Analysing QoS Trade-offs in Wireless Sensor Networks

Rob Hoes[1], Twan Basten[2], Chen-Khong Tham[1], Marc Geilen[2] and Henk Corporaal[2]

[1]Department of Electrical and Computer Engineering, National University of Singapore
{r.hoes,eletck}@nus.edu.sg
[2]Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, Netherlands
{a.a.basten,m.c.w.geilen,h.corporaal}@tue.nl

ABSTRACT

Quality of Service (QoS) support for wireless sensor networks (WSN) is a fairly new topic that is gaining more and more interest. This paper introduces a method for configuring the nodes of a WSN such that application-level QoS constraints are met. This is a complex task, since the search space is typically extremely large. The method is based on a recent algebraic approach to Pareto analysis, that we use to reason about QoS trade-offs. It features an algorithm that keeps the working set of possible configurations small, by analysing parts of the network in a hierarchical fashion, and meanwhile discarding configurations that are inferior to other configurations. Furthermore, we give WSN models for two different applications, in which QoS trade-offs are made explicit. Test results show that the models are accurate and that the method is scalable and thus practically usable for WSN, even with large numbers of nodes.

Categories and Subject Descriptors: C.2.4 [Computer-Communication Networks]: Distributed Systems; C.4 [Performance of Systems]: Modelling Techniques

General Terms: Algorithms, Performance, Theory.

1. INTRODUCTION

Wireless sensor networks (WSN) have been suggested for a diversity of applications, such as observing animals in a nature park, assistance in rescue operations, in-home entertainment systems or to monitor people's health. All these applications require different types of sensor nodes and have diverse performance demands. A lot of research has been done on topics such as the physical design of sensor nodes, self-organisation, and communication protocols, with a strong focus on energy efficiency. Recently, researchers are also considering Quality of Service (QoS) provisioning to ensure that explicit performance targets are met. Recent surveys suggest that there is a need for a middleware layer that negotiates between an application and a network to match QoS demands and the availability of WSN resources [2, 14]. This is challenging, because QoS requirements are often conflicting, and furthermore, it needs adequate ways to predict the behaviour and performance of a possibly heterogeneous network of nodes, under various circumstances. This paper proposes a trade-off analysis method. The current work focuses on design-time QoS provisioning, but the method is suitable for a distributed, run-time implementation as well.

A sensor node can typically be configured in many ways, as a node has several settings, such as the transmission power of its radio and its sleep/wake schedule. Moreover, configuring a whole WSN implies setting the parameters for all nodes in the network. Each such configuration has a certain effect on the quality metrics of the application, to which a user may have attached (QoS) constraints. This paper describes a novel method that determines sets of parameters for nodes in a WSN that satisfy or trade off multiple application-level constraints. The method is efficient and scalable to large networks, despite the inherent exponential complexity: for an example on a 100-node network with 8^{100} possible configurations, the algorithm took just 44 seconds to complete, while never more than 4096 configurations needed to be compared at the same time. Our approach uses Pareto-algebraic methods as introduced by Geilen et al. [4], in order to explore trade-offs between system properties, and to incrementally compute a set of feasible configurations. Pareto optimality is used as a central criterion to compare configurations and discard the ones that cannot be optimal, to keep their number manageable.

Related work is reviewed in the next section. To demonstrate that the algebraic approach is practically useful for WSN, Section 3 gives explicit models of sensor nodes and the way they work together to perform target-tracking or spatial-mapping tasks. The method is not limited to these tasks: other tasks are analysable in essentially the same way. Section 4 reviews concepts from Pareto analysis, after which Section 5 introduces the main method for WSN. Section 6 shows test results on networks of varying sizes and the quality of the obtained results.

2. RELATED WORK

Chen et al. [2] give an overview of recent approaches and challenges related to QoS support in WSN. There are some network protocols that offer QoS support [1, 6], often based on delay constraints, but only a few approaches look at application level demands. One example of the latter type attempts to guarantee a minimum reliability level while maximising network lifetime [10]. There is a clear need for a middleware system for WSN that supports application level QoS provisioning [12, 14]. However, this is still a mostly open research problem. Our algebraic framework for handling multiple QoS requirements at the same time is an important step in this direction. This paper focuses on design time trade-off analysis, but the compositional analysis is well suited for distribution, and hence run-time application. To the best of our knowledge, this approach is unique in the area of WSN.

The Pareto-optimality criterion is a general concept that originally comes from economics. The Pareto points of a system precisely capture all the trade-offs in a multi-dimensional optimisation space. In engineering, it is used, for example, in design-space exploration for embedded systems [9, 13]. The recent development of Pareto algebra [4] offers a very structured way of analysing the design space. More traditional ways to find Pareto optimal solutions include genetic algorithms [15] or related algorithms like tabu search. The disadvantages of these approaches are their random nature and the use of a flat search space: they may find some Pareto solutions fairly quickly, but finding all solutions would require an exhaustive search of the whole space. Our algebraic approach intelligently searches the whole space in a hierarchical manner; it is able to quickly find all locally Pareto-optimal solutions at different levels and incrementally combine them into a complete set of solutions, thus providing a better basis for configuring WSN under often conflicting QoS constraints. Q-RAM [7] is another framework that uses the Pareto criterion to find QoS trade-offs, but it does not use algebraic trade-off computation and it focuses on resource allocation for multiple tasks sharing a single resource, which does not directly apply to WSN configuration.

3. WSN MODELS

Before introducing the Pareto method, we define basic models of a WSN for two different tasks (applications), and reveal the inherent trade-offs. These models serve as examples that can be extended or adapted as needed.

3.1 Architectural Overview

Consider a network that consists of a collection $\mathcal{N}$ of identical sensor nodes. The nodes are randomly scattered in an area, and do not move once deployed. The end-user employs the sensor network for a specific task; we look at two different tasks in this example. Firstly, *spatial mapping* (SM), in which all nodes periodically take samples that are sent to the user, for instance to determine the temperature profile over the area. The second task is *target tracking* (TT), in which the objective is to detect and follow target objects. The main difference is that nodes in SM continuously transmit data, while a node in TT only sends a report if it detects a target. We assume that the node locations are known and the network automatically creates a routing tree. The user is in direct contact with the root node of the network.

We use a hierarchical system of requirements and hardware parameters, where the task that runs on the network forms the highest level (the *task level*) and each node is an entity at the lower *node level* (see Figure 1, explained in more detail below, for the TT task). The node level contains a node's hardware settings, such as sample rate and transmission power. From these low-level *parameters* we derive *quality metrics*, like reliability and lifetime. We may use additional metrics to store intermediate results. Quality metrics form the interface between the node and task levels. From the metrics of all nodes together, we derive task-level quality metrics, such as coverage degree and network lifetime. Between the node and task level, we introduce additional *cluster levels*. A cluster is defined as a collection of nodes with the restriction that these nodes form a sub-tree

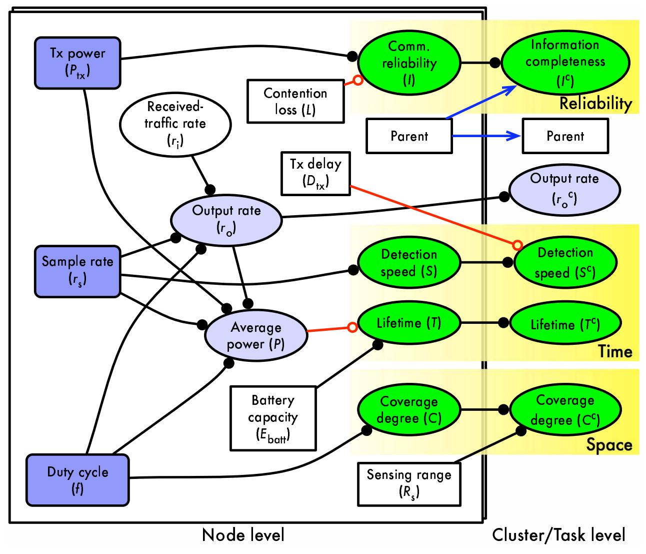


Figure 1: Hierarchical trade-off model: relations between parameters (left), node-level quality metrics and task-level quality metrics (in the shaded boxes).

of the networks routing tree. A cluster's quality metrics are the same as for the task, but they are defined with respect to the cluster's root instead of the network's root. The use of clusters becomes clear in Section 5.

The user is interested in performance at the task level, and may set QoS constraints here, such as a minimum reporting rate in the SM scenario or a maximum delay between the detection of a target and the arrival of the report at the user for TT. Some quality metrics (e.g. network lifetime), may be maximised. The QoS constraints are considered 'soft' (a certain percentage of violations is acceptable), because the unpredictable nature of wireless networks makes it practically impossible to give hard guarantees. The only factors that can be directly set by the system are the hardware settings of individual nodes: the parameters. These are the 'knobs' that should be tuned such that the task-level goals are met. To easily compare different solutions, all quality metrics are defined such that larger values are preferred over lower ones.

3.2 Node-Level Trade-off Models

The left side of Figure 1 gives an overview of parameters (the rounded rectangles) and how they relate to node-level quality metrics (on the right, in the box). The figure only shows the TT task; as explained below, the parameters of the SM task are the same, while the metrics are slightly different. Rectangles represent constant values, while ovals are metrics that depend on parameters. The quality metrics are grouped in three different dimensions: *reliability*, *time* and *space*. Lines with a filled circle at the end represent positive relationships: if the incoming parameter/metric becomes larger, the other also becomes larger. Likewise, lines with an open circle are negative relationships, while lines with an arrowhead are relations that are not clearly positive or negative. The diagram shows that configuring a node means making trade-offs: adjusting a parameter has a positive influence on some quality metrics and a negative influence on others.

The node-level models for the SM and TT tasks are explained below. For every metric we give an example *map-*

ping function in Table 1, which explicitly defines its relation to the parameters. The relations do not necessarily need to be defined analytically. Other ways to obtain mappings for a set of parameter vectors are, for example, simulations or neural networks. Here, we use explicit equations mainly for speed and ease of use. Our simulations described in Section 6 show that the mapping functions are sufficiently precise to accurately predict quality metrics for both tasks.

Table 1: Node-level mappings (F_n) for a node n

Reliability	$I(n) = \left(1 - Q\left(\sqrt{2P_\mathrm{rx}(P_\mathrm{tx}(n))/N}\right)\right)^b (1 - L)$	(1)
Time	$T(n) = \dfrac{E_\mathrm{batt}}{P(n)}$	(2)
	$r(n) = r_\mathrm{s}(n)$ [**SM**]	(3)
	$S(n) = \left(\dfrac{1}{r_\mathrm{s}(n)} + D_\mathrm{s}\right)^{-1}$ [**TT**]	(4)
Space	$C(n) = f(n)$	(5)
Additional metrics	$r_\mathrm{i}(n) = \sum_{i \in ch(n)} r_\mathrm{o}(i)$	(6)
	$r_\mathrm{o}(n) = r_\mathrm{i}(n) + f(n) r_\mathrm{s}(n)$ [**SM**]	(7)
	$r_\mathrm{o}(n) = r_\mathrm{i}(n) + m\dfrac{\pi R_\mathrm{s}^2}{A} f(n) r_\mathrm{s}(n)$ [**TT**]	(8)
	$P(n) = E_\mathrm{s} r_\mathrm{s}(n) f(n) + P_\mathrm{mcu} f(n) + E_\mathrm{tx} r_\mathrm{o}(n) + k E_\mathrm{rx} r_\mathrm{i}(n)$	(9)

The set of children of n is denoted by $ch(n)$.

The first parameter of a node is its sensor's *sample rate* r_s, the number of times per second measurements are taken and processed. The *transmit power* P_tx is the power level at which the node's radio transmits data. Lastly, a node employs a periodic sleep/wake schedule, with fixed period and *duty cycle* f. When the node is in sleep mode, it does not take samples and its micro-controller (MCU) is in low-power mode. We assume the transceiver (including the MAC protocol) does its own power management. Finally, each node has a *parent*, which is the node it sends its reports to according to the routing tree. The metrics are defined below.

Reliability. When a packet is transmitted, another node can receive it with a probability depending on the received SNR and the size of the packet. According to a path-loss model, the received signal power P_rx depends on P_tx and the distance to the receiver d, according to $k_r P_\mathrm{tx}\left(\frac{d_0}{d}\right)^\alpha$, for constant gain factor k_r, path-loss coefficient $\alpha \geq 2$ and reference distance d_0. Moreover, transmissions may interfere with transmissions of other nodes. For simplicity, we assume a constant contention-loss probability L. Then, the *communication reliability* I (the probability of correctly transferring a data packet) assuming a fixed packet size b and BPSK signalling, is expressed as (1), with $Q(x) = \frac{1}{2}\mathrm{erfc}(\frac{x}{\sqrt{2}})$, and noise level N [5].

Time. The node's battery has a limited capacity E_batt. The *lifetime* T of a node, follows from the average power level P via (2). We define the average power P as (9), with constant energy to take and process a sample (E_s), power of the MCU in active mode (P_mcu), and energy to transmit/receive a packet (E_tx and E_rx). We assume the transceiver uses a MAC protocol that minimises idle listening, such as B-MAC [11]. The power level depends on all three parameters, as well as on the additional metric *output traffic rate* r_o, which is the average rate of packet transmissions ((7) for SM, (8) for TT). This rate includes the node's own generated traffic, but also traffic that is to be relayed on behalf of other nodes: the *received traffic rate* r_i. The latter is a special type of metric that depends on the r_o values of the child-nodes in the routing tree (Eq. 6), and hence only indirectly on the parameters, which is why it is visualised differently in Figure 1. Since TT nodes only transmit when a target is in range, r_o does not only depend on the sample rate and duty cycle, but also on the target's trajectory. The fraction of the time that a target is expected to be near is equal to the number of targets m times the fraction of the total area A that is covered by the sensor (with sensing range R_s).

A quality metric unique to the SM task is the *reporting rate* r, a measure of the amount of information generated. It is taken equal to the sample rate in (3). Since TT nodes do not continuously report information, the reporting-rate metric is not used. Instead, we are interested in the *detection speed*. We define the detection delay for a sensor node in the active mode, as the time it takes from the arrival of a nearby target until its detection. This delay depends on r_s, and (combined in the constant D_s) the duration of sampling and detection. The (worst-case) detection speed S, given by (4), is the inverse of the detection delay.

Space. A sensor is said to 'cover' the area within its sensing range when it is active. Since a sensor node is typically asleep most of the time, it does not continuously cover this area. We therefore define the metric *coverage degree* C in (5) as the fraction of the time the sensor is switched on.

3.3 Task- and Cluster-Level Trade-off Models

The mapping functions for the task- and cluster-level metrics, as shown in Table 2, are explained below. The functions in the table are for a cluster c. Task-level mapping functions are obtained by substituting the network $\mathcal{N}$ for c.

Reliability. In both scenarios, nodes send reports to the user. However, because the communication of reports usually has a limited reliability, not all reports may reach the destination. We want to know how complete the data is that is received by the user: the *information completeness* I^c is a measure of the fraction of all generated reports that arrive. This is approximately equal to the average end-to-end communication reliability over all nodes, given by (10), where $\bar{p}^i$ is the path from node i to the root node.

Time. For an SM task, we are interested in the *reporting rate* of the network/cluster, which is defined by (11) by the average reporting rate over all nodes. A common timeliness metric of a WSN that does target detection is the time it takes from the appearance of a target until the detection report reaches the user. For each node, this delay depends on its detection speed and the hop count $|\bar{p}^i|$ to the root node. The worst-case *detection speed* S^c is given by (12), where D_tx is the transmission delay (including MAC delay). Further, we use the definition (13) for the *lifetime* T^c that considers all nodes in the network/cluster as essential.

Space. The area that is covered by the nodes, if all nodes would be active, is called the covered area. However, a sensor node only covers the area in its range for a fraction of

Table 2: Cluster-level mappings (G_{nc}) for a cluster c

Reliability	$I^{\mathrm{c}}(c) = \dfrac{\sum_{i \in c} \prod_{j \text{ on } \bar{p}^i} I(j)}{\lvert c \rvert}$	(10)
Time	$r^{\mathrm{c}}(c) = \dfrac{\sum_{i \in c} r(i)}{\lvert c \rvert}$ **[SM]**	(11)
	$S^{\mathrm{c}}(c) = \min\limits_{i \in c} \left\{ \left(\dfrac{1}{S(i)} + \lvert \bar{p}^i \rvert D_{\mathrm{tx}} \right)^{-1} \right\}$ **[TT]**	(12)
	$T^{\mathrm{c}}(c) = \min\limits_{i \in c} \{T(i)\}$	(13)
Space	$C^{\mathrm{c}}(c) = \min\limits_{i \in c} \{C(i)\}$	(14)
Additional metrics	$r_{\mathrm{o}}^{\mathrm{c}}(c) = r_{\mathrm{o}}(rt(c))$	(15)
	$\mathit{parent}^{\mathrm{c}}(c) = \mathit{parent}(rt(c))$	(16)

The root node of cluster c is denoted by $rt(c)$.

the time. We therefore introduce the metric *coverage degree* C^{c}. For a point in the covered area, C^{c} is defined as the percentage of the time that it is covered by *at least one* sensor, during a certain period. The coverage degree for the whole covered area is the minimum coverage degree over the whole area. To calculate C^{c} for the network/cluster, we need the locations and sensing ranges, plus the coverage degrees of all the nodes. We could approximate the target area by a grid of points and take the minimum coverage degree for each point. For this example, however, we use the form of (14), which is accurate if every sensor covers some area that cannot be covered by any other sensor.

Finally, a cluster's output traffic and parent node are defined as the output traffic and parent node of its root node.

4. THE PARETO-ALGEBRA APPROACH

Our method to configure a WSN is rooted in the algebraic approach to Pareto analysis introduced by Geilen et al. [4]. Pareto analysis is a way to analyse trade-offs in a system in order to find optimal combinations of parameters and quality metrics, called *Pareto points*. We survey some essential concepts from [4].

4.1 Configurations and Minimisation

Consider a general system with parameters and quality metrics. Each of the parameters or metrics can hold values in a specific range or domain that is determined by the characteristics of the hardware and its environment. Such a domain is called a *quantity*, which is a discrete set Q of values, with a partial order $\preceq_Q$ (if the quantity is clear from the context, we simply write $\preceq$). If $q_1, q_2 \in Q$, then $q_1 \preceq_Q q_2$ means that the value q_1 is considered better than q_2. The ordering of a quantity allows to express a preference of certain values over others. For example, for a WSN, we can have $\mathit{CovDegree} = \{0.3, 0.5, 0.8\}$ with $0.8 \preceq 0.5 \preceq 0.3$ ($\preceq$ equals $\geq$ for greater-is-better).

A configuration space $\mathcal{S}$ is the Cartesian product $Q_1 \times \ldots \times Q_n$ of a finite number of quantities, and a configuration $\bar{c} = (c_1, \ldots, c_n)$ is an element of such a configuration space. The configuration space holds all possible configurations of a system, given a set of quantities. Since the space can be very large, it is desirable to select only potentially useful configurations for further analysis, instead of analysing all possibilities. Pareto analysis is able to make such a selection, given the preferences expressed in the ordering of quantities.

A dominance relation is used to find configurations that are clearly worse than others and do not have to be considered any further. For $\bar{c}_1, \bar{c}_2 \in \mathcal{S}$, configuration $\bar{c}_1$ is said to *dominate* $\bar{c}_2$, denoted by $\bar{c}_1 \preceq \bar{c}_2$, if and only if for every quantity Q_k of $\mathcal{S}$, $\bar{c}_1(Q_k) \preceq_{Q_k} \bar{c}_2(Q_k)$. Dominance is a partial order and hence a reflexive relation: every configuration dominates itself. The irreflexive variant, *strict dominance*, is denoted $\prec$. Configuration $\bar{c}_1$ dominates an other configuration $\bar{c}_2$, when it is better in at least one quantity and not worse in any of the other quantities. For example, if we have $\mathcal{S} = \mathit{CovDegree} \times \mathit{Lifetime}$, with $\mathit{Lifetime} = \mathbb{N}^+$ and $\preceq = \geq$, then $(0.8, 100) \preceq (0.5, 100)$, which means that we do not have to consider the second configuration. However, $(0.8, 100) \not\preceq (0.5, 200)$ and also $(0.5, 200) \not\preceq (0.8, 100)$, implying none of the two is clearly better.

Definition 1 (Pareto Minimal). *A set $\mathcal{C}$ of configurations is* Pareto minimal *iff for any $\bar{c}_1, \bar{c}_2 \in \mathcal{C}$, $\bar{c}_1 \not\prec \bar{c}_2$.*

We denote the Pareto-minimal set of an arbitrary configuration set $\mathcal{C}$ by $\min(\mathcal{C})$ and call the process of computing it *minimisation*. For every configuration in $\mathcal{C}$, there is an element of $\min(\mathcal{C})$ that dominates it. The selected configurations are called *Pareto (optimal) configurations* or *Pareto points*. The Pareto-minimal set is unique for finite sets of configurations. Hence, when using a finite configuration set $\mathcal{C}$, we only need to consider the subset $\min(\mathcal{C})$ and we can ignore all the other configurations.

4.2 Building Sets of Configurations

A system often has metrics that depend on other metrics: high-level metrics could be derived from lower-level metrics, while these lower-level metrics themselves may depend on parameters. For a configuration space $\mathcal{S}$, we can define a function $f : \mathcal{S} \to Q$, where the new quantity Q is called a *derived quantity*. In this work, we call f a *mapping function*. We can extend a configuration set $\mathcal{C}$ using f, to create $\mathcal{C}_f = \{\bar{c} \cdot f(\bar{c}) \mid \bar{c} \in \mathcal{C}\}$, where the dot $(\cdot)$ denotes concatenation. However, for Pareto algebra to be useful, we need to impose an extra restriction on mapping functions. Suppose we have two configurations $\bar{c}_1, \bar{c}_2 \in \mathcal{C}$, with $\bar{c}_1 \preceq \bar{c}_2$ and $f(\bar{c}_1) \not\preceq f(\bar{c}_2)$. This would mean that configurations that are not part of $\min(\mathcal{C})$, could be part of $\min(\mathcal{C}_f)$. This is undesirable, because when minimising before adding the new quantity (the key idea of Pareto algebra is the ability to minimise at intermediate steps of the analysis), potentially optimal configurations may get lost. Therefore, mapping functions that are applied after minimisation should be *monotone*.

Definition 2 (Monotonicity). *A function $f : \mathcal{S} \to Q$ is* monotone *iff for any $\bar{c}_1, \bar{c}_2 \in \mathcal{S}$ such that $\bar{c}_1 \preceq_{\mathcal{S}} \bar{c}_2$, $f(\bar{c}_1) \preceq_Q f(\bar{c}_2)$.*

This is the generic definition of monotonicity for partial orders. A function h on real numbers (where $\preceq$ equals $\geq$), for example, is monotone if $x \geq y$ implies $h(x) \geq h(y)$ for all $x, y \in \mathbb{R}$ (h is a non-decreasing function).

A configuration set can be constructed by adding derived quantities, but we can also combine two configuration sets from different spaces. For example, the configuration sets of two sensor nodes may be combined into one joint configuration set. This is done by the *free product* operation. The free product of configuration sets $\mathcal{C}_1$ and

$\mathcal{C}_2$ is the Cartesian product $\mathcal{C}_1 \times \mathcal{C}_2$. If $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively contain n and m configurations, then $\mathcal{C}_1 \times \mathcal{C}_2$ contains nm configurations. The free product preserves minimality: $\min(\mathcal{C}_1 \times \mathcal{C}_2) = \min(\mathcal{C}_1) \times \min(\mathcal{C}_2)$.

4.3 Constraints and the Final Decision

When a configuration set of the whole system has been constructed, one can remove configurations that do not meet the QoS constraints. The result is a set of achievable quality-metric values plus the parameters to attain these values. If this set is empty, the user's requirements cannot be met by any combination of the given sets of parameters. If there are multiple solutions, a choice between them should be made on other grounds, for example by prioritising some metrics.

5. CONFIGURING A WSN

Consider a network as a set of nodes denoted by their indices, $\mathcal{N} = \{0, 1, \ldots, |\mathcal{N}| - 1\}$. As mentioned in Section 3, the network is organised as a tree, where the root of the tree receives reports from all nodes and delivers these to the user. Each node parameter is associated with a quantity that contains all its possible values. For example, transmit power could have a quantity $\mathit{TxPower} = \{0, 5\}$ (in dBm). The *parameter space* for a sensor node, denoted $\mathcal{S}_{P,\mathrm{n}}$, is the Cartesian product of all parameter quantities. In our model, we have $\mathcal{S}_{P,\mathrm{n}} = \mathit{TxPower} \times \mathit{SampleRate} \times \mathit{DutyCycle}$. The parameter space for the whole task $\mathcal{S}_{P,\mathrm{t}}$ is equal to $(\mathcal{S}_{P,\mathrm{n}})^{|\mathcal{N}|}$.

The end-user that is running a task on the WSN is interested in the task-level behaviour of the network, expressed in a collection of quality metrics. Each metric is also associated with a quantity. The *quality space* $\mathcal{S}_M$ for the SM task, for example, is $\mathit{InfoCompleteness} \times \mathit{ReportingRate} \times \mathit{Lifetime} \times \mathit{CovDegree}$. We can see a WSN as a system that transforms a vector of parameters to a vector of quality metrics. A *configuration* is a collection of parameter values together with the collection of resulting quality-metric values.

Definition 3 (Mapping). *A mapping $F : \mathcal{S}_P \to \mathcal{S}_M$, for a parameter space $\mathcal{S}_P$ and quality space $\mathcal{S}_M$, derives a vector of quality metrics from a vector of parameters (like derived quantities, Section 4). More precisely, F is a tuple of* mapping functions *$f_i : \mathcal{S}_P \to Q_i$, one for each quality metric i: $F = (f_0, f_1, \ldots, f_{k-1})$, with k the number of quality metrics. Likewise, an* incremental *mapping $G : \mathcal{S}_{M,1} \to \mathcal{S}_{M,2}$ derives a vector of quality metrics from another vector of (possibly different) metrics. The functions F and G can be lifted to sets: $F(\mathcal{C}) = \{F(\bar{c}) \mid \bar{c} \in \mathcal{C}\}$ and similar for G.*

To find the set of Pareto optimal configurations using a flat system model, we would have to map *every* parameter vector in $\mathcal{S}_{P,\mathrm{t}}$ to a vector of quality metrics by means of a mapping F_t[1], and Pareto minimise the resulting set of metrics. F_t can be defined by combining the functions in Tables 1 and 2. However, the size of $\mathcal{S}_{P,\mathrm{t}}$ increases exponentially with the number of nodes in the network, so such an approach would not be scalable. A solution is to exploit hierarchy in the system, by building and minimising sets of configurations step by step: start with individual nodes, and then incrementally combine nodes into clusters (see Section 3). We expect that minimisation discards a significant number of dominated configurations in each step, such that only the resulting (Pareto) configurations need to be forwarded to the next level. The correctness of this approach – the resulting set of configurations from the one-step and clustered approaches should be identical – is investigated in Section 5.1–5.3. The algorithm's complexity and distributed implementation are discussed in Section 5.4 and 5.5.

[1]For a mapping F, we use subscripts 'n', 'c' and 't' for a mapping to node, cluster (Section 5.1) and task level respectively. For G, we use a two-letter subscript, to indicate the source and destination level.

Algorithm 1: Combining clusters incrementally
```
create initial one-node clusters
repeat until a single cluster remains:
    monotonically combine $\geq 2$ clusters
    derive quality metrics of new cluster
    minimise new cluster's configuration set
```

5.1 The Cluster Method

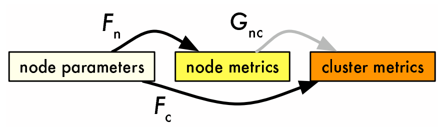
After mapping the parameter spaces to node-level quality metrics by functions from Table 1, and minimising the resulting sets, we obtain Pareto-optimal subsets of the configuration space $\mathcal{S}_{M,\mathrm{n}}$ for each node in the network. A cluster is a subset of $\mathcal{N}$, and the parameters of its containing nodes become the cluster's parameters. Therefore, an n-node cluster i has parameter space $\mathcal{S}_{P,i} = (\mathcal{S}_{P,\mathrm{n}})^n$. A cluster's quality metrics are derived from its parameters by means of the mapping $F_\mathrm{c} : (\mathcal{S}_{P,\mathrm{n}})^n \to \mathcal{S}_{M,\mathrm{c}}$ (see Figure 2(a)). The cluster-level quality metrics are the same as the task-level quality metrics, only the number of nodes could be different. Thus, if we can compute the quality metrics of a cluster containing all nodes, we immediately obtain the quality metrics for the task. Note that F_c can be implemented by deriving cluster metrics from node metrics by mapping G_nc of Table 2; see the grey arrow in Figure 2(a). In the analysis below, we use F_c to keep the explanation clear and general.

Clusters can be combined into a larger cluster by constructing the free-product of the parameter spaces of those clusters, and deriving new cluster quality metrics with F_c (see Figure 2(b)). However, it is usually more efficient to *incrementally* derive new metrics from the cluster metrics of the clusters that are combined instead. We show below that this is possible for our WSN models, and in general for all practically useful cases.

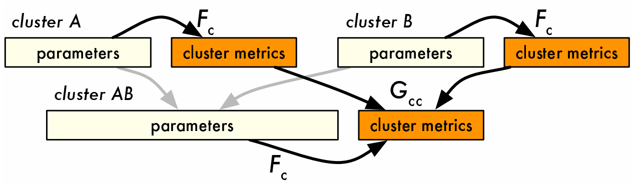
The basic structure of the algorithm is given as Algorithm 1. First, each node is converted into a one-node cluster (using F_c). In each loop iteration, multiple clusters are combined into one large cluster. Line 3 combines two or more clusters (selected under conditions defined below) by taking the free product of the configuration sets of these clusters. Then, the set of quality-metric vectors is derived and minimised (lines 4–5). The loop stops when one cluster remains that contains all nodes in the network. The result is a set of Pareto-optimal task-level configurations.

However, it is generally not possible to combine any arbitrary set of clusters. Firstly, all mapping functions to cluster quality metrics need to be defined for the combined cluster. Some mapping functions in our model, for instance, are defined with respect to the routing tree. Therefore, each cluster that uses this function needs to be a tree: a sub-tree of the network's routing tree.

Secondly, we need to check for *monotonicity*. We need to make sure that the configurations that are removed by minimisation in earlier clustering steps, could never become optimal in later steps. This concept is related to monotonicity as defined in Definition 2.



(a) Cluster-level quality metrics can be derived from the parameters of the nodes in the cluster by a mapping F_c. Alternatively, they can be derived from node-level quality metrics (G_{nc}).



(b) Clusters A and B are combined into cluster AB. The cluster quality metrics of AB can always be directly derived from the parameters of the nodes in the cluster (F_c). If they can also be derived from the cluster-level metrics of A and B and this mapping G_{cc} is monotone, the combining action is also monotone.

Figure 2: Cluster-level mappings.

Definition 4 (Monotonicity of a clustering step). *Suppose we are combining clusters 0 to $\ell - 1$ with parameter sets $\mathcal{C}_{P,i} \subseteq \mathcal{S}_{P,i}$ for all clusters $0 \leq i < \ell$. The action of combining these clusters is monotone, iff for all $\bar{c}_1^i, \bar{c}_2^i \in \mathcal{C}_{P,i}$ and $0 \leq i < \ell$, $F_c(\bar{c}_1^i) \preceq F_c(\bar{c}_2^i)$ implies $F_c(\bar{c}_1^0 \cdot \ldots \cdot \bar{c}_1^{\ell-1}) \preceq F_c(\bar{c}_2^0 \cdot \ldots \cdot \bar{c}_2^{\ell-1})$.*

A monotone clustering step preserves the dominance order of cluster configurations. This implies that a cluster configuration $\bar{c}$ that is dominated before the clustering step, can safely be removed by minimisation, because all configurations of the combined cluster that incorporate $\bar{c}$ would be dominated by other configurations after clustering. In other words, *if all clustering steps are monotone, none of the eventual task-level Pareto configurations are lost*, and the result from the clustered algorithm would be the same as the result from an all-at-once algorithm. The following lemma is illustrated in Figure 2(b).

Lemma 1. *A clustering step in which clusters 0 to $\ell - 1$ are combined is monotone, if the function F_c to calculate the quality metrics of the combined cluster can be rewritten as a monotone function of only the values of the cluster-level quality metrics of clusters that are combined (metrics of individual nodes are not explicitly needed). This means that F_c can be rewritten as*

$$F_c\left(\bar{c}^0 \cdot \ldots \cdot \bar{c}^{\ell-1}\right) = G_{cc}\left(F_c(\bar{c}^0), \ldots, F_c(\bar{c}^{\ell-1})\right) \qquad (17)$$

for some monotone ℓ-ary function $G_{cc} : (\mathcal{S}_{M,c})^\ell \rightarrow \mathcal{S}_{M,c}$. The function G_{cc} is a tuple $(g_0, g_1, \ldots, g_{k-1})$ of monotone functions that incrementally *compute new cluster-level quality metrics (k is the number of quality metrics).*

PROOF. Immediately from the monotonicity of G_{cc}. □

This result implies that new cluster-level quality metrics can be derived from clusters that are monotonically combined in several ways: by the function F_c from the parameters of all *nodes* inside the new cluster, or by the function G_{cc} from the quality metrics of all the *clusters*, or a hybrid form. This is an implementation choice that can be made per mapping function, based on efficiency of computation and storage. Usually, the second option would be the most efficient. Before providing G_{cc} for the models of Section 3, we first illustrate the method by means of an example.

5.2 The Method Applied

Consider a mapping function f_d to calculate a maximum-delay metric (captured in quantity Q_d) for a cluster, given a parameter vector $\bar{c}$. Note that the WSN model for target tracking has a speed metric instead of a delay metric. For the sake of the example, however, we use the delay, for which lower values are preferred ($\preceq_{Q_d}$ equals $\leq$). We express f_d in terms of the values T_i, the time to send a message from node i to its parent in the routing tree.

$$f_d(\bar{c}) = \max_{p \in L}\left\{\sum_{i \text{ on } p} T_i\right\}, \qquad (18)$$

where L is the set of paths from all leaf nodes in the cluster to the root node (including the root node itself). Clearly, this function does not depend only on the T_i values of the nodes in the cluster; also the way the nodes are connected in the routing tree plays an important role. Therefore, when combining clusters with maximum delay as a quality metric, it is necessary that these clusters are connected by links in the network's routing tree, such that the new cluster is a tree. Furthermore, we need to make sure that the clustering step is monotone by ensuring that (17) holds for f_d. We first give an example of a clustering strategy in which the monotonicity condition fails, and then we show a clustering strategy that is monotone. Throughout the example, we only show the delay metric, but keep in mind that there may be other quality metrics as well. This means that even if a configuration has a worse delay than another, it may or may not be a Pareto configuration, depending on the values of other metrics. We write a configuration $\bar{c}$ as a vector of only T_i values.

Suppose we have a cluster $\{1, 2, 3\}$, as in Figure 3(a). The cluster has multiple possible configurations (different values of T_1, T_2 and T_3). The figure shows two configurations: $\bar{c}_1^{\{1,2,3\}} = (1, 4, 1)$ and $\bar{c}_2^{\{1,2,3\}} = (1, 1, 3)$. The cluster delays are $f_d(\bar{c}_1^{\{1,2,3\}}) = \max\{1 + 4, 1 + 1\} = 5$ and $f_d(\bar{c}_2^{\{1,2,3\}}) = \max\{1 + 1, 1 + 3\} = 4$ (the thick arrows in the figure show the bottleneck path). Thus, $f_d(\bar{c}_2^{\{1,2,3\}}) \preceq_{Q_d} f_d(\bar{c}_1^{\{1,2,3\}})$ and after minimisation, $\bar{c}_1^{\{1,2,3\}}$ may be eliminated (depending on the other metrics). Now we join this cluster with downstream cluster $\{5\}$, with one configuration $\bar{c}^{\{5\}} = (2)$. The new configurations are $\bar{c}_1^{\{1,2,3\}} \cdot \bar{c}^{\{5\}} = (1, 4, 1, 2)$ and $\bar{c}_2^{\{1,2,3\}} \cdot \bar{c}^{\{5\}} = (1, 1, 3, 2)$. The delays become $f_d(\bar{c}_1^{\{1,2,3\}} \cdot \bar{c}^{\{5\}}) = \max\{1 + 4, 1 + 1 + 2\} = 5$ and $f_d(\bar{c}_2^{\{1,2,3\}} \cdot \bar{c}^{\{5\}}) = \max\{1 + 1, 1 + 3 + 2\} = 6$. This implies that $f_d(\bar{c}_1^{\{1,2,3\}} \cdot \bar{c}^{\{5\}}) \preceq_{Q_d} f_d(\bar{c}_2^{\{1,2,3\}} \cdot \bar{c}^{\{5\}})$, but $\bar{c}_1^{\{1,2,3\}}$ may have been discarded in the previous step! This implies that this clustering step is non-monotone. The reason is that the addition of a *downstream* cluster extends the bottleneck path in one configuration but not in the other one. Observe that this cannot occur when adding upstream clusters; the

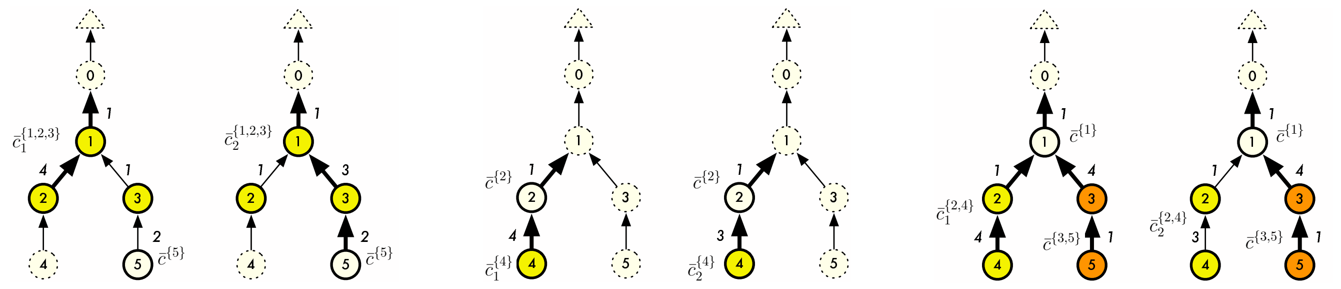


Figure 3: Examples of non-monotone (a) and monotone clustering steps (b)–(c). A number at the arc coming out of node i is the delay T_i.

bottleneck path is then always affected.

Algorithm 1 prescribes that the procedure starts by initialising all nodes as one-node clusters. After this, clusters can be combined step by step, but we need to ensure, firstly, that the new cluster contains a tree (otherwise f_d is undefined), and secondly, that the clustering step is monotone. Consequently, a situation as in Figure 3(a) should not occur. For clusters with a maximum-delay metric as defined above, we can combine clusters with a special restriction: *if a cluster is selected for combination, at least all clusters that contain nodes lower in the routing tree (all the way to the leaf nodes) need to be combined in the same step as well.* We can thus include this condition in line 3 of Algorithm 1.

Figure 3(b) shows a possible first clustering step, after initialisation. Here one-node clusters $\{2\}$ and $\{4\}$ are combined. The figure shows two configurations $\bar{c}_1^{\{4\}} = (4)$ and $\bar{c}_2^{\{4\}} = (3)$ of cluster $\{4\}$, and one configuration $\bar{c}^{\{2\}} = (1)$ of $\{2\}$. The mapping function for the combined cluster $\{2, 4\}$ is simply $T_4 + T_2$, which is 5 and 4 for configurations $\bar{c}_1^{\{4\}} \cdot \bar{c}^{\{2\}}$ and $\bar{c}_2^{\{4\}} \cdot \bar{c}^{\{2\}}$ respectively. This step is clearly monotone, and this is always the case when combining clusters that contain only one path: the max-function can be left out and the remaining summation is always monotone.

The second clustering step in the example would be the combination of $\{3\}$ and $\{5\}$, which goes in the same way as step 1. A possible third step is given in Figure 3(c). In this step, we are joining cluster $\{2, 4\}$ (having configurations $\bar{c}_1^{\{2,4\}} = (1, 4)$ and $\bar{c}_2^{\{2,4\}} = (1, 3)$) with $\{3, 5\}$ (configuration (4,1)) and $\{1\}$ (configuration (1)). Configuration $\bar{c}_1^{\{2,4\}}$ has a longer delay $(1+4 = 5)$ than $\bar{c}_2^{\{2,4\}}$ $(1+3 = 4)$, so it could have been discarded in step 1. Therefore, it should not be possible that a combination of this configuration with any of the other clusters' configurations becomes a unique Pareto point. The joint cluster's delay $f_\text{d}(\bar{c}^{\{2,4\}} \cdot \bar{c}^{\{3,5\}} \cdot \bar{c}^{\{1\}})$ is

$$\begin{aligned}
&\max\{T_4 + T_2 + T_1,\ T_5 + T_3 + T_1\} = \\
&\max\left\{f_\text{d}(\bar{c}^{\{2,4\}}) + f_\text{d}(\bar{c}^{\{1\}}),\ f_\text{d}(\bar{c}^{\{3,5\}}) + f_\text{d}(\bar{c}^{\{1\}})\right\} = \\
&\max\left\{f_\text{d}(\bar{c}^{\{2,4\}}),\ f_\text{d}(\bar{c}^{\{3,5\}})\right\} + f_\text{d}(\bar{c}^{\{1\}}). \qquad (19)
\end{aligned}$$

The last line can be seen as a function $g(x, y, z) = \max\{x, y\} + z$, which is monotone, and therefore this clustering step is monotone according to Lemma 1. In the example, using $\bar{c}_1^{\{2,4\}}$ or $\bar{c}_2^{\{2,4\}}$ will lead to a combined-cluster delay of $1+4+1 = 6$ or $4+1+4 = 6$ respectively. Although $\bar{c}_1^{\{1,2,3,4,5\}}$ is not strictly worse than $\bar{c}_2^{\{1,2,3,4,5\}}$, it is still dominated (equal values dominate each other), so $\bar{c}_1^{\{2,4\}}$ could have been safely removed. Configuration $\bar{c}_1^{\{1,2,3,4,5\}}$ may be worse in dimensions other than Q_d to render it strictly dominated, or the two configurations may be identical in terms of quality metrics, in which case only one needs to be kept. But $\bar{c}_1^{\{1,2,3,4,5\}}$ will never strictly dominate $\bar{c}_2^{\{1,2,3,4,5\}}$.

We also see in (19) how the delay mapping function for step 3 can be rewritten from a function that operates on node-level metrics to a function that uses cluster-level metrics. The implementation of the latter is clearly the most efficient, since the computation is easier and fewer values need to be stored.

The final step to complete the network is to combine cluster $\{1, 2, 3, 4, 5\}$ with cluster $\{0\}$, with $T_0 = 2$. This step is straightforward: the delay is equal to $f_\text{d}\left(\bar{c}^{\{1,2,3,4,5\}} \cdot \bar{c}^{\{0\}}\right) = f_\text{d}\left(\bar{c}^{\{1,2,3,4,5\}}\right) + f_\text{d}\left(\bar{c}^{\{0\}}\right)$ and this step is thus monotone. The delays are $6 + 2 = 8$ for both configurations.

Proposition 2 (Monotonocity of Alg. 1 for f_d). *All clustering steps of Algorithm 1 are monotone for f_d, if in each clustering step, a cluster that is selected for combination is combined with at least all clusters that contain nodes lower in the tree (down to the leaf nodes).*

Proof. For each step of the algorithm, we need to ensure that f_d is defined for the combined cluster, and that the step is monotone. The algorithm maintains two invariants:

1. Each cluster is a tree. This ensures that f_d is defined for each cluster.

2. A cluster either contains exactly one node, or it contains all nodes that are lower in the network's routing tree than the cluster's root node (it is a *leaf cluster*).

Line 1 initialises both invariants, while line 3 plus the selection condition trivially maintains them, whichever cluster is chosen for combination. Invariant 2 ensures that, when combining clusters, the root of a multiple-node cluster M is always connected to the root R of the new cluster via a path containing only one-node clusters. If not, a node belonging to another multiple-node cluster would be on this path, but this implies the existence of a multiple-node cluster that is not a leaf cluster, which contradicts invariant 2. The maximum delay from any leaf node in cluster M to R is equal to the maximum delay of cluster M as a whole, plus the delays of the extra nodes on the path (including R). This is a summation of only cluster quality metrics, which is monotone.

Algorithm 2: Cluster combining for WSN model

```
1  function CreateCluster(root):
2      if root is leaf node:
3          set received traffic $r_{\mathrm{i}} = 0$
4          return one-node cluster for root
5      for each child call CreateCluster(child)
6      determine received traffic $r_{\mathrm{i}}$ for root
7      create one-node cluster for root
8      combine root and child clusters
9      derive quality metrics of new cluster
10     minimise configuration set
11     return configuration set
```

Table 3: Incremental mappings (G_{cc}) for a cluster c

Reliability

$$I^{\mathrm{c}}_{\Sigma}(c) = I^{\mathrm{c}}_{\Sigma}(rt(c)) \cdot \left(1 + \sum_{i \in ch(c)} I^{\mathrm{c}}_{\Sigma}(i)\right) \tag{20}$$

Time

$$r^{\mathrm{c}}_{\Sigma}(c) = \sum_{i \in sub(c)} r^{\mathrm{c}}_{\Sigma}(i) \quad [\mathbf{SM}] \tag{21}$$

$$S^{\mathrm{c}}(c) = \min\left\{S^{\mathrm{c}}(rt(c)), \min_{i \in ch(c)}\left\{\frac{1}{S^{\mathrm{c}}(i)} + D_{\mathrm{tx}}\right\}\right\} \quad [\mathbf{TT}] \tag{22}$$

$$T^{\mathrm{c}}(c) = \min_{i \in sub(c)} \{T^{\mathrm{c}}(i)\} \tag{23}$$

Space

$$C^{\mathrm{c}}(c) = \min_{i \in sub(c)} \{C^{\mathrm{c}}(i)\} \tag{24}$$

Additional metrics

$$r^{\mathrm{c}}_{\mathrm{o}}(c) = r^{\mathrm{c}}_{\mathrm{o}}(rt(c)) \tag{25}$$

$$parent^{\mathrm{c}}(c) = parent^{\mathrm{c}}(rt(c)) \tag{26}$$

Note: (20), (22), (25) *and* (26) *depend on a tree; the others do not. For combined cluster c, the root cluster is denoted $rt(c)$, the set of child clusters $ch(c)$; $sub(c) = \{rt(c)\} \cup ch(c)$.*

Thus, function f_{d} applied to a combination of one-node and multiple-node clusters is equivalent to the maximum over the maximum delays for all leaf clusters (either multiple-node or one-node) to R. This is a monotone function using only metrics of the combined clusters, which ensures that the clustering step itself is monotone for the delay metric (Lemma 1). □

5.3 Correctness for the WSN Models

To apply the cluster method to the WSN models of Section 3, we need to slightly reorganise the algorithm. The node model contains the metric received-traffic rate, which depends on the output-traffic rates of nodes lower in the tree. Since the algorithm works from leaves to root, and it is most efficient to climb up the tree node by node (see Section 5.4), we can interleave the computation of received rate – and thus the nodes' initialisation – with the clustering steps, instead of computing all node-level configurations beforehand. Algorithm 2 is the adapted algorithm, written in a recursive form that should be called with the network's root node as argument. To prove the correctness of the cluster method for the WSN model, we need to provide a monotone mapping G_{cc} (Lemma 1). Monotonicity should be verified for each mapping function g_i separately, and a clustering step should only combine clusters that can be monotonically combined (note that functions F need *not* be monotone). Table 3 gives G_{cc}. I^{c}_{Σ} and r^{c}_{Σ} are cumulative metrics that should be divided by $|c|$ to get the actual quality metrics I^{c} and r^{c} of Table 2. The incremental computation of these cumulative metrics is more efficient than the computation of the actual metrics.

Theorem 3 (Monotonocity of Algorithm 2). *The cluster method (Alg. 2) is monotone for the WSN model.*

Proof. It can be shown by similar arguments as in Proposition 2 (plus the fact that minimum, addition and multiplication are monotone) that, given the clustering strategy in Algorithm 2, all mapping functions in the model can be written as monotone functions from cluster to cluster metrics as in Table 3. Therefore, by Lemma 1, Algorithm 2 is monotone for the WSN model. □

5.4 Complexity of the Cluster Method

The operations of Pareto algebra mostly have a polynomial time complexity which is at most quadratic (a basic minimisation algorithm) in the number of configurations n. A crucial operation is combining two sets of configurations of size n and m by a free product, which has complexity $\mathcal{O}(n \cdot m)$, and increases the number of configurations from $n + m$ to $n \cdot m$. The free product increases the number of configurations, while minimisation and applying constraints usually reduce this number.

The efficiency of Algs. 1 and 2 mainly depends on the number of clusters ℓ that are combined per step, and the number of configurations $|\mathcal{C}_i|$ in each cluster i. Line 3 of Algorithm 1 (line 8 of Algorithm 2) combines configuration sets with a free product. The size of the resulting set $\mathcal{C}_{\mathrm{prod}}$ is equal to $|\mathcal{C}_0| \cdot \ldots \cdot |\mathcal{C}_{\ell-1}|$, and the time complexity of the free-product operation is $\mathcal{O}(|\mathcal{C}_{\mathrm{prod}}|)$. The complexity of the derivation step in the following line also depends on $|\mathcal{C}_{\mathrm{prod}}|$ (metrics need to be derived for each new configuration), but on the complexity of the mapping functions as well. Finally, the complexity of the minimisation operation also depends on $|\mathcal{C}_{\mathrm{prod}}|$, as said above.

If we consider the number of configurations per node as given and at most n, mapping functions can be evaluated in constant time, and assuming a quadratic minimisation algorithm is used, the complexity of joining all nodes in one step is $\mathcal{O}(n^{2|\mathcal{N}|})$. This is obviously not scalable and therefore not useful for WSN in general. Therefore, it makes sense to join as few clusters as possible in each step and to rely on minimisation to keep the configuration sets small. The algorithm's complexity could even become linear, if each step is able to reduce the set size to roughly m when cluster configuration sets of size m are combined. On the other hand, if the minimisation operation does not manage to significantly reduce the size of $\mathcal{C}_{\mathrm{prod}}$, the complexity would again be exponential. This number of configurations that can be minimised away greatly depends on the mapping functions and the configurations' values, and it is hard to give general bounds. However, from experiments we see that in practical cases, the algorithm does behave about linearly in the number of nodes (see Section 6).

An extra constraint comes from the fact that clusters can only be combined if this action is monotone. In case we have mapping functions that are defined with respect to a routing tree, like the delay function discussed earlier, and we use Algorithm 2, the number of clusters that we need to combine in one step depends on the number of immediate child nodes of the parent cluster. Therefore, the complexity of the algorithm would also depend on the node degree in this case: the algorithm performs best for routing tree with a low node degree. Since for WSN a low node degree is generally preferred, because of the better distribution of traffic, this is not a severe limitation.

5.5 Distributed Execution

The algorithm as described in this section is given as a centralised algorithm that is run separately from the WSN, before starting the WSN's task. However, Algorithm 2 treats nodes in a leaf-to-root fashion; a node only depends on information from downstream nodes to compute configurations for the whole cluster with itself as root. It is therefore also possible to execute the algorithm in a distributed way, in which each node passes the optimal configurations on to its parent, after computing the Pareto set for its cluster. When the network's root is reached and a final task-level configuration has been chosen, this configuration is communicated back down the tree.

6. EXPERIMENTAL RESULTS

We implemented Algorithm 2 and ran it for networks of different sizes, for both the SM and TT models (on a PC with a 1.86GHz Core 2 Duo processor). To gain insight in the scalability of the approach, we recorded the number of configurations at each step after the clusters were combined ($|\mathcal{C}_{\mathrm{prod}}|$), as well as the run time. Constraints were not used in these experiments; the goal was to find *all* Pareto points.

For each network size, we randomly distributed sensor nodes in a square area. To ensure an even distribution across the area, we placed the nodes with a certain variance around fixed grid points. While scaling the number of nodes, the area was scaled accordingly, such that the node density was equal for all networks. For each network, a shortest-path spanning tree (SPST) was created. To ensure a fair comparison between the results for all networks, we only used SPSTs in which each node has at most 4 immediate child nodes (algorithm complexity depends on node degree, see Section 5.4). We first set the quantities for the node-level parameters as follows: $\mathit{TxPower} = \{0, 5\}$, $\mathit{SampleRate} = \{0.1, 0.05\}$, $\mathit{DutyCycle} = \{0.2, 0.4\}$. This leads to $2^3 = 8$ possible configurations per node. The constants in the nodes' mapping functions were chosen to match Crossbow MICA2 [3] sensor nodes (power usage, transceiver parameters); the contention-loss probability in (1) was estimated by simulation (it is reasonable to assume that certain constants can be determined empirically at WSN deployment time). Subsequently, we did the same tests for 12 configurations per node, by including an extra duty-cycle value of 0.6. To get a certain robustness in the measured run times and configuration counts, for each network size and number of configurations, we analysed 10 different networks.

It turned out that the maximum number of configurations (over all steps) stays limited, even when the network size increases. For the tests with 8 configurations per node, this maximum was roughly $60 \cdot 10^3$, for both SM and TT (see Figure 4). For the case of 12 configurations per node, $|\mathcal{C}_{\mathrm{prod}}|$ increased to about $250 \cdot 10^3$. Moreover, judging from Figure 4, the average run time of the algorithm increases roughly proportionally with the number of nodes in all scenarios, which is good considering that the underlying configuration space grows exponentially. Therefore, we may conclude that the algorithm is very well scalable and thus suitable for the configuration of a WSN.

Table 4: Analysis results for 100-node example network. Differences with simulation in parentheses.

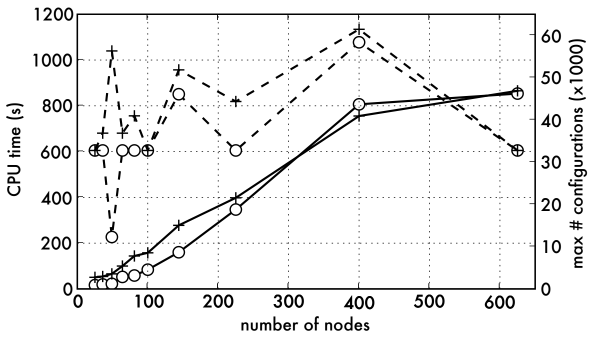
Information Completeness I^c (%)	Detection Speed S^c ($\frac{1}{\mathrm{s}} \cdot 10^3$)	Lifetime T^c (h)	Coverage Degree C^c
62 (-1%)	79 (-7%)	2125 (+2%)	0.2
65 (-7%)	47 (-2%)	2125 (+2%)	0.2
73 (-7%)	79 (-1%)	2116 (+2%)	0.2
73 (+2%)	48 (0%)	1145 (+3%)	0.4
73 (+1%)	79 (-4%)	1142 (+3%)	0.4

For example, for an instance of a 100-node network, the TT scenario, and 8 configurations per node, the algorithm took 44 seconds to complete. The total number of possible configurations is 8^{100}, but there were never more than 4096 configurations to be considered at the same time, during any of the 56 steps that the algorithm took (most often fewer). There are 5 Pareto-optimal configurations, as given in Table 4. Each of these solutions has a corresponding set of parameters for each node. We see that there are trade-offs between most quality metrics.

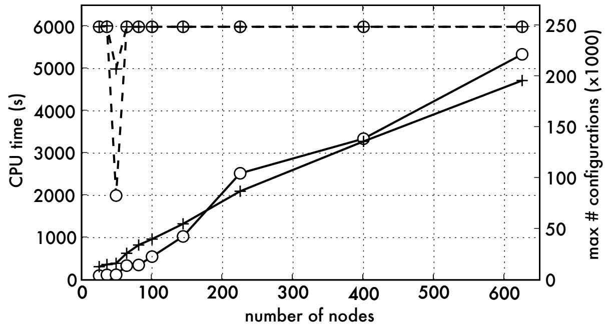
The results of Table 4 were tested in a network simulator based on OMNeT++ [8]. The nodes in the simulator were modelled after the MICA2 as well and use B-MAC [11] to communicate. Energy usage, data loss and delay were determined by running each configuration for 6 simulated hours, after which the quality-metrics information completeness, detection speed and lifetime were computed (the mapping function for coverage degree is accurate by definition, and therefore not tested). The difference between computed and simulated values is given in the table as a percentage in brackets ('-1%' indicates that the computed value is 1% lower than the simulated value). The deviations are very small in general. They are caused by inaccuracies in the model and not in the way Pareto points are computed, and can be further reduced by fine-tuning the model. The same tests were done for 18 other random networks of different sizes, for both SM and TT (5 random configurations per network). On average, the deviations were 7.1%, 4.2% and 4.1%, for completeness, speed and lifetime respectively.

Another interesting observation from the example in Table 4 is that there seem to be only two truly different lifetime values, around 1140 and 2120. This may suggest that the third and the fifth configuration are the most interesting ones, and we could have decided to remove the other three. It may be possible to reduce the configuration sets at each step of the algorithm based on such considerations, which would reduce the algorithm's run-time complexity, while maintaining a good quality in terms of the final set of configurations obtained. This is one of our ideas for future work.

We also explored the configuration space of the same network via a genetic algorithm [15]. Even after running for several hours, it returned 8 configurations (e.g. (70, 45, 1070, 0.2)), which were all distinctly dominated by the ones in Table 4. It turns out that configurations with the highest metric values are so rare and isolated in the total space of size 8^{100}, that the genetic algorithm is doomed to fail: the probability of finding the Pareto points goes to 0. The best metric values found were 70, 45, 1137 and 0.2 (order as in the table), which is 4%, 34%, 46% and 50% lower than computed by our method. This result confirms the expected result that a search space of 8^{100} is too large to search ef-



(a) 8 configurations per node



(b) 12 configurations per node

Figure 4: Experimental results: for increasing network size, the run time (solid lines) does not grow more than linearly for both the SM (circle markers) and TT (cross markers) scenarios and two different sizes of the parameter space. The reason for this linear growth is that the maximum number of simultaneously considered configurations at any clustering step (dashed lines) stays limited.

ficiently and accurately via a randomised approach, and it emphasises the strength of our exact algebraic approach.

7. CONCLUSION

We have introduced a method for configuring the nodes of a WSN such that high-level QoS constraints are met. The method is based on Pareto algebra, which is used to reason about QoS trade-offs. It features an algorithm that keeps the working set of possible configurations small, by analysing parts of the network one by one, and meanwhile discarding configurations that are Pareto dominated by others. We gave accurate models for WSN doing target tracking and spatial mapping, in which the available trade-offs are made explicit. We showed how to configure such WSN with the proposed method and presented test results for various network sizes. These results show that the method is scalable and therefore practically usable for WSN, even with large numbers of nodes. Furthermore, the nature of the algorithm makes it very well suitable for a distributed implementation, which scales even better and is suitable for run-time application. We also showed that a traditional genetic algorithm approach does not scale. The WSN models and configuration scheme can easily be extended for more complicated applications and networks. Future work will focus on expanding the method with middleware features, such as adaptation to run-time changes. Another interesting aspect for future work is complexity control. The algorithm's complexity is dominated by the size of the working set of configurations. This set can be kept small by limiting the number of Pareto points at intermediate steps, e.g. via approximation techniques or simply via selection of the most interesting configurations from the quality perspective.

8. REFERENCES

[1] K. Akkaya and M. Younis. An energy-aware QoS routing protocol for wireless sensor networks. In *ICDCSW 2003, Proc.*, pages 710–715. IEEE, 2003.

[2] D. Chen and P. K. Varshney. QoS support in wireless sensor networks: A survey. In *Int. Conference on Wireless Networks (ICWN 2004)*. CSREA Press, June 2004.

[3] Crossbow technology website. http://www.xbow.com.

[4] M. Geilen, T. Basten, B. Theelen, and R. Otten. An algebra of Pareto points. *Fundamenta Informaticae*, 78(1):35–74, 2007.

[5] S. Haykin. *A Introduction to Analog and Digital Communications*. John Wiley & Sons, 1989.

[6] T. He, J. Stankovic, C. Lu, and T. Abdelzaher. SPEED: A stateless protocol for real-time communication in sensor networks. In *ICDCS 2003, Proc.* IEEE, May 2003.

[7] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek. On quality of service optimization with discrete QoS options. In *Real-Time Technology and Applications Symposium, Proc.* IEEE, June 1998.

[8] OMNeT++ website. http://www.omnetpp.org.

[9] G. Palermo, C. Silvano, and V. Zaccaria. Multi-objective design space exploration of embedded systems. *Journal of Embedded Computing*, 1(3), 2006.

[10] M. Perillo and W. B. Heinzelman. Providing application QoS through intelligent sensor management. In *Int. Workshop on Sensor Network Protocols and Applications (SNPA '03)*. IEEE, 2003.

[11] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Embedded networked sensor systems (SenSys '04), Proc.*, pages 95–107, New York, NY, USA, 2004. ACM Press.

[12] K. Römer, O. Kasten, and F. Mattern. Middleware challenges for wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):59–61, 2002.

[13] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *39th Design Automation Conference (DAC 2002)*, pages 880–885, New Orleans LA, USA, June 2002. ACM Press.

[14] Y. Yu, B. Krishnamachari, and V. Prasanna. Issues in designing middleware for wireless sensor networks. *IEEE Network*, 18(1):15–21, Jan/Feb 2004.

[15] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, Nov 1999.