

QoS Management for Wireless Sensor Networks with a Mobile Sink

Rob Hoes^{1,2}, Twan Basten^{2,3}, Wai-Leong Yeow⁴, Chen-Khong Tham¹,
Marc Geilen², and Henk Corporaal²

¹ National University of Singapore ² Eindhoven University of Technology
³ Embedded Systems Institute ⁴ Institute for Infocomm Research

Abstract. The problem of configuration of Wireless Sensor Networks is an interesting challenge. The objective is to find the settings, for each sensor node, that optimise certain task-level QoS metrics. An existing configuration method is available for tree-based static networks. We extend this method to support a mobile sink. First, the routing tree is adapted to the sink's new location, after which the parameters of the nodes are optimised. Both algorithms are distributed and localised, and therefore efficient and scalable, and are able to flexibly trade reconfiguration cost (time and energy) for quality to match the demands of the application. Various options are analysed, and evaluated in simulation.

1 Introduction

The research on Wireless Sensor Networks (WSNs) in recent years has focused extensively on hardware, operating systems and communication protocols. Quality-of-Service (QoS) management, in which one places constraints on performance criteria on several observable quality metrics such as lifetime, reliability, or delay, is becoming an important area of interest as well. A relatively new, but very significant topic deals with the question of how to configure a WSN that has been designed and deployed to fulfil a certain task. Sensor nodes all have a number of hardware or software settings that can be individually set. Carefully fine-tuning these parameters can yield significant performance gains. The problem is very challenging due to the vast number of possible configurations a WSN has, a number that grows exponentially with the size of the network.

In earlier work, we provided a solution to the configuration problem for static networks [1]. The solution is a scalable configuration method that intelligently searches through the full configuration space and delivers Pareto-optimal solutions or *Pareto points*: the best possible trade-off points. However, sensor networks are often dynamic, and the only way this method is able to deal with events such as moving nodes, is by completely reconfiguring the network.

This paper extends the configuration method by providing facilities for efficient reconfiguration upon a move of the data sink. Supporting a mobile sink is of interest for lifetime extension, as it relieves the energy bottleneck that naturally exists at nodes close to the sink, which need to transfer much more data than nodes further away [2, 3]. Furthermore, the application may have the need for a mobile sink, for example in disaster-recovery situations in which rescue workers walk around with handheld devices to collect information about the scene.

The configuration method assumes that a routing tree is used for communication between sensors and the sink. If the sink moves, the tree likely breaks, and needs to be fixed. Furthermore, the change in situation may have rendered the current configuration sub-optimal, or worse, QoS constraints may have been violated. Efficient reconfiguration of the tree and node parameters requires distributed and localised algorithms. In this paper, we describe an algorithm for tree reconstruction that is able to trade the cost of reconfiguration in terms of time and energy, for the quality of the new tree measured by the average path length. The algorithm selectively reconfigures a local area around the sink of which the size can be adjusted, while guaranteeing that a correctly rebuilt tree results. This goes hand in hand with a localised QoS-optimisation scheme that is able to find new Pareto points given that only a sub-set of the nodes may be touched. Both methods and their practical and seamlessly integrated implementation, are the main contributions of this paper.

2 Related Work

Many researchers recognise the need for methods that deal with conflicting performance demands and set up a sensor network properly. Some authors suggest to use a knowledge base to make a match between task-level demands and network protocols to use [4, 5]. These efforts choose a mode of operation that is common for all nodes in the network, while our configuration method determines settings for each node individually. MONSOON [6] is a distributed scheme that uses agents to carry out application tasks, while the behaviour of these agents is adapted to the situation at hand according to evolutionary principles. Lu et al. [7] also look at WSN configuration in a method for address allocation. The overhead of the configuration protocol itself is optimised, but unlike our approach, the performance of a higher-level application is not. None of the existing configuration approaches explicitly deals with adaptation to sink movement.

Several topology-maintenance schemes have been suggested earlier [8,9], some of which aim at mobile sinks or targets [10,11]. The tree-reconstruction method of Zhang et al. [12] comes closest to our method, as they also flood a restricted area. However, our way of combining such restricted flooding with a baseline mechanism that ensures connectivity is new. By doing this, we enable a wide range of possible trade-offs between maintenance costs and task-level quality metrics. Furthermore, contrary to many existing approaches, our tree-reconstruction method does not require any knowledge about the deployment of nodes or movement of the sink, and is robust to message loss. Moreover, our way of integrating topology control with node configuration to meet task-level QoS goals is unique.

3 Configuring a WSN with QoS Requirements

In this section, we give an overview of the configuration process and the type of networks we target. Our dynamic reconfiguration method follows the same general steps. The focus of this work is on large networks of static sensor nodes

positioned in some target area. In addition, there is sink node, whose job is to collect the data from the WSN. We assume all nodes have similar communication capabilities and all network links are symmetric. The network has a specific task, such as tracking a target or creating a map of the area based on measured data. We further assume that a routing tree is used to connect each node to the sink.

The process of configuring a WSN consists of a number of phases and can be executed in both a centralised or a distributed fashion. The first configuration phase is the construction of the tree. The goal of this phase is to create a tree that is good in terms of the quality of the task running on the network, but also tuned to relax the complexity of the following phase. Given the tree, the QoS-optimisation phase is responsible for finding a Pareto-optimal set of configurations (see below) for the parameters of all nodes in the network, in terms of a number of quality metrics. An algorithm to do this efficiently was introduced earlier [1]. One of the found configurations is then chosen based on constraints and other considerations, and loaded into the network (the loading phase).

3.1 Routing-Tree Construction

A routing tree has two properties that are especially relevant for the configuration problem: the average path length and the maximum node degree. Short paths in the tree are favourable for delay metrics. The degree of a node is defined as the number of child nodes it has. A low maximum node degree leads to good load balancing, and therefore to a better lifetime. Furthermore, the complexity of the QoS-optimisation algorithm greatly depends on the degree of nodes (see below). We assume that the quality of the task improves if the average path length is reduced for a given maximum node degree, and vice versa.

As minimising the average path length and maximum node degree are conflicting goals, the challenge is to find a suitable trade-off between them. We developed a method that finds a shortest-path spanning tree (SPST) within a given maximum-degree target [13]. First, the network is flooded to construct an SPST. In practise, flooding does not always result in an SPST, but here we accept this sub-optimality in exchange of efficiency and ease of implementation. Subsequently, each node that has a degree higher than the target tries to reduce its degree by requesting some of its children to find a suitable other parent node. Further details are not important for this paper, and can be found in the reference.

3.2 QoS Optimisation

Each node has *parameters*, which are hardware or software settings that can be controlled, and every parameter has a set of possible values. Suppose a node has three parameters with three possible values each; then there are $3^3 = 27$ possible parameter vectors per node. In a network of n nodes, with $3n$ independent parameters, this leads to 27^n possible *configurations* for the whole network. Each parameter vector of a node maps to a vector of measurable properties called *quality metrics* for this node. Likewise, a parameter vector of all nodes together maps to a vector of quality metrics for the whole network, which are our

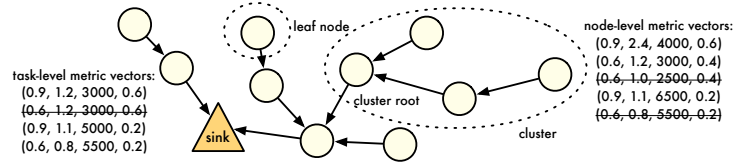


Fig. 1. Illustration of a network, cluster, leaf node, and node/task-level quality-metric vectors (tuples of information completeness, detection speed, lifetime and coverage degree); dominated configurations are crossed out.

optimisation targets. Such mappings can be captured in equations, and together form a model of the WSN and the task running on it. Figure 1 shows an example of a network, configurations, and related concepts.

An example of a model for a target-tracking task is available in [1]. In this task, one or more targets move around in an area of interest. Any node that detects a target, sends a report to the sink node. The model has four task-level quality metrics: *information completeness*, the fraction of the data messages from the sensor nodes that arrive at the sink; *detection speed*, a measure for the delay from detection of a target to notification at the sink; *lifetime*; and finally *coverage degree*, the minimum percentage of the time that each part of the area is observed by at least one sensor. There are inherent trade-offs between these metrics.

The challenge is to find a set of network configurations (vectors of parameters plus metrics they map to) that are *Pareto optimal*. A configuration $\bar{c}_1$ *dominates* a configuration $\bar{c}_2$ if and only if $\bar{c}_2$ is not better than $\bar{c}_1$ in any of the quality metrics. The Pareto-optimal configurations, or *Pareto points*, of a set of configurations are all configurations that are not dominated by any other configuration in the set, and are therefore considered to be the best. The process of finding all Pareto points of a configuration set is called *minimisation* [14].

An obvious way to find the Pareto points for a WSN is to compute all configurations and minimise this set. However, since the total number of configurations for a WSN typically increases exponentially with the number of nodes, this is generally not feasible. A solution is to find the Pareto points of groups of nodes called *clusters*, and incrementally grow these groups. At each step, dominated configurations are removed to keep the sets small. First, every node is initialised as a one-node cluster. This means that all parameter vectors are mapped to cluster-level metrics, and the resulting configuration set is minimised. Then, two or more clusters are combined, cluster metrics for this higher-level cluster are derived using cluster-to-cluster mappings, and the resulting configuration set is minimised. This is repeated until there is a single cluster left, which contains all nodes. The quality metrics of this cluster are the task-level quality metrics.

It is not possible to combine any arbitrary group of clusters into a new cluster, and then minimise the new configuration set, without the risk of losing configurations that are Pareto optimal at the task level. We need a property called *monotonicity*: a clustering step is monotone if and only if dominated configurations from the clusters that are being combined can never be part of a non-dominated configuration in the combined cluster. If all clustering steps are

Algorithm 1. Distributed QoS-analysis algorithm (runs in each node)

```

1 wait until each child node has transferred its Pareto set
2 create one-node cluster Pareto set
3 combine one-node cluster and child-clusters Pareto sets (if any)
4 derive quality metrics of new configuration set
5 minimise configuration set
6 send minimised configuration set to parent

```

monotone, dominated configurations can be safely removed in each clustering step, without potentially losing Pareto-optimal network configurations.

It has been proven [1] that a clustering step is monotone if two conditions are met. Firstly, the cluster-to-cluster mappings need to be monotone (they need to be non-decreasing functions), which is the case for the target-tracking model. Further, every cluster must form a sub-tree of the routing tree. Therefore, a cluster is redefined to be a root node together with *all its descendants* in the tree (all downstream nodes up to and including the leaf nodes). This imposes a clustering order that starts with the leaf nodes of the network, and grows clusters towards the sink. A straightforward distributed algorithm is possible, in which each node runs the program given in Algorithm 1. After the sink completes the program, it has the complete set of Pareto points for the whole network: the quality metrics plus parameter values for each node that achieve these. Next, the sink chooses a feasible Pareto point, and sends it down the tree (the loading phase).

The run time of this algorithm for a single node is roughly proportional to the product of the number of configurations in each Pareto set that is combined, and therefore heavily depends on the number of children the node has. This is one of the reasons to build a routing tree with low node degrees. It is shown that the algorithm, while having an exponential theoretical worst-case complexity, is sub-linear in practise, and therefore scalable to large networks.

To reduce memory usage, and communication in the loading phase, an indexing method is used. Each node maintains an indexing table that links each configuration in its Pareto set to the associated configurations of its child nodes, and does not store configurations of individual nodes in its cluster. In the loading phase, a node only needs an index from its parent to know which parameters to use, and sends indices to its children after a look-up in the indexing table.

4 Adapting the Routing Tree

The previous section shows how to configure a WSN from scratch. We now consider a configured WSN in operation, in which the sink moves to a new location. To ensure the network's task is able to continue running, and its quality meets the demands in the new situation, a reconfiguration of the network is needed. As the routing tree most likely breaks when the sink moves, the tree needs to be reconstructed (see Figure 2(a)). We assume that the sink moves stepwise, and after each step resides at its position long enough to justify rebuilding the tree. For applications with a continuously and relatively fast moving sink, maintaining

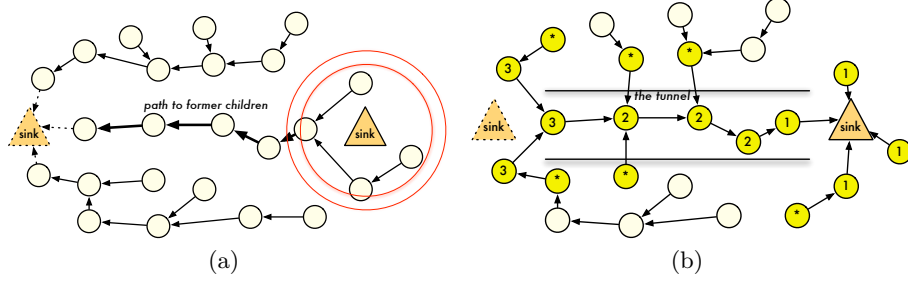


Fig. 2. The sink moved from left (dashed triangle) to right. (a) The nodes that have dashed arrows became disconnected after the move of the sink. Thick arrows indicate a path from the new position of the sink to these nodes. (b) Dark-coloured nodes form the affected area of QuickFix. The number indicates the QuickFix phase they perform.

a routing tree is probably not the best solution and other methods of delivering data to the sink may be more suitable.

As outlined in Section 3.1, our goal is to create a tree in which all nodes have a degree no more than a certain threshold, and paths are made as short as possible within that constraint. After a move of the sink, the cost of globally reconstructing the tree (in time and energy) may be too high, especially if moves are frequent. We therefore propose a new algorithm that recreates the tree only in a certain region around the sink, referred to as the *affected area*, and retains the parts of the existing tree elsewhere, thereby sacrificing some quality (longer paths).

4.1 Minimal-Cost Reconstruction

We consider full tree reconfiguration as a baseline algorithm that provides the best quality against the highest cost. At the other end of the spectrum would be an algorithm that has the lowest reconfiguration cost, but a lower quality as well. This algorithm ensures that all nodes are connected to the sink again, and is therefore required for a minimum service level, but does nothing beyond that to improve the quality. We call this algorithm *QuickFix*. It is similar to the Arrow protocol introduced in a different context [15]. We first explain QuickFix, and then explain under which assumptions it works properly.

After the sink moves to a new position, it may be out of range of some or all of its children in the existing tree. QuickFix creates new paths from these nodes to the sink. All other nodes in the network are descendants of the former children of the sink. Therefore, reconnecting these former children to the sink means that all nodes are connected again. Reconnection is based on the observation that the existing tree has paths to the sink's former child nodes from anywhere in the area, and happens in three phases; see Figure 2 for an overview. The sink, at its new position, starts by broadcasting a control message. The nodes that receive this message become the sink's new children, which all have a path to one of the disconnected former children of the sink. In the second phase, QuickFix follows these paths and reverses the links, until such a former child is reached. Finally,

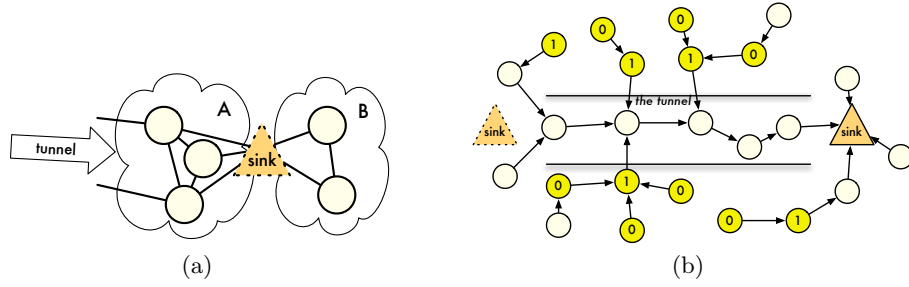


Fig. 3. (a) Former children of the sink, in two disconnected groups. Group A is connected to the tunnel, but none of these nodes can reach a node in group B by broadcasting. Group B's nodes (plus all descendants) remain disconnected after QuickFix. (b) The example of Fig. 2 with Controlled Flooding (CF) having deviation = 1. Dark-coloured nodes have been affected by CF and some of them (those in the bottom part) have found a shortcut to the sink. The numbers indicate the deviation values per node.

this node broadcasts a control message that enables other former children of the sink to connect, which is repeated until all are reconnected.

QuickFix effectively creates a *tunnel*, containing all above mentioned paths, through which all disconnected nodes are reconnected to the sink. This has no effect on the node degrees (except the sink's), but all paths are enlarged by paths of the tunnel. An optimisation that does not cost any extra transmissions can be made: any node that overhears a control message may set the sender of this message as its parent node (nodes marked by an asterisk in Figure 2(b)). By doing this, the node creates a shortcut to the tunnel and shortens the path of itself and its descendants.

QuickFix leads to a tree containing all nodes under the following conditions:

1. The sink's broadcast after the move is received by at least one sensor node. If not, the first phase breaks.
2. QuickFix messages creating the tunnel are not lost. If this happens, the second phase breaks.
3. The sub-network consisting of only the former children of the sink is fully connected. If not, the third phase breaks (see Figure 3(a)).

The first condition implies that the sink needs acknowledgements from its new children that have received its broadcast. If no acknowledgement is received, the sink rebroadcasts. The next condition can also be guaranteed by an acknowledgement scheme, as all transmissions are unicast. The third condition will generally be met if the node density (with respect to the radio range) is sufficiently high. This will usually be the case, as WSNs are typically very dense.

4.2 Improving the Quality

QuickFix reconnects all nodes to the sink in a highly cost-efficient way, but the average path length of the resulting tree will be high. The affected area consists of

only the old and new children of the sink and the tunnel between them. To reduce the average path length, but keep the costs limited, we use another mechanism on top of QuickFix, which enlarges the affected area by a number of hops that can be specified, called the *deviation* parameter. This parameter is part of the control-message format. Any node that overhears a control message does not only connect to the sender (as described in the previous sub-section), but if the deviation value is larger than zero, it will broadcast a new control message with a decremented deviation value. We refer to this as *Controlled Flooding* (CF). By flooding the affected area, a local SPST is constructed, and consequently also the paths of the nodes outside the area are reduced in length (see Figure 3(b)).

QuickFix and CF are not executed consecutively, but run in parallel. To ensure that nodes react to a control message only once, an update number is used in the control-message format. This number is incremented at each reconfiguration (sink move), and only if a node receives a control message which has a higher update number than it has seen before, it will update its parent variable and forward the message. There is one exception to this rule: since QuickFix is crucial to reconnect all nodes in the new tree, QuickFix control messages are always forwarded (to the parent in the *old* tree!), even though a CF message with the same update number arrived earlier. Since all chains of forwarded control messages (QuickFix and CF) originate from the sink, each affected link is pointed to the node that sent the message, and nodes react only once to CF messages of a certain update number, a correct tree (rooted at the sink, loop-free) is formed in the affected area. The nodes at the edge of the affected area keep their existing sub-trees, so all nodes are connected to the affected area and hence to the sink. Loss of CF messages may lead to longer paths, but never results in a broken tree.

After QuickFix and CF finish, the node degrees are reduced as before. Only nodes in the affected area take part in the degree reduction. When only QuickFix is used, the reduction algorithm is not able to do much, since the affected area is small. Therefore, we bound the number of nodes that may directly connect to the sink already in the first phase of QuickFix via some extra handshaking.

The deviation parameter controls the trade-off between reconfiguration cost and quality. A larger deviation value leads to a larger affected area, and thus to more nodes obtaining shorter paths, and a better quality. On the other hand, reconfiguring a larger affected area takes more time and more transmitted control messages. The best value for the deviation parameter depends on the application.

5 Reconfiguring Node Parameters

Normal operation of the network task can continue as soon as the tree has been reconstructed. However, due to the changes in the structure of the network, the level of quality achieved by the running task is typically lower than possible, and could even be such that QoS constraints are violated. It is therefore worthwhile to improve the quality by reconfiguring the nodes' parameters.

While parameter reconfiguration is in progress, the network is in a state of reduced quality. It is therefore desirable to reconfigure as quickly as possible. More-

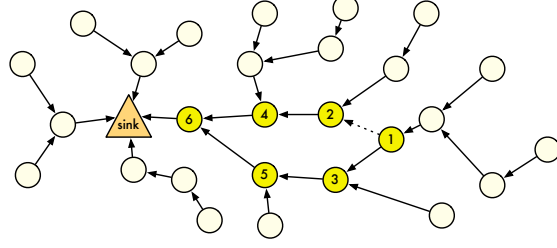


Fig. 4. Node 1 has changed from node 2 to node 3 as its parent. Only the dark-coloured nodes and the sink (the *affected area*) need to recompute their Pareto points.

over, parameter reconfiguration comes at a cost, as processing and communication is needed to compute and load the new settings. In this section, we explore the trade-off between the quality achieved by reconfiguration and the cost it has.

5.1 Optimisation Strategies

We first discuss how to find the best possible node configurations in terms of quality, given the reconfigured tree. The most straightforward, but inefficient, way to do this is to simply re-run Algorithm 1 on all nodes, from the leaves up to the root. Observe however, that if the tree reconfiguration only happens in a local area around the sink, many nodes and their sub-trees/clusters remain unchanged. Therefore, also the sets of Pareto-optimal configurations for all nodes and clusters outside the affected area do not change, and need not be recomputed.

To verify this, first consider a fully configured network in which a single node changes its parent (for whichever reason), as in Figure 4, where node 1 switches from node 2 to 3. This would cause changes in the Pareto set of the cluster with root node 1. Further, the roots of all other clusters that have changes in them need to be updated (remember that a cluster is a node with *all* its descendants): the clusters with as root node 1, its old and new parent (nodes 2 and 3), and all nodes on the paths from these three nodes to the sink (nodes 4, 5 and 6). This implies that the QoS-optimisation algorithm may start at the nodes at the edge of the affected area (further referred to as the *boundary nodes*; in Figure 4, nodes 1 and 2 are boundary nodes) instead of at the leaf nodes of the network. The reconfiguration of the affected area reuses the Pareto sets of the clusters just outside the area. Note, however, that the newly selected configuration at the sink, may cause a different configuration to be selected from the Pareto sets of the nodes outside the affected area. This means that, while recomputing the Pareto sets is local, loading the selected configuration still involves all nodes in the network.

To make the reconfiguration completely local, not only the tree reconstruction and QoS analysis phases, also the loading phase should be restricted to a local area. Nodes outside the affected area should retain their configurations, and this should be taken into account in the QoS-analysis: not the full Pareto sets of the non-changing clusters should be used, but only the selected configuration. A boundary node then combines its new one-node cluster set with the selected configurations of each of its children. This has the added benefit that the analysis

becomes simpler (smaller configuration space), significantly reducing the cost of reconfiguration. The price is sub-optimality of the found task-level Pareto set and hence a potentially non-optimal quality of the selected configuration. To further exchange quality for lower cost, we could reduce the area even more (smaller than the area of tree reconfiguration), the extreme case being not reconfiguring at all. However, not re-analysing all nodes in the affected area means that the computed task-level metrics are not accurate; when locally reconfiguring from boundary nodes to root, the computed metrics are always accurate.

5.2 Practical Details

From the previous, it follows that we need to make the boundary nodes start Algorithm 1 with the correct child Pareto sets. However, a node actually does not know whether it is a boundary node or not, and what is more, not every node knows that it is a part of the affected area. In the example of Figure 4 in which node 1 changed parents, only nodes 2 and 3 will be aware of the change, while also 4, 5, and 6 need to be updated. We therefore make every changed node send a message to its parent (after some delay to ensure the tree is stable) that indicates it is part of the affected area. If the parent was not a changed node, it now knows that it is also in the affected area, and forwards the message to its own parent. Subsequently, to start the analysis process, a node outside the affected area that overhears such a message from its parent (all light-coloured nodes in Figure 4 that are children of dark coloured-nodes), knows that its parent is a boundary node, and forwards its unchanged cluster-level Pareto set, or only the currently selected configuration in case of localised reconfiguration. Nodes in the affected area can now continue as in Algorithm 1.

After completing the QoS-analysis phase, the sink proceeds with the loading phase as usual. When the load messages reach outside the affected area, they are no longer forwarded in the localised case. In the globally-optimal case, the load messages are forwarded until the leaf nodes of the network.

6 Experiments

Since the performance of our reconfiguration approach depends on many factors, such as the type of task, the size of the network, and the movement pattern of the sink (size of steps, speed), we use simulations to compare various scenarios. We are especially interested in the influence of the deviation parameter and the choice between localised and globalised QoS analysis on resulting task quality and reconfiguration costs. To see whether parameter reconfiguration really makes sense, we also compare the results with the option of not reconfiguring at all (but we do always need QuickFix, as the tree always needs to be fixed after a move).

6.1 Simulation Overview

All simulations were done in the OMNeT++ discrete-event simulator [16], for networks of 900 TelosB sensor nodes [17] randomly deployed in an area

of 300×300 m. These nodes employ a simple 8 MHz processor and a radio transceiver with 250 kbps bit rate. To ensure an even distribution of the nodes in the area, they were placed with some variance around fixed grid points. The communication range of the nodes was set to 20 m. The first scenario we look at has the sink placed at coordinates (100,150), and the network configured with the existing method [1]. Then the sink makes a single move to position (200,150), and the network is reconfigured using the various options described in this paper. We simulated 100 different networks and report the medians of the metrics of interest. In the second scenario, the sink makes multiple consecutive moves, while the network is reconfigured after each move.

Our simulations take into account the processing time of Algorithm 1 on real TelosB sensor nodes. Profiling of a TinyOS implementation on such a node was done to obtain the run times for various sizes of the configuration sets (see [13] for details). All simulations were done for the target-tracking task [1] and 27 different configurations per node. We distinguish four cases: naive and efficient global reconfiguration, local reconfiguration, and no reconfiguration at all. The former two refer to re-analysing all nodes and only the affected nodes respectively, which should both arrive at the same globally-optimal Pareto points.

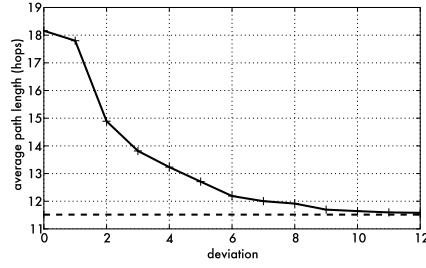
A selection function is needed that chooses one of the Pareto points to be loaded into the network. For easy comparison of the various methods, we use a selection function that assigns a single value to a configuration. We use a weighted sum of all four metrics, where each weight normalises the metric. To this end, we define the *value* of a configuration vector $\bar{c} = (\text{information completeness, detection speed, lifetime, coverage degree})$ (see Section 3) as its inner product with the vector $\bar{v} = (100, 200, 0.1, 100)$.

The evaluation metrics are as follows:

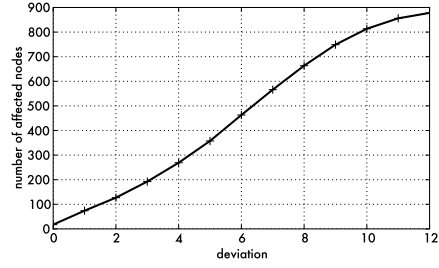
- **Disruption time:** the duration of service disruption just after the sink’s move until the tree has been reconfigured, and thus equal to the time needed for tree reconstruction. During the parameter-optimisation time, the network does function, though its service quality is reduced.
- **Reconfiguration time:** the total duration of the tree- and parameter-reconfiguration process. The total reconfiguration time is also a rough indication of the amount of processing needed on the nodes (the optimisation time is dominated by processing).
- **Communication cost:** the average number of bytes transmitted for reconfiguration, over all nodes in the network.
- **Value loss:** the relative loss in value compared to the best case.

6.2 Tree Reconstruction

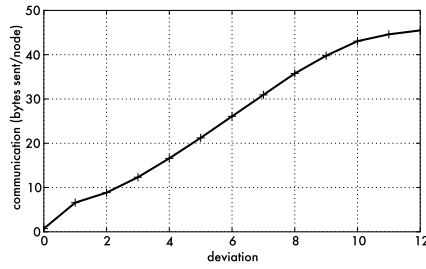
We first study the behaviour of the tree-reconstruction algorithm described in Section 4 in the single-move scenario. All 100 networks were tested with various values of the deviation parameter, as well as full flooding as a baseline. The degree-reduction algorithm with a target node degree of 2 (see Section 3.1) was executed on the resulting networks. The first point to note is that the tree



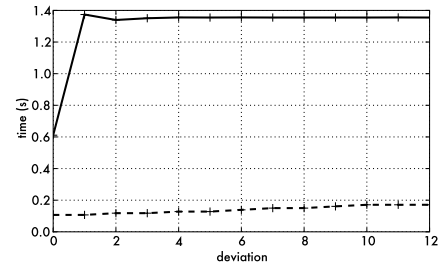
(a) Average path length (the dashed line is the optimum)



(b) Size of the affected area



(c) Communication cost



(d) Total disruption time (solid line) and only QuickFix+CF (dashed line)

Fig. 5. Evaluation of tree reconstruction for various deviation values.

was correctly rebuilt in all of the cases. Figure 5(a) shows that average path length decreases monotonically from almost 18.2 to 11.5 when increasing the deviation from 0 (only QuickFix) to 12. The optimal average path length (when fully flooding the network) is also 11.5. Figure 5(b) indicates that the size of the affected region also grows steadily with the deviation until, at deviation 12, almost the whole network is reconfigured, and hence we obtain an SPST with this deviation (within the degree constraint). Observe that the affected area first grows faster than linearly, but slows down after deviation 6; this is the point where Controlled Flooding reaches the edges of the network.

Along with the affected area, the amount of communication increases in a similar pace (Figure 5(c)). As expected, also the time it takes to rebuild the tree, excluding degree reduction, increases with the deviation (Figure 5(d), dashed line), with an offset due to QuickFix. The time needed for QuickFix/CF is relatively short compared to the time used to reduce the node degrees, and together with the fact that the latter does not depend on the size of the affected area, this explains why the total time spent on tree reconstruction (solid line in Figure 5(d)) is not clearly dependent on the deviation. Also recall that if only QuickFix is used, the degree-reduction algorithm is hardly effective due to the

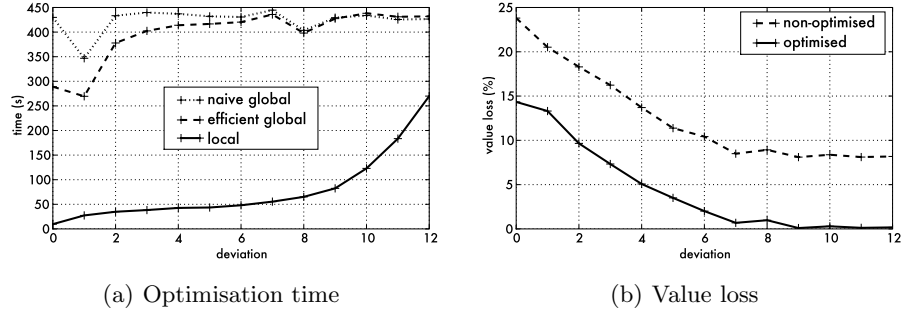


Fig. 6. Timing results and value improvement for parameter optimisation.

small affected area, which is why the run time at deviation 0 is lower than for the others. Overall, the total disruption time is always less than 1.4 s.

6.3 Parameter Optimisation

Now compare the optimisation times of the various cases of parameter optimisation in Figure 6(a). We confirm that efficient global reconfiguration, in which only nodes in the affected area recompute their Pareto points, is always faster than the naive version, and this is most pronounced for small deviations. However, the differences are not as large as might be expected. Local optimisation on the other hand, in which the same nodes recompute their Pareto points as in the efficient global case, but with boundary nodes using just one configuration for their child nodes outside the affected area (instead of their full set of Pareto points), is much faster. This may imply that the configuration sets of nodes closer to the sink are larger than those of nodes further away. It is interesting to see that the optimisation time of the localised algorithm initially increases very slowly (sub-linearly) with the deviation, starting at just 9.7 s. Deviation 5 appears to be an inflection point beyond which the rate of increase grows quickly. Eventually, the three lines meet at 418 s (about seven minutes; not visible in the graph), when fully flooding the network; this is equivalent to deviation infinity, as the affected area is the whole network.

Next, we test the quality of the resulting configurations by comparing their values. The best value occurs when using full flooding and global parameter optimisation. Using this value as a baseline, Figure 6(b) shows the relative loss in value when using the other methods. It turns out that, for the target-tracking task, all methods for parameter optimisation, local and global, achieve the same quality (the solid line), while not optimising is significantly worse (8 to 10 percentage points; the dashed line). Given its low overhead, this makes local reconfiguration very attractive for any deviation. The best value possible when not optimising (at deviation 12) can be attained with optimisation already at deviation 3. For a larger deviation, we see a steady improvement in value, which is consistent with our assumptions in Section 3. At deviation 0, the difference with the optimum is quite large at 14.3%, but after deviation 6 it becomes smaller than 1%.

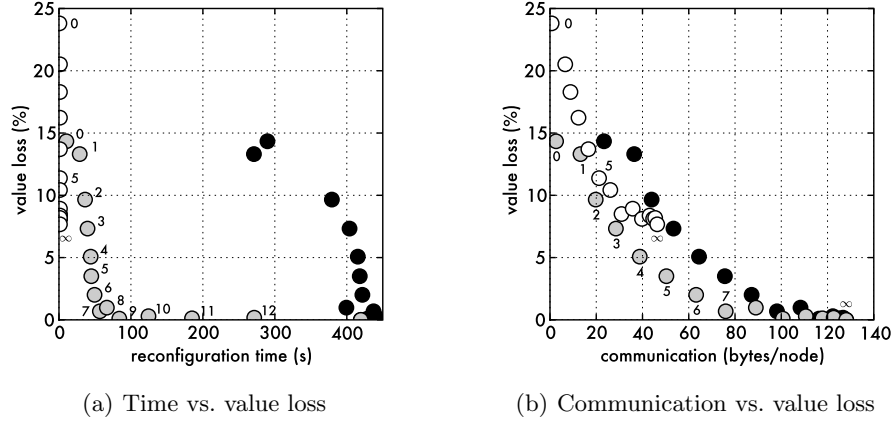


Fig. 7. Pareto plots for the reconfiguration process. Black dots belong to (efficient) global parameter reconfiguration, grey to local, and white to no parameter optimisation. Deviation values are given at interesting trade-off points.

6.4 Quality/Cost Trade-offs

Combining all results yields the totals for the reconfiguration process in the evaluation metrics. The disruption time only depends on tree reconstruction and has been reported above. Figure 7 gives an overview of the trade-offs between the total time and communication costs of reconfiguration, and the value loss of the resulting configuration, for all reconfiguration options. The two plots should be seen together as a three-dimensional trade-off space. It is immediately clear that all global-reconfiguration points (the black dots) are dominated by the locally optimised (grey) and non-optimised (white) options. In contrast, most of the other points are Pareto optimal. As non-optimisation is obviously the fastest, it is the best choice when speed and low processing costs are most important, although the loss in value is at least 7.7%. In many cases, however, local reconfiguration provides the best trade-off between the three metrics: low cost and good quality. At deviation 5, for example, local reconfiguration takes 44.9 s (of which the service is disrupted for 1.4 s), costs 50.3 bytes of communication per node, and the overall quality is 3.5% lower than the best case. The configuration space that was analysed in this time, for the affected area of 350 nodes, has a size of 27^{350} configurations, of which the found configuration has the optimal value.

6.5 Multiple Moves

It is interesting to see what happens to the loss of value when the sink moves repeatedly, and the network is locally optimised at each move. Figure 8 shows the value loss of local optimisation with deviation 5 compared to the optimal case, for 25 consecutive moves of 50 m in random directions. The results are averages over five different runs. A very irregular pattern is visible, but the trend is a

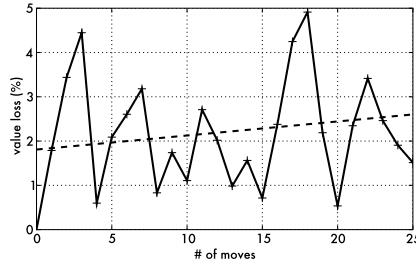


Fig. 8. Multiple consecutive moves: value loss compared to optimal with trend line.

slowly increasing loss for each move. We therefore suggest to do a full network reconfiguration periodically, or when the attained value becomes too low.

7 Conclusions and Future Work

This paper provides a method for reconfiguration of a WSN with a mobile sink. It does not only reconstruct the routing tree, but also optimises the parameters (hard- or software settings) of the nodes that were affected to improve the service level. Since reconfiguration takes time and energy, which are scarce resources, optimally reconfiguring the network each time the sink moves is likely to be infeasible. Trade-offs can be made between the effort spent on optimisation and the resulting level of service quality of the running task in terms of QoS metrics. The algorithms work in a distributed and localised way by reconfiguring only an area around the sink, of which the size can be adjusted via a deviation parameter. Practical implementation details are given, and experiments show that the localised algorithms indeed manage to find suitable trade-off points.

The best choice of reconfiguration method and deviation value heavily depends on the application and its requirements, and the sink behaviour. Due to the unpredictable nature of the sizes of the Pareto sets and therefore the optimisation time, as well as the quality of the resulting task-level Pareto set, it is hard to give guidelines for this choice. In practise, a system could first have a calibration phase to tune the deviation value, or simulations like ours can be used.

The methods that do all processing in-network are useful for applications in which the sink stays at its position for a while before moving again (e.g. when moving the sink for lifetime improvement), to justify the cost and speed of reconfiguration. For scenarios such as disaster recovery, in which the sink (rescue worker with handheld) may move a bit faster, an interesting option is to deploy special, more powerful nodes that handle most of the optimisation duties, or even do all the work at the sink. This is again a trade-off: between communication and processing cost (offloading computation effort increases the amount of communication between sensors and sink), and of course the cost of the additional nodes. For example, doing the QoS analysis for deviation 5 as above on a laptop (Intel Core 2 Duo processor at 2.4 GHz) takes 3.0 s for the globally-optimal case, and just 0.6 s for the localised case. For handheld devices,

these numbers would be higher, but still much lower than when done in-network on sensor nodes. However, the communication costs per node increase by about five times (both global and local). Future work will focus on this trade-off in more depth.

Acknowledgements. The authors would like to thank Dr. Vikram Srinivasan for his help in the early stages of this project. This work was partly supported by EC FP6 project IST-034963.

References

1. Hoes, R., Basten, T., Tham, C.K., Geilen, M., Corporaal, H.: Quality-of-service trade-off analysis for wireless sensor networks. Elsevier Performance Evaluation (2008) <http://dx.doi.org/10.1016/j.peva.2008.10.007>.
2. Luo, J., Hubaux, J.P.: Joint mobility and routing for lifetime elongation in wireless sensor networks. In: INFOCOM '05, Proc., IEEE (2005)
3. Wang, W., Srinivasan, V., Chua, K.C.: Using mobile relays to prolong the lifetime of wireless sensor networks. In: MobiCom '05, Proc., ACM (2005) 270–283
4. Pirmez, L., Delicato, F., Pires, P., Mostardinha, A., de Rezende, N.: Applying fuzzy logic for decision-making on wireless sensor networks. In: Fuzzy Systems Conference '07, Proc., IEEE (2007) 1–6
5. Wolenetz, M., Kumar, R., Shin, J., Ramachandran, U.: A simulation-based study of wireless sensor network middleware. Network Management **15**(4) (2005) 255–267
6. Boonma, P., Suzuki, J.: MONSOON: A coevolutionary multiobjective adaptation framework for dynamic wireless sensor networks. In: HICSS '08, Proc., IEEE (2008) 497–497
7. Lu, J., Valois, F., Barthel, D., Dohler, M.: Fisco: A fully integrated scheme of self-configuration and self-organization for wsn. In: WCNC '07, Proc., IEEE (2007) 3370–3375
8. Cerpa, A., Estrin, D.: ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. In: INFOCOM '02, Proc., IEEE (2002) 23–27
9. Schurgers, C., Tsiatsis, V., Srivastava, M.B.: STEM: Topology Management for Energy Efficient Sensor Networks. In: Aerospace Conference, Proc., IEEE (2002)
10. Luo, J., Panchard, J., Piórkowski, M., Grossglauser, M., Hubaux, J.P.: Mobiroute: Routing towards a mobile sink for improving lifetime in sensor networks. In: DCOSS '06, Proc. (2006)
11. Bhattacharya, S., Xing, G., Lu, C., Roman, G.C., Harris, B., Chipara, O.: Dynamic Wake-up and Topology Maintenance Protocols with Spatiotemporal Guarantees. In: IPSN '05, Proc., IEEE (2005)
12. Zhang, W., Cao, G.: Optimizing tree reconfiguration for mobile target tracking in sensor networks. In: INFOCOM '04, Proc., IEEE (2004)
13. Hoes, R.: Configuring Heterogeneous Wireless Sensor Networks Under Quality-of-Service Constraints. PhD thesis, TU/e and NUS (2009) (to appear).
14. Geilen, M., Basten, T., Theelen, B., Otten, R.: An algebra of Pareto points. Fundamenta Informaticae **78**(1) (2007) 35–74
15. Demmer, M., Herlihy, M.: The arrow distributed directory protocol. In: DISC '98, Proc., Springer (1998) 119–133
16. OMNeT++. <http://www.omnetpp.org>
17. Crossbow Technology. <http://www.xbow.com>