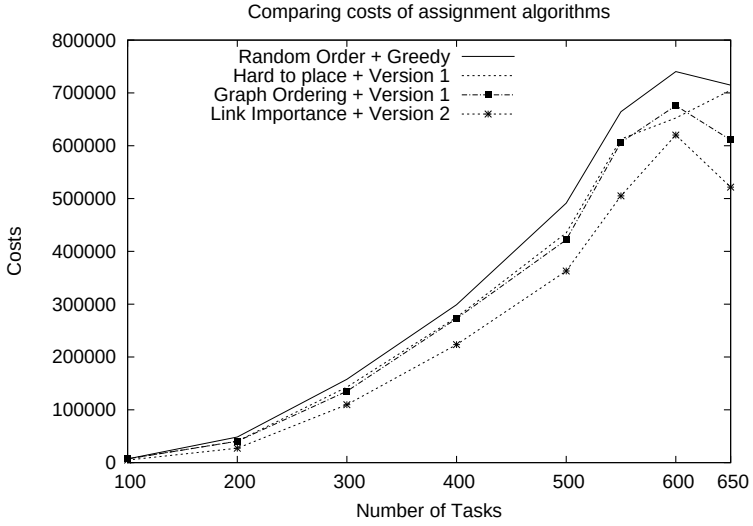

Figure 7: