Quality-of-service provisioning for dynamic heterogeneous wireless sensor networks

Quality-of-Service Provisioning for Dynamic Heterogeneous Wireless Sensor Networks

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven, op gezag van de
rector magnificus prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op donderdag 31 oktober 2013 om 16.00 uur

door

Marcel Steine

geboren te Rotterdam

Dit proefschrift is goedgekeurd door de promotor:

prof.dr.ir. T. Basten

Copromotor:
dr.ir. M.C.W. Geilen

Quality-of-Service Provisioning for Dynamic Heterogeneous Wireless Sensor Networks
By Marcel Steine, 2013
A catalogue record is available from the Eindhoven University of Technology Library
ISBN: 978-90-386-3468-5

Quality-of-Service Provisioning for Dynamic Heterogeneous Wireless Sensor Networks

The work presented in this thesis has been partially supported by the European Commission under the Framework 6 IST Project Wirelessly Accessible Sensor Populations (WASP, IST-2006-034963).

This work was carried out in the ASCI graduate school.
ASCI dissertation series number 286.

Printing: Printservice Technische Universiteit Eindhoven

Abstract

Quality-of-Service Provisioning for Dynamic Heterogeneous Wireless Sensor Networks

A Wireless Sensor Network (WSN) consists of a large collection of spatially distributed autonomous devices with sensors to monitor physical or environmental conditions, such as air-pollution, temperature and traffic flow. By cooperatively processing and communicating information to central locations, appropriate actions can be performed in response. WSNs perform a large variety of applications, such as the monitoring of elderly persons or conditions in a greenhouse.

To correctly and efficiently perform a task, the behaviour of the WSN should be such that sufficient Quality-of-Service (QoS) is provided. QoS is defined by constraints and objectives on network quality metrics, such as a maximum end-to-end packet loss or minimum network lifetime. After defining the application we want the WSN to perform, many steps are involved in designing the WSN such that sufficient QoS is provided. First, a (heterogeneous) set of sensor nodes and protocols need to be selected. Furthermore, a suitable deployment has to be found and the network should be configured for its first use. This configuration step involves setting the controllable parameters of all nodes, such as the neighbouring node(s) to communicate to and the transmission power of its radio. Configuring the network is a complex task as the number of parameters and their possible values are large and trade-offs between multiple quality metrics exist. High transmission power may result in a low packet loss to a neighbouring node, but also in a high power consumption and low lifetime. Heterogeneity in the network causes the impact of parameters to be different between nodes, requiring parameters of nodes to be set individually. Moreover, a static configuration is typically not sufficient to make the most efficient trade-off between the quality metrics at all times in a dynamic environment. Run-time mechanisms are needed to maintain the required level of QoS under changing circumstances, such as changing external interference, mobility of nodes or fluctuating traffic load.

This thesis deals with run-time reconfiguration of dynamic heterogeneous wireless sensor networks to maintain a required QoS, given a deployed network with selected communication protocols and their controllable parameters. The main contribution of this thesis is an efficient QoS provisioning strategy. It consists of three parts: a re-active reconfiguration method, a generic distributed service to estimate network metrics and a pro-active reconfiguration method.

In the re-active method, nodes collaboratively respond to discrepancies between the current and required QoS. Nodes use feedback control which, at a given speed, adapts parameters of the node to continuously reduce any error between the locally estimated network QoS and QoS requirements. A dynamic predictive model is used and updated at run-time, to predict how different parameter adaptations influence the QoS. Setting the speed of adaptation allows us to influence the trade-off between responsiveness and overhead of the approach, and to tune it to the characteristics of the application scenario. Simulations and experiments with an actual deployment show the successful integration in practical scenarios. Compared to existing configuration strategies, we are able to extend network lifetime significantly, while maintaining required packet delivery ratios.

To solve the non-trivial problem of efficiently estimating network quality metrics, we introduce a generic distributed service to distributively compute various network metrics. This service takes into account the possible presence of links with asymmetric quality that may vary over time, by repeated forwarding of information over multiple hops combined with explicit information validity management. The generic service is instantiated from the definition of a recursive local update function that converges to a fixed point representing the desired metric. We show the convergence and stability of various instantiations. Parameters can be set in accordance with the characteristics of the deployment and influence the trade-off between accuracy and overhead. Simulations and experiments show a significant increase in estimation accuracy, and efficiency of a protocol using the estimates, compared to today's current approaches. The service is integrated in various protocol stacks providing different kinds of network metric estimates.

The pro-active reconfiguration method reconfigures in response to predefined run-time detectable events that may cause the network QoS to change significantly. While the re-active method is generally applicable and independent of the application scenario, the, complementary, pro-active method exploits any a-priori knowledge of the application scenario to adapt more efficiently. A simple example is that as soon as a person with a body sensor node starts walking we know that several aspects, including the network topology, will change. To avoid degradation of network QoS, we pro-actively adapt parameters, in this case, for instance, the frequency of updating the set of neighbouring nodes, as soon as we observe that a person starts to walk. At design time, different modes of operation are selected to be distinguished at run-time. Analysis techniques, such as simulations, are used to determine a suitable configuration for each of these modes. At run time, the approach ensures that nodes can detect the mode in which they should operate. We describe the integration of the pro-active method for two practical monitoring applications. Simulations and experiments show the feasibility of an implementation on resource constrained nodes. The pro-active reconfiguration allows for an efficient QoS provisioning in combination with the re-active approach.

Contents

Chapter 1

Introduction

This chapter discusses Wireless Sensor Networks (WSNs) and the challenging task of configuring them such that the end-user can successfully use the WSN to perform a task. The central problem of run-time reconfiguration is introduced. In Section 1.1, we give an overview of the practical interest of WSNs and discuss the importance of providing sufficient Quality-of-Service (QoS). We furthermore discuss the dynamic and heterogeneous nature of typical WSNs. In Section 1.2, we state the problem of efficient QoS provisioning. We elaborate on the impact both dynamism and heterogeneity have on the complexity of QoS provisioning. Section 1.3 gives an overview of the QoS provisioning strategy introduced in this thesis. The contributions of this thesis are presented in Section 1.4. Finally, we discuss the contents of this thesis in Section 1.5.

1.1 Motivation

A WSN consists of a set of small autonomous devices, called *nodes*, that collaborate to perform a given task. The nodes are small, low cost, devices combining one or more sensors with a processing unit, storage and wireless communication interface. Depending on the application, actuators, such as a display or speaker, may also be incorporated in the nodes. The ever decreasing size of the nodes results in the application domain of WSNs to be broad and expanding. Due to their limited size, dependence on batteries or power harvesting devices, and inability to be easily replaced or maintained, the sensor nodes have significant limitations on the available processing power, storage capacity and especially power consumption. This, together with the highly cooperative nature of the nodes, makes the design of WSNs significantly different from traditional wireless ad-hoc networks.

The use of WSN in practice is vastly increasing. We start to encounter WSNs throughout our entire live; from the cradle unobtrusively monitoring a baby's health status to elderly care centers equipped with sensors to, for example, give a

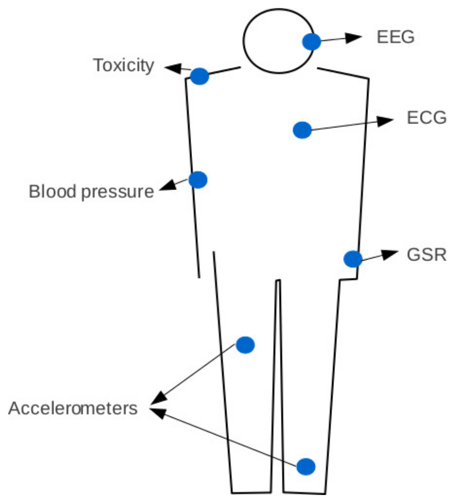


Figure 1.1: Typical body sensors

warning if the stove is on and left unattended. Almost any scenario that implies monitoring of sensible phenomena qualifies as a target application for WSNs, of which examples can be found in [28, 33, 34, 47, 66, 75, 76]. A typical application is health-monitoring of elderly [15, 57, 74, 78]. Many elderly dread the prospect that chronic medical issues require them to regularly seek medical assistance by visiting the doctor. Especially with the increasing percentage of elderly [72], this will result in large costs for society to support the current health-care system. By extending the monitoring of health from the doctor's office and hospital to a persons home, expensive medical assistance and hospital care can be limited by early detection and prevention of health issues. A wireless sensor network is a suitable technology enabling convenient, unobtrusive, in-home monitoring. Nodes can be equipped with several types of sensors and can be attached to body, clothes or accessories, such as a watch [36]. Figure 1.1 shows some of the typical sensors. EEG (electroencephalogram) sensors monitor brain activity, ECG (electrocardiogram) monitors heart signals and GSR (galvanic skin response) monitors the skin conductivity, used as a measure of emotional response. Toxicity sensors can be used to detect dangerous levels of toxins, such as carbon monoxide, in the area of the monitored person. Accelerometers are used to determine the activity and mobility of the body. The data collected by the sensors is often collected at a central on-body node which communicates with the rest of the network. Besides the nodes attached to persons, static nodes are deployed at fixed locations throughout the area in which persons are monitored. Figure 1.2 shows an example deployment of static nodes, and mobile nodes with fixed limited mobility, as used in an experiment by the Roessingh Research and Development center [51] to monitor

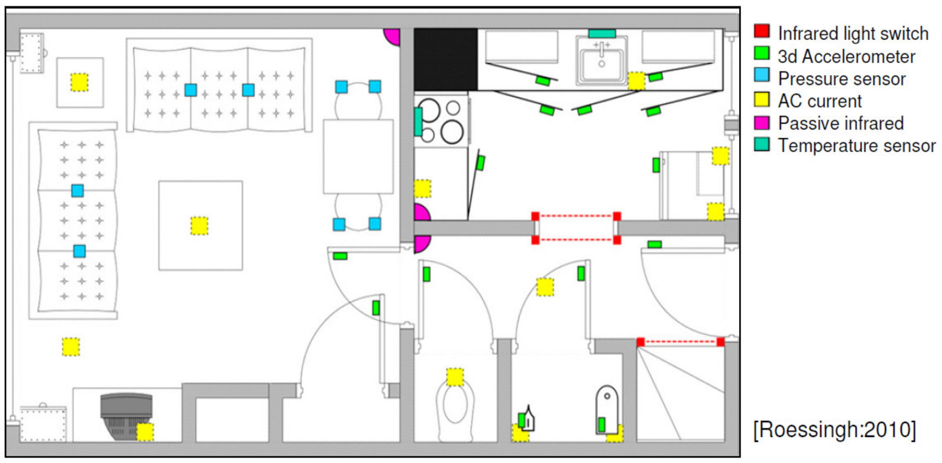


Figure 1.2: Node placement in health-monitoring experiment

persons with COPD (Chronic Obstructive Pulmonary Disease) in the ALwEN project [1]. For patients with COPD the amount of activity performed during the day is an important parameter in the treatment of symptoms. Static nodes collect information on, for example, environmental properties, such as temperature or humidity, and the current activity of the person, such as watching television or cooking. Data from the static and mobile nodes is propagated to one or more central locations over one or multiple hops. At a central location, a sink, data is collected and combined to derive a complete picture of the current health for diagnosis. Care takers may predict upcoming health issues and can remotely adjust the treatment of the monitored persons or suggest changes to their environment, e.g., temperature setting, or behaviour.

Practical monitoring networks, such as for the health-monitoring application as described above, are inherently *heterogeneous*. Nodes differ in their capabilities, such as the attached sensors, processing power and storage capabilities. As nodes attached to a monitored person are typically required to have a small form factor they have more stringent resource constraints than static nodes that have more freedom and space to be deployed at a convenient location. Besides the difference in hardware, the task performed may vary between nodes. Nodes responsible for monitoring hearth signals, typically sample ECG sensors a couple of hundred times a second. The sampling of a temperature sensor can be much less frequent as it typically only changes slowly over time. As a result, the amount of data processed and communicated by the nodes varies. Also the location of a node in the network impacts this data load. Nodes closer to a sink, or in a part of the network with a higher node density, often need to forward a larger amount of data. Finally, external interference due to signal attenuation by fading and shadowing, and multi-path effects, are environment dependent and typically non-uniform across the deployment area. Nodes closer to an interference source, such as a microwave or person walking around, will typically need to spend more effort to communicate to other nodes. Typical WSNs are furthermore *dynamic*. Characteristics of the network are not only heterogeneous, but do also change during network operation. Microwaves may be turned on and off, persons move through the monitored area, doors may be opened and closed or events occur that require an increased frequency of sensor readings. Another important factor is the mobility of nodes. Moving nodes change the topology of the network. Aspects such as the node density, link qualities, the distance to the sink, and communication paths thereby change. All these events affect the environment in which the node operates. The dynamics in a network vary in frequency and duration. The impact of environmental properties, such as temperature and humidity, on wireless communication capabilities is slow and small, while a quickly moving person can have an immediate large, but relatively short, impact on link qualities and network topology. A dynamic and heterogeneous monitoring sensor network, consisting of both static and mobile nodes, is typically found in practice and is the main focus of this thesis.

To enable the end-users, e.g., the care takers, to successfully use the WSN for their intended application, e.g., predict upcoming health issues, it is important that the WSN meets explicit performance targets, which may change over time. Performance targets could, for example, relate to the percentage of sensor data samples that is successfully received at the sink, i.e., end-to-end delivery ratio. Sufficient ECG samples need to be delivered at the sink to support accurate continuous monitoring of a person's hearth-rate. The time between sampling the sensor data and receiving it at the sink allowing it to be analysed, i.e., the end-to-end latency, is another aspect to be constrained to ensure that changes in health properties can be observed in time. Furthermore, the power consumption of all nodes is typically heavily constrained and should be as low as possible to maximize the network lifetime. We refer to the target values of these network performance metrics as the required *network Quality-of-Service*. They are expressed by *constraints*, e.g., an end-to-end latency of at most 10 seconds, and *objectives*, e.g., maximizing the network lifetime. The increasing application of WSNs and increasing expectations of its end-users emphasize the importance of providing a sufficient network QoS, making QoS provisioning an important topic in recent research. This thesis deals with the complex problem of ensuring that the current network QoS requirements are met by the dynamic heterogeneous WSN.

1.2 Problem Statement

Many steps are involved in designing a WSN to provide the QoS required to successfully execute an application. In [52] an overview of this extremely large design-space is given. The first step of WSN design is the selection and/or development of a set of sensor nodes and communication protocols. Especially in the early years of WSN research, a lot of effort was put in designing specialized hardware and communication protocols for WSN. Subsequently, a suitable positioning of the nodes in the area we want to monitor has to be found. This involves many considerations, such as the density of the network [82]. Before actually deploying the nodes, the network should be configured for its first use. The configuration step involves setting all controllable protocol parameters of the network and each individual sensor node, such as the radio transmission power, buffer sizes, MAC protocol duty cycle and selected routing parent(s). These controllable parameters play an important role in how the network behaves. Configuring the network such that all QoS constraints are met, while objectives are optimized, is non-trivial. The configuration-space is enormous due to the large number of nodes, parameters and potential values per parameter. Furthermore, improving one aspect of the system may deteriorate other important characteristics, i.e., trade-offs between multiple metrics exist. Typical conflicting metrics are end-to-end delivery ratio and power consumption. Parameters that positively influence the end-to-end delivery ratio, such as the transmission power of a node, will often negatively influence the power consumption of that node. Several approaches exist to efficiently find a set of node configurations (i.e., the Pareto points [45]) for the initial static (heterogeneous) WSN at design-time [6, 13, 16, 22, 40, 87].

In practical scenarios, such as health monitoring, we see that WSNs operate in a dynamic context where events, such as changing external interference, mobility of nodes and fluctuating traffic load, constantly influence the behaviour, and thereby the provided QoS, of the network. On the other hand, QoS requirements of the application may change. Both may result in an emerging mismatch between the required and provided network QoS. A potential solution would be to focus on a design-time solution where we determine the worst-case situation in which the network could operate, determine the best configuration for that situation, and use it during the entire operation of the WSN. The main disadvantage is that a single worst-case configuration is typically not able to provide an efficient trade-off between the QoS metrics at all times, especially if the network is not often experiencing its worst-case situation. A worst-case situation may furthermore be hard to determine due to unknown impact of run-time dynamics. With a worst-case configuration, we often experience QoS over-provisioning, where constraints are met, but with a superfluous use of resources, limiting the optimization of objectives, such as network lifetime.

For efficient QoS provisioning in a WSN with unpredictable dynamics, we cannot resort to a design-time solution. We therefore focus on run-time QoS provisioning to efficiently tune the behaviour of the network to ensure that the time the network is providing a sufficient QoS is as high as possible. The adaptation of parameters is an effective and flexible way to influence the behaviour of nodes at run-time. Instead of using a single static configuration, the parameters of one or more nodes are adapted according to a given strategy. Finding a single configuration for a static deployment is already found to be a difficult task due to the large configuration space and trade-offs involved. Run-time reconfiguration significantly increases the problem as it constantly requires to find a suitable configuration for the current dynamics in the network, with only limited time and resources available.

The main challenge of any run-time reconfiguration approach is to provide a strategy to decide when and how to adapt controllable parameters. One could think of either a centralized or distributed strategy. With a centralized approach, such as, for example, proposed in [26, 86], information on the performance of the network is collected at a central location after which it is determined how the controllable parameters of every node should be set to provide sufficient QoS. The values of the controllable parameters are then communicated to all nodes to inform them on any changes they should make to their configuration. It allows decision making based on knowledge of the QoS of the (entire) network, but scalability is limited. With larger networks, a lot of information needs to be collected and disseminated. The time needed to respond to a changing QoS may be high as the effects of dynamics first need to be observed by the central location. Nodes have to wait for the decisions made and communicated by this central location. For fast (and frequent) dynamics there is no time to wait for the centralized location to initiate a reconfiguration. Most current run-time reconfiguration approaches apply a distributed strategy where nodes themselves observe when and decide how to adapt parameters. Several examples can be found in [12, 63, 69, 81]. More details on current work and references to other literature that is associated to specific parts of this work are given in the respective chapters. In a distributed system, such as a WSN, distributed reconfiguration is expected to be more efficient, but sufficient information is needed by every node to make accurate reconfiguration decisions in a distributed manner. The focus of current distributed run-time reconfiguration approaches is to optimize for one or more local performance metrics, typically power, instead of the multiple (conflicting) network-level metrics that are of interest to the end-user. In a dynamic heterogeneous network, a node is typically not able to make efficient reconfiguration decisions by only focusing on local metrics, such as the latency to the parent as part of the latency to the sink. This is because a QoS constraint on a network metrics, e.g., latency to the sink, can typically not efficiently be translated to fixed QoS constraints on a node metric, e.g., latency to the parent node for every node on the path to the sink.

In summary, current QoS provisioning approaches are either centralized and able to support network QoS provisioning, or distributed and aim at local QoS provisioning. What we need for efficient reconfiguration of dynamic heterogeneous WSNs to ensure a QoS defined by multiple conflicting network metrics, is an approach that is *distributed* and aims at *network QoS provisioning*. This requires efficient collaboration between nodes. The reconfiguration strategy should furthermore be efficient in terms of overhead and flexible to deal with the different kinds of dynamics present in a WSN.

1.3 Approach

In this thesis, we introduce a reconfiguration strategy for QoS provisioning in dynamic heterogeneous WSNs. We assume the protocol stack to be selected and to have an interface to adapt the controllable parameters of any of its protocols. We furthermore assume the network to be initially deployed and configured.

To allow distributed reconfiguration decision making while keeping network QoS in mind, estimates of the metrics pertaining to the entire network or parts of the network, such as the latency to the sink or minimum lifetime of the nodes on a path, are locally used by the nodes. To efficiently solve the non-trivial problem of estimating network metrics in dynamic heterogeneous WSNs independently of the protocol using the estimates, we introduce a generic distributed service. This service can be instantiated to locally estimate any type of network metric that can be expressed by a recursive equation in terms of old estimates of the network metric and additional information collected from the network. Dynamism and heterogeneity complicate the estimation of this kind of information. N-hop forwarding of information is used to accommodate information exchange between neighbouring nodes to overcome asymmetric links caused by heterogeneity in the network. As the values of network metrics change over time, n-hop forwarding is repeated at a given update interval. The update interval allows a trade-off between the accuracy of the estimates and the overhead involved. With fast impact dynamics, more overhead is needed to get accurate estimates compared to a network with slower dynamics.

Given that nodes have such estimates of the network metrics, nodes can locally determine whether sufficient QoS is provided by the network. If not, reconfiguration is needed. The first part of our QoS provisioning approach is a re-active reconfiguration approach, which repeatedly checks current QoS estimates and directly reconfigures nodes based on discrepancies with the currently required network QoS. The re-active approach is independent of the dynamics that cause an impact on QoS and can therefore directly be used in any scenario. Important for the efficiency of the re-active approach is its responsiveness to QoS errors. The responsiveness, or speed, of the approach can be influenced by several parameters, including the parameters of the underlying metric estimation service, and needs to be set in accordance with the impact and frequency of the dynamics in the network. The frequency of reconfiguration should be high enough to ensure that the time spent in a state with insufficient QoS is short enough. To ensure a stable reconfiguration approach, updates of local estimates of network QoS, propagated by an instance of our distributed service, should be available fast enough. This results in a trade-off between the overhead of re-active reconfiguration and its responsiveness. To determine how to adapt parameters to get the right level of QoS, adaptive predictive models are used. These models are adapted at run-time to increase prediction accuracy.

A re-active approach does not exploit the additional a-priori knowledge of network dynamics or application behaviour. A complementary pro-active approach is introduced that explicitly exploits knowledge of the dynamics to adapt the configuration before dynamics affect the QoS. It classifies the current situation, and adapts the configuration accordingly if needed, using a predetermined response in terms of adapting controllable parameters. Exploiting a-priori knowledge of dynamics in the network avoids waiting for a suboptimal QoS, but anticipates for it by reconfiguring for predefined dynamic changes. With design-time knowledge of the occurrence of events we can furthermore consider the option of changing parameters that are set at the network level, equal for all nodes. These network parameters, such as the duty cycle for some MAC protocols, have a large impact on the QoS, but require synchronization to be changed.

By using the complementary re-active and pro-active approach in a single QoS provisioning strategy, we exploit design-time knowledge of the dynamics in the network, while we are still able to respond to unexpected or unpredictable events impacting the network QoS.

1.4 Contributions

The overall contribution of this thesis is a run-time reconfiguration approach that efficiently provides a required QoS, defined by multiple network quality metrics, in a dynamic heterogeneous WSN. We focus on monitoring networks that consist of a combination of static and mobile nodes, communicating sensed data to one or more central locations. The reconfiguration approach combines multiple parts described below. While these parts combined form an efficient run-time reconfiguration approach, on their own, they have their practical use both in- and outside the area of QoS provisioning. Our contributions can be divided in the following sub-parts:

- A distributed service to efficiently estimate network-level metric information in dynamic heterogeneous WSNs (Chapter 2). Such service is generally applicable for any protocol that uses local estimates of network-level metrics or consensus on network metrics. An initial version of our service is published in [58].
- A re-active run-time reconfiguration approach in which every node controls its parameters based on local estimates of network QoS (Chapter 3). It responds to any deviations in the locally known estimates from the required network QoS. With this response the heterogeneity in the adaptation impact on network QoS is considered. Initial work on our re-active reconfiguration approach has been published in [59].
- A pro-active run-time reconfiguration approach in which every node controls its parameters based on a-priori defined observable events (Chapter 4). This knowledge is exploited in the reconfiguration process to adapt controllable parameters before a change in dynamics affects the network QoS. This work has been published in [60].

1.5 Thesis Overview

The thesis continues in Chapter 2 with the introduction of our generic distributed estimation service. The problem of distributed estimation of network metrics is discussed in detail. The service is explained, together with its convergence and stability properties. Performance of the service is analysed using simulations and experiments with an actual deployment.

Chapter 3 introduces our re-active run-time reconfiguration strategy. The re-active approach, and the required instantiations of our service, are discussed in detail. Simulations and experiments with an actual deployment are used to explore the parameters of the approach, to analyse its performance and compare it with current techniques.

In Chapter 4, we introduce our pro-active run-time reconfiguration approach. Details on the approach are given and we discuss how to integrate a pro-active approach for a practical cow-monitoring and office-monitoring application. Analysis of the integration is done using simulations and experiments with an actual deployment.

Chapter 5 gives directions for future work and concludes this thesis.

Chapter 2

Distributed Estimation of Network Metrics

Many protocols for WSNs rely on sensor nodes to locally have accurate estimates of metrics pertaining the network as a whole, of which examples can be found in [21, 24, 32, 41, 43, 54, 83]. Nodes can estimate these network metrics in a distributed fashion by exchanging information with neighbouring nodes. A typical example is the minimum-energy path from a node to a given reference node, used for selecting an energy-efficient routing parent [10, 11, 61, 80]. It can locally be determined from the minimum-energy path to the reference node of all neighbouring nodes, and the energy needed to reach them. The accuracy of locally estimated network-level metrics is directly related to the availability of accurate information. Both dynamism and heterogeneity found in typical WSNs make the accurate collection and quick dissemination of information a non-trivial task. Dynamics, such as moving nodes and changing external interference, cause information to become outdated. Heterogeneity, due to, for example, heterogeneous settings of the transmission power, can result in communication links with asymmetric quality, complicating the information dissemination. Existing protocols often rely on an ad-hoc local broadcasting approach to estimate network metric values ignoring the impact of dynamism and/or heterogeneity. In this chapter, we introduce an efficient generic distributed service for dynamic heterogeneous WSNs that provides nodes with estimates of metrics pertaining to the entire network. Nodes disseminate information and manage the validity of received information to allow iterative computation of accurate estimates. Estimated information can then be used for the efficient execution of any protocol in the protocol stack. Local dissemination can employ controlled n-hop forwarding to overcome asymmetric links. The dissemination of information is repeated at a given interval to avoid stale information caused by changes in the local information. With the required information locally available, nodes regularly estimate the network metrics.

Instantiating our generic service consists of three steps. First, a recursive function characterizing the network metric to be estimated is defined. This recursive function is in terms of information collected from one or more other nodes in the network. Second, convergence and stability of this recursive function needs to be shown. With a converging and stable recursive function, the distributed estimation of the service is guaranteed to converge to the correct estimates for all nodes. Finally, the parameters of the service need to be set to tune its behaviour for the characteristics of the deployment.

In this chapter, we state the recursive functions, and show how convergence and stability of the service can be shown, for several specific instances of the service. The design and parameters of the service allow an easy integration into existing and new WSN deployments and allow a trade-off between the accuracy of the estimates and the overhead of the service. The characteristics of the deployment have a large impact on this trade-off. Using simulations and experiments, we provide insight in this impact and insight on how to set the parameters of the service for a required accuracy. We furthermore show that using the service results in a significant increase in the accuracy of the estimated network metric information and the efficiency of a protocol using the estimates, compared to the typically used local broadcasting approach. We position our approach as a *service*, a commonly used abstraction for providing functionality to other protocols, which works independent of other protocols in the network stack. This relieves protocols from the non-trivial task of obtaining their own estimates. Information can furthermore easily be shared among multiple protocols in the stack avoiding additional overhead.

The remainder of this chapter is organized as follows. In Section 2.1, we elaborate on the importance of estimating network metrics. Section 2.2 introduces our distributed service to estimate network metrics locally on the nodes. In Section 2.3, we go into more detail on the convergence and stability of the used iterative distributed way to estimate network metrics. Section 2.4 discusses the performance of the service in more detail and shows simulation and experimental results. In Section 2.5, we discuss related work. Section 2.6 concludes.

2.1 Importance of Network Metric Estimation

A WSN is a distributed system, but protocols often rely on information pertaining to the (entire) network. Consensus problems cover a large number of problems where nodes need to reach an agreement regarding a certain quantity of interest that depends on the state of all nodes [43]. A typical example of a quantity of interest is the maximum residual power in a cluster of nodes as used by, for example, cluster-head and leader election protocols [21, 83]. The average of measurements done by a set of nodes is used by, for example, data fusion [41] and clock-synchronization [32] protocols. Many protocols furthermore rely on estimates of the cost of communication paths towards a reference node, where the cost can take any form, such as the hop-count or expected energy or latency of sending a packet over the path. The most obvious use of the minimum-cost path information is for routing and data collection protocols, where packets are forwarded to the 'best' neighbouring node(s) through which a sink can be reached. With Directed Diffusion [24] nodes determine their parent, or 'gradient', based on the rate of received interest packets. Gradient-Based Routing (GBR) [54] is a modified version of Directed Diffusion, where nodes record the number of hops to reach the sink and packets are only forwarded when received from a node with a higher hop-count. As energy-efficiency is important for WSNs, many routing protocols focus on energy-efficient routing [10, 11, 61, 80] and use energy-related factors, such as communication energy consumption rates and the residual energy levels, to determine the cost of a path. Furthermore, the Collection Tree Protocol [17] as currently available in TinyOS [70], depends on information of the minimum number of transmissions to reach a sink. Besides routing protocols, a minimum-cost path is also used in localization protocols, for example based on the well-known 'DV-Hop' protocol [42]. This localization protocol derives the physical location of a node from the number of hops that the node is away from anchor nodes with a known location.

The above protocols have in common that for the iterative distributed estimation of *network metrics*, information from other nodes in the network should be collected. They only differ in the kind of information collected and the way to compute local estimates from the collected information. For example, the estimation of the maximum value in the network can be done by each node selecting the maximum value from its own value and that of neighbouring nodes. It is easy to see how repeating this process in all nodes disseminates this information throughout the network. Similarly, the cost of the minimum-cost path to a given sink can be estimated from the estimated minimum-cost to the sink of neighbouring nodes to which it can send packets and the cost to communicate to them. The network metric to be estimated can be equal for all nodes, such as the maximum, or differ per node, such as the minimum-cost-path cost to the sink.

For the correct and efficient operation of protocols, it is important that the local network metric estimates are *accurate*. For example, Gradient-Based Routing [54] depends on local estimates of the minimum number of hops, hop-count, to send packets to the sink. Packets are only forwarded if received from nodes with a higher hop-count. If nodes estimate their hop-count higher than it really is, routing becomes suboptimal and the network load may increase. Underestimating hop-count can even result in packets not reaching the sink at all.

The accuracy of estimations is directly related to the availability of accurate information from neighbouring nodes. In most current work, this information is simply assumed to be available, ignoring the practical challenges of obtaining this information, or the distribution of the information is done in an ad-hoc manner as part of the protocol. This is usually done using a local broadcasting approach, where the information is disseminated by simple broadcasting. It is thus implicitly assumed that a broadcast is enough to efficiently communicate to all nodes that need the information. For a (homogeneous) deployment with relatively little asymmetry in the link-quality, this assumption is typically valid. In practice we often see that the link-quality, for example expressed as a Packet Reception Ratio (PRR) [77], is not symmetric [27, 84, 85]. Asymmetry (and uni-directionality) exists, for example, due to the use of heterogeneous nodes with different transmission capabilities and environmental interference. A simple broadcast might not be sufficient, but communication over multiple hops may be needed to communicate the information to the intended nodes, to overcome low quality or unidirectional links. Furthermore, information becomes outdated due to the dynamically changing behaviour of the WSN. The information, and the neighbouring nodes from which information should be used in the network metric estimation, dynamically change over time due to, for example, changing external interference, dynamic adaptation of parameters, such as the transmission power [29], or node mobility. The actual network metric value can become better, but also worse over time. Management of information validity is needed to ensure that accurate estimates are made.

2.2 Network Metric Estimation

In this section, we introduce the problem of network metric estimation, the local broadcasting approach, and our generic distributed service. We first discuss the generic problem of distributed estimation of network metrics in Section 2.2.1. In Section 2.2.2, we introduce the estimation of network metric values and used terminology in more detail with an example. In Section 2.2.3, we discuss the current approach of local broadcasting and the problems that arise when using this approach in dynamic heterogeneous WSNs. Section 2.2.4 describes the functionality of our service and parameters involved. For the ease of explanation, we focus the details of our service on estimating a particular network-level property, namely the minimum-cost path of a node to a given reference node (such as a sink or cluster-head). To make it more concrete, we focus on the estimation of the cost of the path, and the neighbouring node on the path closer to the sink, its minimum-cost parent. Although we focus on a specific network metric to introduce our service, it can estimate any network metric estimated according to the general problem statement given in Section 2.2.1. Extensions to the introduced service illustrating the broader applicability are discussed in Section 2.2.5.

2.2.1 Generic Problem Statement

Network metric values, by definition, depend on the characteristics of multiple or all nodes in the network. To locally estimate network metric values, (aggregated) information from one or more other nodes in the network is needed. We consider a class of metrics that can be expressed as a fixed point of an equation of the form

$$\bar{x} = C(\bar{x}, \bar{z}) \tag{2.1}$$

where $\bar{x}$ is a vector with the local estimates of all nodes in the network and $\bar{z}$ is a vector of additional information locally known, measured or computed by the nodes. Typical additional information is the cost of using a link, which is related to the quality of the link. A low quality link for instance, may increase the likelihood of a packet being lost and thereby may require more retransmissions and hence more time and/or energy to communicate a single packet. Many different metrics exist to measure the quality of individual links, typically referred to as Link Quality Estimators (LQEs) [2], based on sent and/or received packets. Well known examples are the Received Signal Strength Indicator (RSSI), Signal-to-Noise Ratio (SNR) and Packet Reception Ratio (PRR). Our service is independent of the specific LQE and therefore, the choice of LQE for a particular deployment is outside the scope of this chapter. C is a function that characterizes local estimates of all nodes in terms of the local estimates (of other nodes) and additional information of all nodes in the network.

The fixed point of this equation can be computed in an iterative fashion:

$$\begin{aligned} \bar{x}(0) = &\quad C^0(\bar{z}(0)) \\ \bar{x}(i+1) = &\quad C(\bar{x}(i), \bar{z}(i)) \end{aligned} \tag{2.2}$$

The new set of local estimates, $\bar{x}(i+1)$, is a function C of the currently known set of estimates $\bar{x}(i)$ and current additional information $\bar{z}(i)$ of nodes in the network. Initially, only the additional information of all nodes, $\bar{z}(0)$, is available to a node to make an estimate. In general, the initial estimate, $\bar{x}(0)$, can be a function C^0 of this additional information. In particular cases it may also be just a constant.

For the distributed iterative calculation by every individual node n from the set of all nodes N, function C can be decomposed to functions C_n, for every node $n \in N$, computing the local estimate of node n, $x_n(i+1)$. The new estimate $x_n(i+1)$ is a function C_n of the currently known set of local estimates $\bar{x}(i)$ and information $\bar{z}(i)$ of all other nodes in the network. Similarly, the initial estimate is a function C_n^0 of the additional information. The resulting equation is as follows:

$$\begin{aligned} x_n(0) = &\quad C_n^0(\bar{z}(0)) \\ x_n(i+1) = &\quad C_n(\bar{x}(i), \bar{z}(i)) \end{aligned} \tag{2.3}$$

Note that in the worst case information from all nodes is locally needed, but nodes can only communicate to neighbouring nodes. This can be resolved by a mechanism that transports the information over multiple hops, or, more commonly, by defining the equation such that only information from neighbouring nodes is required. In the latter case, updates iteratively propagate the information through the network, which is more efficient and often possible as shown in the examples discussed next. Furthermore note that the calculation will be performed completely distributedly without global synchronization.

Various practical network metrics, including the examples given in the previous sections, can be expressed according to such an equation. For the local estimation of the maximum value in the network for node n, $maxest_n$ (an instance of consensus, for example used for cluster leader election in [21, 83]), Equation 2.3 is instantiated as follows.

$$\begin{aligned} maxest_n(0) &= value_n \\ maxest_n(i+1) &= max(\{maxest_x(i) | x \in nb_n\} \cup \{value_n\}) \end{aligned}$$

The maximum in the network for node n is locally estimated to be the maximum of all the network-maximum values estimated by the set of neighbouring nodes, nb_n, and the value of node n itself, $value_n$. Initially, the maximum is estimated to be equal to its own value. More specifically, Equation 2.3 is instantiated as follows: function C_n is equal to the maximum, $\bar{x}(i)$ is the set of maximum value estimates of all nodes, where only the maximum value from the set of neighbouring nodes is of interest, and $\bar{z}(i)$ only contains the value determined by the node locally, $value_n$. C_n^0 results in the initial estimate of node n to be equal to its own value. When the network is strongly connected, all nodes will eventually have the correct estimate. A similar formula can be used to describe the iterative estimation of, for example, the minimum in the network.

In the case the local values, $value_n$, change over time, the above consensus instance estimates the maximum over the entire run-time of the network. In practice, the maximum in the network may become higher or lower over time due to changing local values. With a changing maximum, one might be more interested in *periodic consensus*. Achieving periodic consensus involves achieving consensus on a value in a given time interval, for example, a day. Periodic consensus can also be expressed in the form of Equation 2.3, if time-stamps are added to the local estimates and observed values. We furthermore assume a synchronized time is available by every node. For the distributed calculation of periodic consensus on the maximum value during a day for node n, pm_n, we can instantiate Equation 2.3 as follows, where the function sd returns true if and only if the day at which the received estimate is derived is equal to the day at which the local value is determined.

$$\begin{aligned} pm_n(0) &= value_n \\ pm_n(i+1) &= max(\{pm_x(i) | x \in nb_n \wedge sd(pm_x(i), value_n)\} \cup \{value_n\}) \end{aligned}$$

Achieving periodic consensus is similar to achieving consensus as discussed above, but an extra check is done on the estimates of neighbouring nodes used in the local estimation. If the estimate of a neighbour has a time-stamp of the same day as the local value, it is used in the estimation (included in $\bar{x}(i)$). Estimates from a different day are not used.

Besides these consensus problems, the estimation of the cost of the minimum-cost path to a reference node, the sink, also fits in the considered class of problems formulated by Equation 2.3. The minimum-cost path of a node is selected from all possible paths from that node to the sink. Several examples of protocols relying on minimum path-cost information are given in the previous sections. Given any monotone recursive function f, expressing path costs in terms of link costs, the minimum path-cost of node n, $cost_n$, can be computed using the following recursive procedure.

$$cost_n = \begin{cases} identity(f) & \text{if} \quad n = Sink \\ \min\limits_{x \in neighbours_n} f(cost_x, lc_{n,x}) & \text{if} \quad n \neq Sink \end{cases} \tag{2.4}$$

The sink node does not need information from other nodes to locally estimate its minimum-cost as it is a fixed value. The sink has a minimum-cost equal to the identity element ϵ of function f such that $f(\epsilon, a) = a$ for all a, i.e., 0 for summation, 1 for multiplication, etc. For all other nodes, Equation 2.3 is instantiated with function C_n equal to a combination of the minimum and function f. $\bar{x}(i)$ are the local estimates of path cost and $\bar{z}(i)$ are the measured link costs of the neighbours to which it can send packets. $lc_{n,x}$ equals the cost of the link from node n to neighbour x to which it can send packets. The actual minimum-cost path cost is the fixed point of Equation 2.4.

With the network metric defined as an instantiation of Equation 2.3, a node can accurately estimate the network metric if accurate estimates of $\bar{x}(i)$ and $\bar{z}(i)$ are locally available. It is the goal of any network metric estimation approach to provide nodes with these accurate estimates. How this is achieved by our service, and how this improves on the currently used local broadcasting approach, is discussed in the remainder of this section. The following example is used to support this discussion and to introduce used terminology.

2.2.2 Illustrative Example

Figure 2.1 gives an example of a six-node WSN deployment with designated sink node S. The connectivity of a WSN can be described by a weighted directed graph. The vertices of the graph represent the sensor nodes. The directed edges of the graph specify the directed communication links between nodes and are labelled with the current cost associated with using them. One could for example assume that two nodes are connected if a certain percentage of communications is successful or if received signal strength exceeds a given threshold; cost could for example represent energy usage or communication delay. For the example of Figure 2.1, we assume the cost of a link to represent the expected number of retransmissions, which is strongly related to the expected energy needed for sending a packet [55].

Unless it is completely isolated from the rest of the network, every node has neighbouring nodes from which it can receive packets and/or to which it can send packets. These neighbouring nodes can be divided into *inbound neighbours* and *outbound neighbours*.

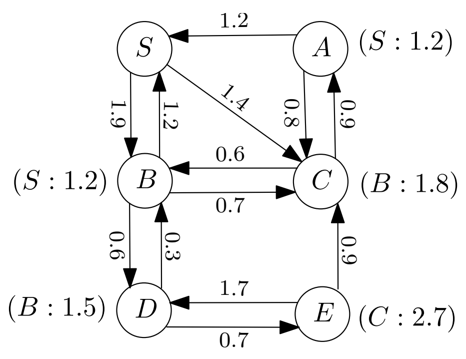


Figure 2.1: Minimum-cost path to sink S information

Definition 1. (Inbound neighbour) *An inbound neighbour of node n is a neighbouring node which has a directed edge in the connectivity graph towards node n, and from which node n thus can potentially receive packets.*

Definition 2. (Outbound neighbour) *An outbound neighbour of node n is a neighbouring node towards which node n has a directed edge in the connectivity graph, and to which it thus can potentially send packets.*

The inbound and outbound neighbours are independent of the used routing algorithm. In Figure 2.1, the only inbound neighbour of node A is node C. The outbound neighbours of node A are nodes S and C. Note that due to asymmetry in link quality, the inbound neighbours for a particular node can be different from its outbound neighbours.

The instance of Equation 2.4 to calculate the minimum sum of retransmissions on a path for node n, $minretrans_n$, is presented below, where $nrretrans_{n,x}$ is the average number of retransmissions expected for the link between nodes n and x. The recursive function f is summation (with identity element 0).

$$minretrans_n = \begin{cases} 0 & \text{if} \quad n = Sink \\ \min\limits_{x \in outbounds} (minretrans_x + nrretrans_{n,x}) & \text{if} \quad n \neq Sink \end{cases}$$

Given this recursive procedure, nodes can estimate the minimum-cost and the parent on the minimum-cost path in a distributed manner using only (i) the minimum-cost for all of its outbound neighbours and (ii) the cost of communicating to those neighbours based on LQE information. In Figure 2.1, the non-sink nodes have been labeled with their minimum-cost to S and with the parent on that path. For example, node E has two outbound neighbours. The minimum-cost path using node C as a parent has cost $1.8 + 0.9 = 2.7$, while using node D would result in a minimum-cost of $1.5 + 1.7 = 3.2$. The minimum-cost is therefore determined by E to be 2.7 and the parent to be C.

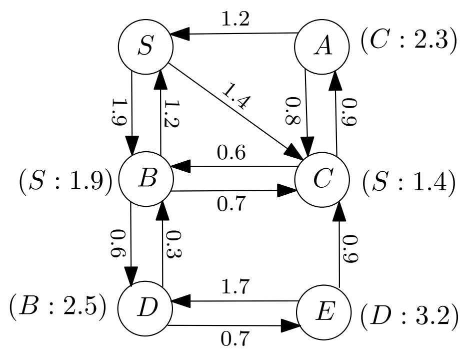


Figure 2.2: Incorrect network metric estimation with local broadcasting (assuming link symmetry)

2.2.3 Local Broadcasting

Providing network metric information to neighbouring nodes would be easy in a homogeneous static deployment, where links are symmetric, with identical link quality in both directions, and where link quality does not change over time. The inbound neighbours of a node are equal to the outbound neighbours and are always at one hop distance. Furthermore, the cost of communicating to an outbound neighbour can be derived by every node locally. The currently used local broadcasting approach would suffice in providing nodes with the required information on the minimum-cost of all outbound neighbours, and the cost of communicating to these outbound neighbours, in an iterative fashion.

With the current local broadcasting, every node repeatedly broadcasts the network metric estimates, e.g., the minimum-cost to S in the example of Figure 2.1, as soon as it is known to inform neighbouring nodes. After information from a neighbour is received, network metric estimates are updated if needed. In the case of the example, if using the neighbour from which information is received results in a path to the sink with a lower cost than the currently known minimum-cost path, the minimum-cost path is updated.

Figure 2.2 shows the estimated network metric values, in this case the minimum number of retransmissions needed to reach the sink, when applying only local broadcasting despite asymmetric links. Every node with an estimated minimum-cost broadcasts it. Nodes receiving this information (possibly inaccurately) assume they can communicate with the node from which they have received the information at a cost estimated based on the incoming link-cost. The cost of node B is thus estimated to be 1.9, while it should really be 1.2. The inaccuracy is caused by the asymmetric links which violate two assumptions made by the local broadcasting approach. First, communicating information to inbound neighbours in the presence of asymmetric links may require more than one hop, in case of unidirectional links, for example from node S to A. Communicating over multiple hops can furthermore have a lower cost (for instance higher probability of successful transmission) than shorter paths with a bad quality. Second, although LQEs based on transmitted packets may still allow nodes to determine the cost of communicating to outbound neighbours locally, the LQEs based on received packets (most of the well-known LQEs) do not. To allow nodes to accurately derive their minimum-cost path, nodes with a known minimum-cost path should therefore communicate both its minimum-path cost and the observed link-cost of the inbound neighbours to all their inbound neighbours.

This example shows the situation at a particular moment in time. With changing network metric values it is not only important to communicate the required information to the appropriate set of nodes, but also to ensure that this information stays accurate. A straightforward solution is to let nodes broadcast metric information every time a (significant) change of it is observed and use the most recently received information in the estimation. This approach can lead to an uncontrollable number of packets in the network, as there could be arbitrarily many changes. Furthermore, a problem arises with the neighbours that are lost over time, for example because they moved out of range. Nodes should eventually become aware of the fact that they lost an outbound neighbour to prevent them from using stale information. The node itself may not be able to observe this change and the outbound neighbour that does observe this can be out of range. Management of the validity of received information is needed to ensure that the estimation performed by node is accurate.

Accurate network metric estimation should consider both the complicated distribution of information due to asymmetric links and the need for information accuracy maintenance due to dynamically changing network metric values. For this we introduce our service.

2.2.4 Distributed Service

Our service consists of two concurrently executed parts: (1) information required by neighbouring nodes is communicated to them at a given interval, using n-hop forwarding to overcome any asymmetric links, and (2) the network metric value is estimated based on the locally collected and up-to-date information of outbound neighbours, using link-validity management considering dynamically changing metric values.

Network metric information dissemination To provide inbound neighbours with up-to-date information on the minimum-cost path, nodes at a given interval, referred to as the *update interval*, use a controlled n-hop forwarding approach to communicate the required information to inbound neighbours. Nodes broadcast information and nodes receiving it immediately forward the received information for a predefined number of hops, n. By using more than one hop, information can be relayed around unidirectional and low quality links (as long as an alternative path exists). The n-hop flooding is repeated iteratively to accommodate for lost

$senderId$	$cost$	$inbounds(id, linkCost)$	$seqNr$	htl

Figure 2.3: Format of the minimum-cost information packet broadcast to neighbouring nodes

Algorithm 1: Receiving minimum-cost information packet

```
1 for $in \in inbounds$ do
2     if $(myId = in.id)$ then
3         Add or update $(senderId, cost + in.linkCost, currentTime)$ in
          $myOut$;
4 end
5 if $(htl > 0 \wedge myId \neq senderId \wedge lastSeqNo[senderId] < seqNr)$ then
6     Broadcast $(senderId, cost, inbounds, seqNr, htl - 1)$;
7     $lastSeqNr[senderId] := seqNr$;
```

packets and changed network metrics. This allows nodes to locally determine their outbound neighbours and estimate network metrics.

While for local broadcasting it is sufficient to only disseminate the network metric values, the service needs additional administrative information to be communicated to allow nodes to determine whether received packets are from one of their outbound neighbours and/or should be forwarded to inform more nodes (also see Algorithm 1 as discussed in more detail later). Figure 2.3 shows the format of the packet sent by nodes every update interval. Besides the cost of the network metric, *cost*, we add the sender's unique identifier, *senderId*, and a list with information about the intended recipients, $inbounds(id, linkCost)$ with the inbound neighbours' unique identifiers *id* and link-cost to reach them, *linkCost*, to the packet. The total size of a packet is an important aspect of the overhead and mainly depends on both the density and the size of the network. These aspects influence both the average number of inbound neighbours of which information is included in the packet and the number of bytes needed to uniquely identify a node. If we assume a network of 1000 nodes, 10 bits would be needed to identify a node. If we assume that the cost of a link requires at most 10 bits, and a node has a maximum of 10 inbound neighbours, the size of the packet would stay below 256 bits (= 32 bytes), assuming that the amount of bytes needed to represent the sequence number and the hops-to-live is low. This is well below the maximum packet length of commonly used radios such as the CC1100 (255 bytes) or the CC2420 (128 bytes).

Algorithm 1 shows the steps involved when a broadcast network metric information packet is received. For this algorithm, and those in the rest of this thesis, indentation is used to indicate nesting of statements. The algorithm consists of

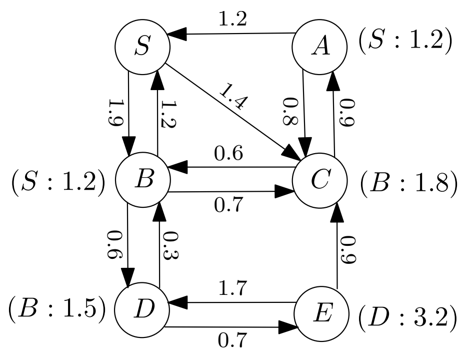


Figure 2.4: Minimum-cost path information estimated using asymmetry-aware 2-hop forwarding

two steps. The first step is to process the information in the packet if it is received from an outbound neighbour. If the receiving node is in the list of inbound neighbours of the sending node, as stored in the minimum-cost information packet, this implies that the packet is received from an outbound neighbour (lines 1 and 2). Similarly as for the example of Section 2.2.2, we assume the local network metric estimate to be the sum of the cost of the outbound neighbour to communicate to the sink, *cost*, and the cost to reach this outbound neighbour, *in.linkCost*. The value for this particular outbound neighbour is updated or added, together with the unique identifier and the current time, *currentTime*, to the locally maintained list of outbound neighbours, *myOut* (line 3). The current time is later used to check the validity of the stored information. The second step is to prevent redundant forwarding of packets by not forwarding the same information more than once. The node checks whether the packet should be forwarded (line 5). This is needed if the packet still needs at least one hop to travel and is not already forwarded by this node. The latter is checked by comparing the sequence number of the packet with the sequence number of the last forwarded packet from this sender. If a packet is forwarded (line 6), *htl* of the packet is reduced and the sequence number of the sender locally stored (line 7).

Figure 2.4 shows the result of using a 2-hop forwarding approach for every node in the network. While for the local broadcasting approach the cost and the parent are inaccurate for almost all nodes (compare Figure 2.1 and Figure 2.2), the accuracy has been significantly increased. This is due to two properties of the n-hop forwarding approach. First, the cost of outbound and inbound links is explicitly distinguished (resulting in an increased accuracy for, for example, node B) and, second, nodes are informed over more than one hop (resulting in an increased accuracy for node A). Note that, for this example, a 3-hop forwarding approach would result in an even more accurate solution, which is the optimal solution, as shown in Figure 2.1.

Algorithm 2: Update minimum-cost path information

```
1 $minCost := \infty, minParent := null$;
2 for $out \in myOut$ do
3     if $(out.lastUpdate > (currentTime - validityInterval))$ then
4         if $(out.pathCost < minCost)$ then
5             $minCost := out.pathCost, minParent := out.id$;
6     else
7         $myOut := myOut \setminus out$;
8 end
9 $myCost := minCost, myParent := minParent$;
```

Controlled n-hop forwarding allows for a trade-off between the accuracy of the derived network metric values, and the overhead required to derive it, by setting the update interval length and the number of hops, n, to forward. On the one hand, we want to reach as many inbound neighbours as possible to improve accuracy. On the other hand, we do not want to increase the traffic load in the network too much as this can potentially negatively impact the overall network performance. The value of n, in the n-hop forwarding approach, trades off the number of inbound neighbours reached for the overhead incurred. Furthermore, setting the update intervals allows us to influence the trade-off between the responsiveness to changes in the network, i.e., the temporal accuracy of the information, with the service overhead. Note that it is not necessary to synchronize the update intervals between the nodes. Appropriate setting of the update interval is done in conjunction with the validity interval introduced below and depends on the characteristics of the deployment. Setting the parameter values and the trade-offs involved are discussed in more detail in Section 2.4.

Estimating network metric information While the controlled n-hop forwarding allows us to provide inbound neighbours with the required information, the network metric values may change over time due to changes in the network. This can make earlier communicated information inaccurate. It is therefore important for the accuracy of the estimated minimum-cost path information that nodes are informed about both changes in minimum-cost and in cost of the links towards any of the outbound neighbours. To control the overhead of the service and to reduce the likelihood that nodes derive their information based on outdated information, we use an approach in which we do not assume information to be valid forever, without reconfirmation, but where received information from an outbound neighbour has a limited *validity interval.* This validity interval indicates how long we trust the received information from outbound neighbours to be valid.

Nodes should regularly check their locally collected outbound neighbour information to detect changes in the network metrics. It should not only be done to check new minimum-cost information after received updates, but also to determine whether information is still valid. Assuming that the processor time is not a significant part in the overhead, nodes can check, and update the minimum-cost path information if needed, at a regular and short interval, say one second. Algorithm 2 shows the pseudo-code to estimate the required network QoS information, i.e., the cost of the minimum-cost path, $myCost$, and the parent on this path, $myParent$. We refer to the information of the set of outbound neighbours, collected by Algorithm 1, as $myOut(id, pathCost, lastUpdate)$. For every outbound neighbour $o \in myOut$, $o.id$ is the unique identifier of the node, $o.pathCost$ is the cost of using this neighbour to communicate to the sink, and $o.lastUpdate$ is the time at which the cost information is updated. Initially, we set temporary values to an extreme value (line 1). We iteratively find the minimum-cost outbound neighbour by checking whether the cost of a valid neighbour is lower than the currently known minimum. A neighbour is valid if the last update of the neighbour is at most as long ago as the defined validity interval, $validityInterval$, (lines 2 and 3). The cost and parent estimates of the node are set to the minimum found, so far (line 4 and 5). If the neighbour is not valid anymore it is removed from the set of outbound neighbours (lines 6 and 7). Finally, the minimum-cost and parent are estimated to be the found minimum (line 9).

2.2.5 Generalization

We discussed a particular instance of our service, where local estimates and link-costs are propagated to inbound neighbours, over multiple hops if needed, to provide them with sufficient information to locally determine the minimum-cost path. The service can be generalized at several levels, using different link quality metrics, using different definitions of path cost and entirely different network metrics from the class defined by Equation 2.3.

Our service is independent of the used cost function and the chosen LQE. In this section, we focused our discussion on LQEs which determine quality by received packets, which make up the majority of well-known and most used LQEs. For LQEs which are based on transmitted packets, such as the Required Number of Packet transmissions (RNP) [9], nodes can locally determine the cost to communicate to their outbound neighbours. The controlled n-hop forwarding approach is still required to overcome asymmetric links and communicate the node's cost value, but nodes do not need to communicate their inbound neighbour's id and link-cost.

The service can furthermore easily be instantiated to work with path-costs to multiple reference nodes, by adding the minimum-cost to each of the reference nodes to the packet. Note that other information, such as the link-cost of the inbound neighbours, remains the same, which makes this service scale efficiently to multiple reference nodes. Additional information about the minimum-cost path can be added, such as the complete set of nodes on the minimum-cost path, which may be of interest to routing protocols. In that case, every node should communicate the unique identifiers of all the nodes on its own minimum-cost path to its inbound neighbours as well.

Besides its generality for the problem of maintaining minimum-cost path information, we can further extend the applicability of our service to distributed maintenance of other network-level properties that fit the pattern of recursive distributed calculation of Equation 2.3, on the condition that one can show its convergence and stability. The latter is discussed in the next section. In Chapter 3 the service is used for distributed decision making for run-time reconfiguration. The service estimates both the *maximum*-cost for a node to reach the sink using any of its routing parents and information about the lifetime of all nodes on the maximum-cost path. Other examples would be the estimation of the maximum residual energy of any of the nodes in a cluster, the reachability of services provided by specific nodes in the network or deriving information on connectivity to other nodes or clusters in the network. The main difference between estimating the different network-level properties is the metric information that needs to be disseminated to allow a local estimation.

2.3 Convergence and Stability

Our service is based on continuous distributed updating of a recursive equation. Nodes update their local estimates based on updates from their neighbours which they receive asynchronously. To ensure that the service indeed computes the intended function, we have to show that the distributed update calculations *converge* to the desired solution of the equation characterizing the network metric values. Convergence arguments assume that the input to the problem, the values $\bar{x}(i)$ and $\bar{z}(i)$ of Equation 2.3, do not change over time i. In any practical scenario, however, this assumption is not satisfied. Therefore, there will typically not be convergence to a single, constant solution. We want, besides convergence of the equation with fixed input, to enforce that the approach is *stable*, i.e., small changes in the input, do not result in large variations in the estimates. In this section, we discuss the requirements for our service to converge and provide a convergence argument, followed by a discussion on the stability of our approach.

Any suitable fixed-point theorem (e.g., Kleene, Banach) [25] or stability theorem (e.g., Lyapunov) [31] can be used to demonstrate that a *synchronous* calculation according to Equation 2.3 converges to a designated fixed point. Such theorems essentially show that the distance to the fixed-point solution, measured in some suitable metric, converges to zero. In WSNs, local updates take place *asynchronously*, even in synchronized WSNs, due to the unreliable wireless channel used for communication. As a consequence, we cannot make assumptions on the order in which information from neighbouring nodes is received and applied. Therefore, our service falls in the class of *asynchronous distributed iterative processes*. Convergence of such processes has been studied among others in [3, 4].

An individual asynchronous update of a local estimate does not by itself necessarily make the distance to the fixed-point solution measurably smaller, but a collection of updates including all nodes should. Essential for proving the convergence of asynchronous processes is the notion of an *iteration*. The sequence of all updates of local estimates is partitioned in subsequences in which every node updates itself at least once. Such a subsequence is called an iteration. Convergence is proven by first of all showing that every iteration has a finite duration. We assume that, with the stochastic process of sending packets over the communication channels in the WSN, every node will eventually receive updated information from all the required nodes. If not, convergence might still be achieved in practice, but cannot be guaranteed.

Besides having a finite duration, every iteration should bring the set of estimates known by the nodes, $\bar{x}(i)$, significantly closer to the final fixed-point solution, such that the error between the set of estimates and the final solution goes to 0. We can define a metric $e(i)$ that defines the error between the set of estimates and the final solution at time i. It is equal to 0 if and only if the set of estimates of the nodes are equal to the fixed-point solution of Equation 2.3. The following equation expresses the difference of the error between subsequent iterations.

$$e(i+1) \leq \alpha \cdot e(i) \tag{2.5}$$

If we can show that α is smaller than 1, then the following equation holds, where $e(0)$ is the initial sum of errors in the network.

$$\lim_{i \to \infty} e(i) \leq \lim_{i \to \infty} \alpha^i \cdot e(0) = 0 \tag{2.6}$$

With an error that goes to 0, every iteration brings the estimates by the nodes significantly closer to the final fixed-point solution and convergence to the fixed-point solution is shown.

The selection of metric $e(i)$ and proving that Equation 2.5 holds for $\alpha < 1$ depends on the selected function, C_n. We illustrate the process for the service estimating the maximum value in the network. Informally the argument goes as follows. Initially, there is (at least) one node which has the maximum value in the network. By the definition of an iteration, the shortest distance between the nodes that are unaware of the maximum value in the network and the closest node that is aware of this value decreases every iteration by at least one hop. At least one node previously unaware of the maximum has the correct maximum after every iteration. This reduces the total error of the estimates from the actual maximum until all nodes are aware the maximum. Metric $e(i)$ can in that case be defined as follows, where $x_n(i)$ is the local estimate of node n at time i, where n is from the set of all nodes N and $0 \leq n < |N|$, and L_{max} is the maximum value in the network.

$$e(i) = \sum_{n=0}^{|N-1|} (L_{max} - x_n(i)) \tag{2.7}$$

If $e(i) = 0$, then the estimates of all nodes are equal to the maximum value in the network and vice versa. If $e(i) > 0$, then at least one node in each iteration goes from a state in which $x_n(i) \neq L_{max}$ to $x_n(i+1) = L_{max}$. The difference between $x_n(i)$ and $x_n(i+1)$ for this node is at least $|L_{max} - L_{max_2}|$, where L_{max_2} is the second highest value originally in the network. (Note that L_{max_2} exists unless convergence has already occurred.) Then the following equation holds.

$$e(i+1) \leq e(i) - (L_{max} - L_{max_2}) \tag{2.8}$$

To determine for which α Equation 2.5 holds, we first rewrite Equation 2.8:

$$e(i+1) \leq e(i) \cdot \left(1 - \frac{L_{max} - L_{max_2}}{e(i)}\right) \tag{2.9}$$

In the worst-case, the (absolute) difference between the estimate of a node and the maximum value is L_{max}. Given that $e(i) \leq |N|\, L_{max}$, we rewrite Equation 2.9 to:

$$e(i+1) \leq e(i) \cdot \frac{|N|\, L_{max} - L_{max} + L_{max_2}}{|N|\, L_{max}} \tag{2.10}$$

We have shown that α is equal to $\frac{|N|L_{max} - L_{max} + L_{max_2}}{|N|L_{max}}$, which is always smaller than 1. The recursive equation converges and the estimates of all nodes will eventually become equal to the maximum value in the network.

For showing the convergence of the periodic maximum estimation, the same reasoning can be used. The main difference is the period in which the convergence can be achieved. For periodic convergence, the speed of convergence should be sufficiently high to converge within the given period. A similar reasoning can also be applied to show convergence of estimating the minimum-cost path, where in every iteration the distance from the sink to the nodes that have incorrect estimates increases by at least one. For any node (directly or indirectly) connected to the sink, the distance is finite and the correct value is obtained after a finite number of iterations. Note that local estimates might also (correctly) converge to infinity in the case there is no (multi-hop) connection to the sink, as the absence of a connection to the sink may cause cyclic dependencies and nodes repeatedly increasing their minimum-cost estimate.

Besides convergence, we want to show stability. Stability follows directly from the convergence argument we provided, as the steps in the distributed update do not enlarge the distance to the fixed point of Equation 2.3 being estimated. This is visualized in Figure 2.5 for the estimation of the maximum value in the network. The solid line shows the maximum value in the network, and the dashed line is the local estimate of the maximum for a given node. As shown before, the estimate will converge to the actual maximum. At time t the maximum in the network changes due to one or more dynamic events. The discrepancy between the estimate and the maximum does not become more than the difference between the new and old maximum. It directly starts to converge to the new maximum, resulting in a stable iterative estimation.

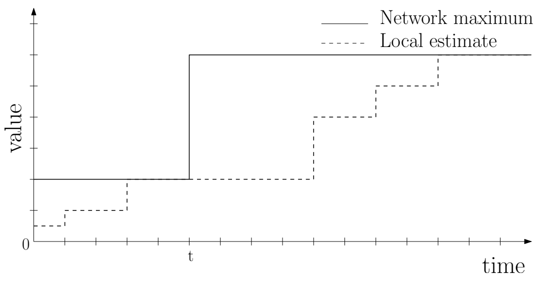


Figure 2.5: Stability of local estimates

2.4 Performance Evaluation

In this section, we evaluate the performance of our service. In Section 2.4.1, we discuss the parameters of our service and provide insights on how to parameterize the service based on the deployment characteristics. For this we use experience with extensive simulations and actual deployments. In Section 2.4.2, the performance of the service is evaluated for a typical WSN deployment and compared to the local broadcasting approach with the use of extensive simulations. Simulations allow us to do fast performance analysis of large deployments for various different parameter settings. Simulations furthermore allow us to control and replicate the environment to fairly compare and stress-test the two approaches. With the design of our service we kept in mind an easy integration into existing and new WSN deployments. In Section 2.4.3, we discuss the integration of our service in an actual test-bed deployment for two different applications with different kinds of protocol stacks and network metric information to estimate. While this actual deployment makes it harder to compare two approaches, as interference and mobility patterns are hard to replicate between experiments, it gives us insight in the feasibility of using our service in practice.

2.4.1 Setting the Service Parameters

As discussed in the previous section, our service has several parameters making it flexible for integration into a given WSN deployment.

First, the number n of hops to forward is a parameter that is set based on the connectivity structure of the network and amount of asymmetry in the network. In the example of Figure 2.4, increasing n from one to two increased the accuracy, but from two to three no further increase is observed. At design-time we can get a good idea about the connectivity structure and the number of hops needed to communicate to inbound neighbours by looking at the (worst-case) difference between transmission range and the potential locations of nodes. In [14], it is observed that forwarding the information over one or two hops is enough to overcome the asymmetry in typical deployments. Also for the performed simulations and experiments we observed that a number of hops larger than two does not significantly increase the accuracy of the maintained network metric information.

Second, we set the *update interval* at which inbound neighbours are informed, and the related *validity interval* of received network metric information, according to the amount of dynamism in the network. The update interval should be selected to be at most the validity interval to avoid that nodes frequently assume to have no valid information. To take packet-loss into consideration within the service, we can furthermore set the update interval such that the nodes are updated multiple times within the validity interval. The update interval parameter is found to be the main driver for the trade-off between accuracy and overhead of the service. Setting the value of the parameter is guided by the amount of dynamic changes of the link quality. While external interference impacts the link quality over time, we see from experience, that the speed of mobile nodes is the most important source of fast dynamic changes to the link quality requiring an adaptation of the estimated network metrics. Currently, our service is found to be very well applicable for the speed of mobility that is typically found in WSN used for monitoring of, for example, persons or animals (see next section), which make up many of the current WSN deployments. Responding to very high dynamics requires a very short update interval to provide accurate network metric values, resulting in a potentially large overhead of our service or any other approach to estimate network metrics locally. The amount of overhead allowed should be at least the overhead needed for the service to provide the required estimation accuracy. The practical use of network-level information, such as the minimum-cost path, in relatively high dynamic networks is questionable, and currently not the focus of our service.

Because the service depends on estimates of the underlying LQE, its accuracy is also influenced by the responsiveness and accuracy of the LQE [2]. The responsiveness influences how quickly changes in the link quality are observed by the LQE, after which it can be disseminated by the service. From our experience we found that the common time windows for averaging link quality introduces a lag and results in a trade-off between the responsiveness and accuracy of the LQE. The accuracy of the LQE is found to be an important and non-trivial aspect in the complete solution of maintaining accurate network metric estimates. Additional experiments are needed to explore this trade-off.

2.4.2 Simulation Results

We implemented both the local broadcasting approach and our generic distributed service as a separate module in OMNeT++ [44], a discrete-event simulation environment, with the use of MiXiM [37], a modeling framework created for static and mobile wireless networks. For the simulations of this section, we instantiate both the local broadcasting approach and our service to derive the cost and parent of the minimum-cost path, as defined by Equation 2.4. For the service, we fixed the validity interval to be two times the update interval. The network metric is estimated every second based on the locally available information. Simulation results in this section show both the accuracy of estimated information and its resulting impact on a minimum-cost routing protocol relying on this information.

We focus on a WSN scenario in which several mobile nodes move through an area covered with static nodes. We consider a network with a hundred static nodes positioned in a grid-like fashion with ten meters between them. There are ten mobile nodes that repeatedly move at walking speed (2 m/s) to a random location in the 100 by 100 meter area and stay at this location for a random amount of time, complying to the Random Waypoint Mobility model [23]. At an interval of one second the mobile nodes send a single packet to the sink, which is located at the left upper side of the static grid of nodes. The transmission power of the static nodes is selected to be -5 dBm, which represents a transmission range of around 12 meter for the used model, allowing the nodes in the grid to be connected to their direct neighbours. The mobile nodes have a lower transmission power of -15 dBm with a range of around 7 meter, which is enough to reach at least one node in the static grid. The fact that static and mobile nodes have different transmission powers results in communication links with asymmetric or even unidirectional quality. Due to the mobility of the nodes, the link qualities and connectivity change over time, resulting in time-varying minimum-cost paths.

The nodes used in the simulated WSN are modeled to have the characteristics of the popular TelosB nodes [68]. The MAC protocol used by the nodes is the non-beacon IEEE 802.15.4 protocol without acknowledgments, as provided in the MiXiM framework. We furthermore implemented a simple minimum-cost routing protocol, which uses the parent on the minimum-cost path for forwarding application packets. The goal of this routing protocol is to minimize the packet-loss, i.e., to reduce the percentage of application packets that do not arrive at the sink. Assuming single-link transmissions are independent, the probability of successful delivery is easily calculated by the product of the probabilities of successful transmission over all links on the path. For the simulations we require an accurate estimator for the cost of individual links. The average observed Bit-Error-Rate (BER) received in the last number of seconds equal to the validity interval is found to result in a good estimation of the cost for the used scenario model. The BER is a function of the receiver Signal-to-Noise Ratio (SNR), which is a hardware LQE integrated in the TelosB CC2420 chip-set.

For a single experiment, every mobile node sends $10,000$ messages to the sink. The nodes start sending packets after an initialization phase, in which nodes initialize their minimum-cost path information. The measured experimental data is collected after the initialization until the nodes have sent all their messages to the sink. To have statistically more reliable results, every experiment was repeated 10 times and shown results are the average over all runs.

By simulating the above scenario, we observe the cost of the minimum-cost path derived with both the local broadcasting approach and the service, and compare it with a simulated oracle that computes the actual minimum cost. With this knowledge, we derive an error for every node by averaging the absolute error

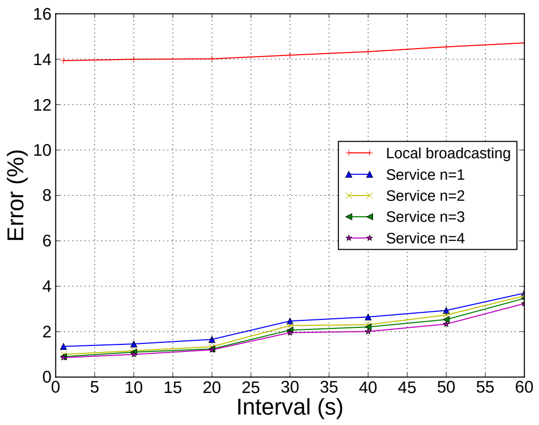


Figure 2.6: Impact of approaches on the error between estimated and actual minimum-cost

between the derived and actual minimum-cost over time. Figure 2.6 shows the average of this error for the ten mobile nodes for different intervals at which the nodes inform neighbouring nodes, i.e., the update interval. We also vary the number of hops used for n-hop forwarding.

The error of the local broadcasting approach is significantly larger than the one for the service, independent of the chosen interval and number of hops. This shows that explicitly considering the presence of asymmetric links, compared to assuming links to be symmetric, has a large impact on the accuracy of the estimates. We also see that reducing the interval at which information is distributed has a positive effect on the error as the maximum time during which wrong information is used is reduced. For the chosen scenario, increasing the number of hops for n-hop forwarding increases the accuracy, but the effect is found to be much less than the increased accuracy of taking asymmetry into account.

The main purpose of the service is to provide accurate information to the protocols that rely on this information. Our routing protocol depends on the parent derived from the knowledge of the minimum-cost path, to minimize the packet-loss. It is expected that when the accuracy of the cost derivation is increased, the packet-loss will reduce as the parent on the path is more accurately chosen. In Figure 2.7, we can see the results of the average end-to-end packet-loss of the ten mobile nodes while varying the parameters of the derivation approaches. As expected from the accuracy results of Figure 2.6, we see that the service results in a much lower packet-loss, independent of the chosen parameters. While the accuracy is invariably increased by reducing the update interval and increasing the number of hops, we see that the packet-loss shows an optimum parameter value, below which the packet-loss increases again. For this deployment, reduc-

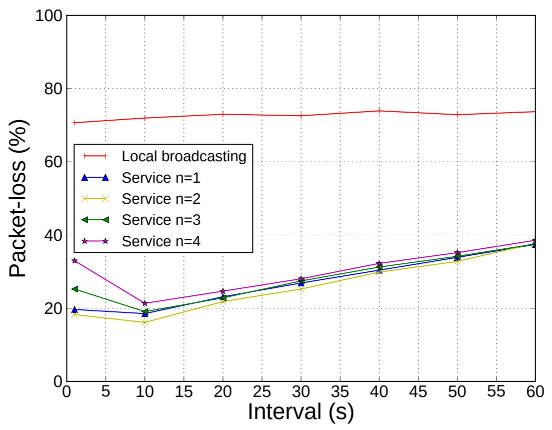


Figure 2.7: Impact of approaches on packet-loss

Table 2.1: Number of packets per update interval

Local broadcasting	*Service* $n = 1$	$n = 2$	$n = 3$	$n = 4$
1	1	4.5	11.8	18.2

ing the update interval to lower than 10 seconds increases the packet-loss even though the error is reduced. The same holds for increasing the number of hops. The packet-loss has a minimum when the number of hops is 2, above which the packet-loss is increased. These effects can be observed better in Figure 2.8, where we plot the average packet-loss versus the number of hops for the ranges of interest. These effects can be explained by the fact that the interference caused by the increased overhead of the service outweighs the benefit of the (here small) increase in accuracy for the simulated scenario.

Both increasing the number of hops and reducing the interval increases the number of packets a node needs to handle. In Table 2.1, we show the average number of packets per update interval for the given scenario. From these numbers, we see that the number of packets is increasing with the number of hops traveled as expected, due to an increased number of neighbours for which information needs to be forwarded.

What we have observed from these simulations is that the service can significantly increase the accuracy of the maintained minimum-cost path and the efficiency of the protocol relying on this information at the expense of an amount of overhead. With (repeated) local broadcasting, accuracy is low and the routing protocol depending on minimum-cost path information lost more than 70% of the

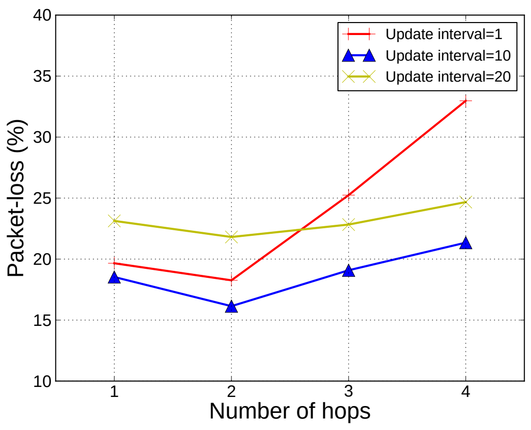


Figure 2.8: Trade-off between accuracy and overhead

packets. We furthermore observed that at some point of the parameter settings, the accuracy of the service may still increase, but the induced overhead reduces the efficiency of the protocol. Depending on the packet-loss allowed by the application, the overhead can easily be reduced by lowering the update interval or number of hops to forward at the cost of only a slight increase in packet-loss. Finally, it is important to note that the amount of overhead as introduced by our service compared to the local broadcasting approach is inevitable when a local broadcasting approach does simply not suffice to provide sufficiently accurate information.

2.4.3 Experiments With Actual Deployments

We implemented both the local broadcasting approach and our generic distributed service as a separate module for TinyOS 2.1 [70]. We discuss the results of integrating our service for two applications using an actual deployment. These applications differ in both the protocol stack used and the network metric needed to be estimated by the service. For the first application, the service is used to derive minimum-cost path information similarly as done for the simulations. The ability of our service to provide accurate estimates to a minimum-cost routing protocol is analysed and compared to the local broadcasting approach. For the second application we take a brief look ahead to the distributed run-time QoS provisioning approach as introduced in Section 3. The service is used to provide several estimates of network metrics allowing nodes to make distributed decision on adapting controllable parameters.

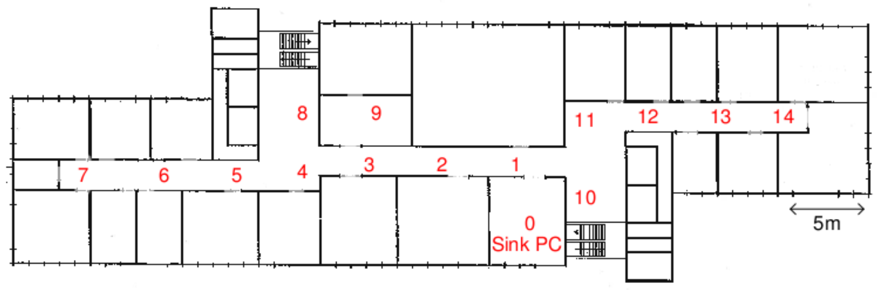


Figure 2.9: Experimental deployment of the 15 static TelosB nodes, monitoring 4 mobile BSN nodes

Minimum-cost routing We experimented with our service and the currently used local broadcasting approach for a dynamic heterogeneous WSN deployment in our office building. Four persons, employees, are equipped with a BSN node [8], which has a similar design as a TelosB node, but has a smaller form-factor and an integrated accelerometer, making them ideal as mobile nodes. These mobile nodes communicate packets with a payload of 32 bytes to a sink PC at a rate of 1 every 5 seconds making use of a static network consisting of 15 TelosB nodes covering the areas of interest for the four monitored employees. We selected the transmission powers of the nodes to be -15 dBm, which allows a reasonable distance between the nodes while it is still low in order to preserve energy spent by the radio. As the antenna characteristics, e.g., size and sensitivity, differ between the BSN and TelosB nodes, the maximum transmission range of a node depends on the type of nodes involved in the communication. The observed maximum transmission range from a TelosB to TelosB node is around 13 meter while being 7 meter from TelosB to BSN. The maximum range of a BSN node communicating to a TelosB node is observed to be around 3 meter. The TelosB nodes are deployed such that the maximum distance between a mobile and static node is at most 2.5 meter in the areas where we monitor the employees, allowing every mobile node to be able to directly communicate to the static network. The resulting deployment set-up is shown in Figure 2.9.

We implemented and used the same protocol stack as used for the simulations, i.e., the IEEE 802.15.4 MAC without acknowledgments and a simple minimum-cost routing protocol. We fixed the validity interval of outbound neighbours to be two times the update interval. Nodes estimate network metric information every second. Again, our goal is to minimize the packet-loss of the packets sent by the mobile nodes. After several small-scale experiments we found the average RSSI value of the packets received in the last interval of time equal to the validity interval to be a good indication of the link quality. It shows a good correlation

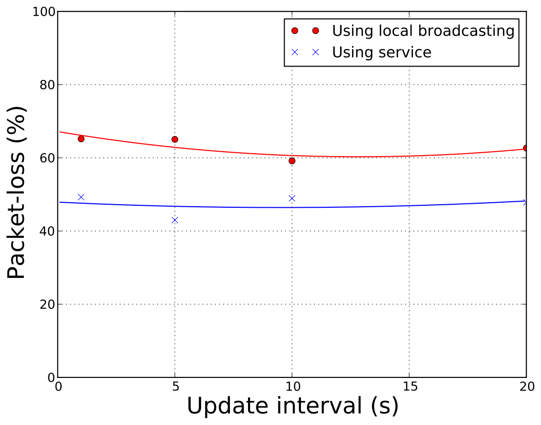


Figure 2.10: Impact of two approaches on packet-loss

with the packet-loss, as also observed in [56].

For a single experiment we let the mobile nodes send packets to the sink for half an hour after. They start after an initialization phase allowing all nodes to initially derive their parent. During the experiment the mobile nodes move through the building based on typical daily activities of the employees, i.e., work in an office, gather at the meeting areas and walk between these areas. We tried to replicate the mobility pattern between experiments as good as possible. After every experiment every node sends packets with statistical information for performance analysis to the sink PC. The number of controlled flooding hops used by the service for the shown results is 1. For the experiments done with a higher number of controlled flooding hops we have seen no significant improvement and even observed slight increase in packet-loss as well. This is because, for the asymmetry as observed in this deployment, we see that one hop is typically sufficient to receive a packet from the parent on the minimum-cost path; multiple hop forwarding gives only a very limited improvement (similarly as observed in the simulations). In some cases, this limited improvement might not be enough to compensate for the increased negative impact of, for example, an increased packet load.

Figure 2.10 shows the resulting average packet-loss observed in our experiments with the two minimum-cost path estimation approaches while varying the update interval. For both approaches the packet-loss is fairly high. This mainly has to do with the chosen protocol stack. We have chosen to focus on a simple protocol stack that lacks robustness to packet-loss as only a single path to the sink is considered and no acknowledgment approach is used (as this should also be adapted to deal with asymmetric links). Even though the packet-losses are high, for the current experiments we clearly see an improvement in the average packet-loss, up to around 20%, when using our service compared to the local broadcast approach. This is because our approach takes link asymmetry into account.

Quality-of-Service provisioning In Chapter 3, our service is used to support a distributed run-time QoS provisioning approach. Estimates of several network metrics are used to locally decide on whether and how controllable parameters should be adapted in order to provide a required network QoS. Experiments are done with the same set-up as used for the minimum-cost routing scenario (Figure 2.9), but a completely different protocol stack is used. Chapter 3 discusses in detail how to instantiate the service for the purpose of QoS provisioning and the resulting efficiency of QoS provisioning. For now we want to take a brief look ahead and mention the main conclusion of the chapter; the QoS provisioning approach allows a more efficient provisioning of QoS compared to typical alternative configuration approaches, by exploiting additional network metric information. Experiments show that the maximum packet-loss in the network is kept at the required level, while a significant improvement of network life-time is achieved. As the additional network metric information is provided by the service, these results illustrate the successful integration of the service in the given protocol stack.

2.5 Related Work

Wireless sensor networks cooperate in a distributed fashion. Local estimates of the performance of other parts of the network is commonly required by (communication) protocols. Estimating network metrics is also referred to as setting up a cost field [79] or gradient field [17, 24, 80]. For the distributed estimation, nodes broadcast information to inform neighbouring nodes, also referred to as 'interests' [24] or 'advertisements' [80].

Directed Diffusion [24] forms the basis of many routing protocols. Sinks repeatedly broadcast packets indicating its interest for data. Nodes receiving interests maintain a list of gradients, where the gradient direction is toward the neighbouring node from which the interest is received. Given a criterion to determine the usefulness, or cost, of a gradient, one or more neighbouring nodes are used to route data to the sink. The Collection Tree Protocol (CTP) [17] uses expected transmissions (ETX) as its routing gradient. A root has an ETX of 0. The ETX of a node is the ETX of its parent plus the ETX of its link to its parent. The ETX value is communicated using periodic sending of control beacons. A node selects its parent based on the lowest ETX received. Gradient Based Routing [54] lets nodes broadcast their hop-count and nodes, in an iterative fashion, determine their hop-count based on the lowest hop-count received from neighbouring nodes.

With these local broadcasting approaches, nodes broadcast their cost and nodes estimate their own cost based on all received cost information. In the presence of communication links with asymmetric quality, caused by heterogeneity in the network, a broadcast might not be sufficient to efficiently communicate required information to neighbouring nodes. Several solutions have been proposed to deal with specific problems related to asymmetric links in particular protocols outside the context of network metric estimation. The majority of protocols ignores them altogether [77]. It is desirable to exploit asymmetric links both for performance and for basic connectivity of wireless networks [53]. To be able to communicate in the presence of asymmetric links, a technique to relay [14] information around these asymmetric links, by using limited forwarding over multiple hops, can be used. As part of our service, we use a controlled version of this n-hop forwarding where already forwarded packets are not forwarded again to avoid extensive packet forwarding.

Dynamism affects the accuracy of the currently collected information from neighbouring nodes. The information might change or the neighbouring nodes might not be present anymore. Repeatedly updating information, and validity management of received information, is needed to ensure nodes use accurate information for their estimation. From the frequency at which updates are done we get a trade-off between the accuracy of the locally available information and the overhead of information communication. Current local broadcasting approaches typically broadcast at a given fixed interval and do not consider an outdating strategy, but only update estimates as soon as information for neighbouring nodes is received. Approaches that use an outdating strategy, such as CTP, do not explicitly consider the trade-off between accuracy and overhead. With our service we explicitly consider this trade-off. Our service uses a repeated update of information at a given interval and explicitly manages the validity of received information, and estimates the network metrics accordingly. By setting the update interval, we adapt the service to the dynamic characteristics of the considered network. We explore the trade-off with extensive simulations and an actual deployment.

Our work differentiates itself from related work by being a generic service that combines n-hop forwarding and explicit information validity management. It can be instantiated for a wide range of metrics to efficiently provide estimates in dynamic heterogeneous WSNs. We furthermore explicitly consider the convergence and stability of our service and explore the trade-off between the accuracy and overhead of network metric estimation.

Alternative to our service are gossiping protocols [20]. In a simple gossip protocol, each node periodically and randomly selects a neighbouring node with whom it exchanges recently observed information. Gossiping can be used for solving consensus problems. Our service can also be instantiated for consensus problems, but is more broadly applicable as it can be used to derive network metric estimates different and dynamically changing for every node in the network, such as the minimum-cost of a path.

2.6 Summary

In this chapter, we have introduced a generic distributed service that allows nodes to accurately estimate network metric information in dynamic heterogeneous WSNs. The service can be instantiated by defining a recursive local update procedure that converges to a fixed point representing the desired metric. We have shown how stability and convergence of instantiations of our service can be shown. The service allows nodes to use controlled n-hop forwarding to communicate information across asymmetric links in heterogeneous networks. To maintain accuracy after dynamic events, the information dissemination and updates of estimates are repeated at a given interval to provide nodes with up-to-date estimates.

The design and flexibility of our service allow it to be easily integrated into existing and new WSN protocols and deployments. We provided insight on the characteristics of a deployment that impact the accuracy of derived minimum-cost path information and the overhead of the service. We furthermore discussed how to set the parameters of the service accordingly. With extensive simulations and experiments with actual deployments, we explored the impact of the parameters on the trade-off between the accuracy and overhead of the service. Simulations and experiments show a significant increase in accuracy of the estimated network metric information and efficiency of the protocol using this information compared to the typically used local broadcasting approach. We showed the feasibility of our service to be integrated in various protocol stacks providing different kinds of network metric estimates.

Chapter 3

Re-active Reconfiguration

By instantiating our generic distributed service, we are able to provide nodes with estimates of the network QoS that is of interest for QoS provisioning. Every node can thereby locally verify whether sufficient QoS is provided by the network. If not, a run-time reconfiguration strategy is needed to restore QoS.

In this chapter, we propose a distributed re-active run-time reconfiguration approach that actively maintains a sufficient QoS at run-time for a heterogeneous WSN in a dynamic environment. Every node uses a feedback control strategy which repeatedly compares the currently estimated and required QoS of the network. To resolve an observed difference, nodes react by adapting controllable parameters of the protocol stack, such as the transmission power and maximum number of packet retransmissions. Models are used, and maintained at run-time, to predict the impact of adapting parameters on the QoS.

In Section 3.1, we formalize the goal of QoS provisioning, present an example to show the need for a reconfiguration approach, and examine the required functionality of a re-active mechanism. It furthermore introduces terminology used throughout this chapter. In Section 3.2, we introduce our distributed run-time re-active reconfiguration approach. Section 3.3 states how we instantiate our generic distributed service allowing nodes to estimate the node and network QoS information needed to support our approach. In Section 3.4, the results of simulations are discussed. Extensive simulations are used to evaluate the accuracy of our re-active reconfiguration approach and give insight in its convergence and stability. Section 3.5 discusses the results of the integration of our approach for an actual monitoring scenario. This shows the feasibility of implementing the approach on resource-constrained nodes. It also shows that, compared to current (re-)configuration approaches, our approach is able to achieve a more efficient QoS provisioning, in this case provide the same packet-loss behaviour with a longer network lifetime. In Section 3.6, we discuss related work. Section 3.7 concludes.

3.1 The Goal of QoS Provisioning

In this section, we discuss the goal of a re-active run-time reconfiguration approach for QoS provisioning in a dynamic heterogeneous WSN. In Section 3.1.1, we formalize the task of QoS provisioning. We introduce the terminology used throughout this chapter in Section 3.1.2. Section 3.1.3 illustrates the desired functionality of the reconfiguration approach to achieve the goal of QoS provisioning using an example.

3.1.1 QoS Provisioning

The WSN should provide sufficient QoS to successfully perform the task specified by the end-user. The performance of individual paths in the network is an important performance aspect in WSN. It influences the properties of its main task; communicating packets over a path to one or more sinks. Constraints on the mostly used network QoS metrics, such as end-to-end latency and end-to-end packet-loss, are path-based. For example, a maximum end-to-end latency of every packet of at most 1 second can be achieved by ensuring that the end-to-end latency of every path is at most 1 second. With our reconfiguration approach we aim at network QoS constraints that are defined in terms of the performance of a path, or can easily be translated to the required path QoS. This enables efficient distributed reconfiguration, as explained in more detail throughout this chapter.

The configuration of the network, i.e., parameter values of the controllable parameters for every node, should establish that the current constraints are met, while the QoS objectives are optimized. Non-conflicting optimization objectives can be optimized simultaneously. When they are conflicting, we assume a cost function is provided to state the relative importance between the conflicting objectives. At least one configuration meeting the constraints is necessary for a reconfiguration approach to be an effective solution for maintaining network QoS. If no such configuration exists, reconfiguration aim at providing the best possible network QoS where the values of constrained metrics are as close as possible to the constraint. Until a configuration is available to provide sufficient network QoS, it cannot be avoided that the network to be (temporarily) not satisfying the expectations of the end-user. If such a situation persists, solutions outside the area of reconfiguration, e.g., adding more nodes or changing the protocol stack, may need to be applied.

If controllable parameters are available that influence only a single constrained metric, finding a configuration meeting the current constraint is fairly straightforward. Values of the parameters can be set to the best possible value without any concern of the other QoS metrics. In practice, we often see trade-offs when adapting parameters. We assume that the configuration of the network involves setting one or more controllable parameters that result in trade-offs between one or more contained metrics and the optimization objective. For example, increasing the transmission power of a node may reduce the latency to communicate a packet to a particular node, but requires more power and thereby reduces the lifetime of the node. Given this trade-off, the configuration should be selected such that a constrained metric, for instance latency to any of the sinks, does not only meet the constraint, but also is close to the constraint to allow freedom to optimize the objective, say maximize the lifetime. In practice, we furthermore see that the QoS metric value of a path, such as the latency to the sink, cannot be matched perfectly with the constraint, due to the discrete nature of the parameters and impact on the QoS metric. We therefore focus on keeping constrained QoS metrics between given lowerbounds and upperbounds, which we assume to be selected such that $lowerbound \leq upperbound \leq constraint$ (in case a lower metric value is better). For QoS metrics where a higher value is considered better, such as delivery ratio, the same approach is taken with the obvious modifications. When the interval between lower and upper bound is small and close to the constraint, small deviations may directly violate the constraint. Infrequent and short violations of the constraint may be tolerated in practice. The *upperbound* can be selected to be lower than the *constraint* to provide a safety margin against violations of the upper bound. However, a large margin reduces the freedom for trade-offs and optimization. For our re-active approach, we assume the required *lowerbound* and *upperbound* to be given. The selection of the lower and upper bounds for a particular deployment, given the constraint, is outside the scope of this chapter.

In summary, our goal of QoS provisioning is defined as follows:

Definition 3. (QoS PROVISIONING) *The goal of QoS provisioning is to maintain the values of all constrained metrics between the respective predefined lowerbounds and upperbounds, while the QoS objectives are optimized.*

3.1.2 Terminology

Figure 3.1 shows a simplified example of a WSN with 9 nodes. The directed edges (both thin and thick) between two nodes represent the links that are actively used to communicate packets, either processed by the node itself or required to be forwarded, towards sink S. Some nodes send packets to multiple neighbouring nodes. We assume a (loop-free) routing protocol to be available to select these parents from all neighbouring nodes within a node's transmission range. Due to the broadcast nature of wireless communication, packets may be overheard by nodes that are not acting as parent, but they are ignored and not forwarded by such nodes. There are multiple *paths* to the sink present in the network.

Definition 4. (PATH) *A path is a sequence of nodes, n_0 to n_k, such that n_i communicates packets to node n_{i+1} for all $0 \leq i < k$.*

For this example, we assume the latency of all packets sent to the single sink (independent of the route taken) as a constrained QoS metric, and maximizing the network lifetime to be the optimization objective. In this example, and the rest of this thesis, we define the network lifetime as the time until the first node

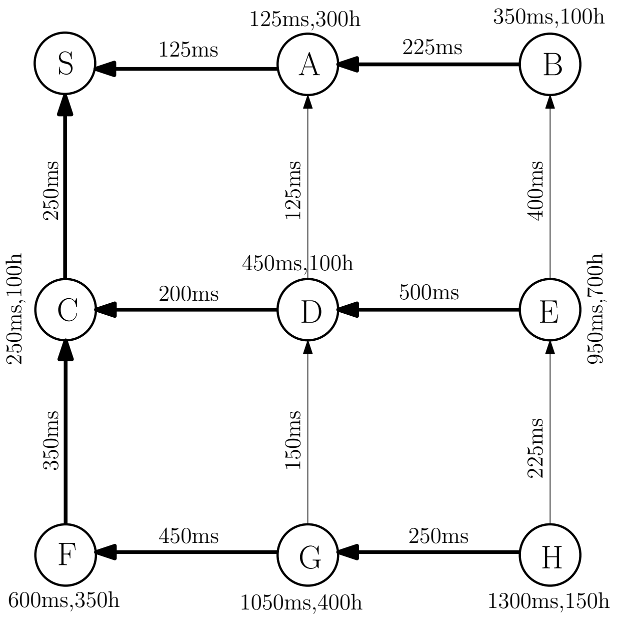


Figure 3.1: Link and critical-path latency and expected remaining lifetime for 9-node sensor network

runs out of battery. Reconfiguration results in a trade-off between these two metrics. The values next to a link show the latency of sending packets over the link. The values next to the nodes show the latency of the node's *critical path* for the latency constraint, the latency-critical path, and the expected remaining lifetime of the node in hours.

Definition 5. (CRITICAL PATH OF NODE n_i FOR A GIVEN QOS CONSTRAINT) *A critical path of node n_i for a given QoS constraint is a path from node n_i with the worst value with respect to the constrained metric, e.g., highest latency to the sink.*

Every node has at least one critical path with respect to each of the constrained metrics. If multiple critical paths exist, one is randomly selected to be the critical path considered in the reconfiguration process. The thicker solid lines, in the example of Figure 3.1, show the links that are part of a latency-critical path. Node D has two neighbouring nodes, A and C, through which it sends packets to the sink. Packets sent through node A have a latency of 250 milliseconds to arrive at the sink, while through C it will cost 450 milliseconds. As a result, its critical path is the path through node C. We want all of the critical paths to comply with the constraint. As the latency can only increase with every additional hop, this is achieved by ensuring that all *end-to-end critical paths* comply with the latency constraint. More specifically, for QoS provisioning we want the latency of these end-to-end critical paths to be maintained between the respective predefined lowerbounds and upperbounds.

Definition 6. (End-to-end critical path for a given QoS constraint) *An end-to-end critical path for a given QoS constraint is a critical path of a node n_i to sink s, where n_i is an end node of the critical paths, i.e., n_i has no inbound neighbours for which n_i is part of their critical path.*

There are three end-to-end critical paths (for the latency constraint) in Figure 3.1 with end nodes B, E and H. Every end-to-end critical path consists of multiple nodes. Adaptation of a node influences its own remaining lifetime and link-latency to neighbouring nodes and thereby the latency of all end-to-end critical path(s) it is a part of. Note that nodes can be on multiple end-to-end critical paths. Node C is on the end-to-end critical paths from nodes E and H to the sink. As the path from node H has the highest end-to-end latency it is the most important path to influence by adapting parameters, and defined to be the *end-to-end critical path of node C*.

Definition 7. (End-to-end critical path of node n_i for a given QoS constraint) *Given a QoS constraint, from all the end-to-end critical paths containing n_i, the end-to-end critical path of node n_i is the one with the worst value for the constrained metric, e.g., highest end-to-end latency to the sink.*

Note that every node has an end-to-end critical path. If multiple paths have the same value, one is randomly selected to be the path currently controlled by our approach. The end-to-end critical path of, for example, node C for the latency constraint starts from H and for nodes A and B, the end-to-end critical path is from B to the sink.

Multiple nodes can share the same end-to-end critical path for a given QoS constraint. These *collaborating nodes* are involved in adapting the same end-to-end critical path and address the same goal of ensuring a satisfied QoS metric constraint for this path.

Definition 8. (Collaborating nodes for a given QoS constraint) *A set of collaborating nodes consists of all nodes which share the same end-to-end critical path for the given QoS constraint.*

For every end-to-end critical path there is a unique set of collaborating nodes. This set may only consist of one node. In our example, nodes A and B form a collaborating set (for the end-to-end critical path from B), as well as D and E (for E), and C, F, G and H (for H).

By the above definitions, the set of collaborating nodes either consist of all nodes of the end-to-end critical path (for the end-to-end critical path from B and H) or forms a subpath starting at the end node of the end-to-end critical path until the node that does not share the same end-to-end critical path (for the end-to-end critical path from E, where node C does not share the same end-to-end critical path with D and E). We additionally introduce the concepts of a *critical parent* and *critical child* of a node. They are used later to distributively identify collaborating nodes.

Definition 9. (CRITICAL PARENT OF NODE n_i) *Let the end-to-end critical path of node n_i consist of nodes n_0 to n_k. If $n_i \neq n_k$, the critical parent of node n_i is n_{i+1}. If $n_i = n_k$ then n_i is the sink node and has no critical parent.*

Definition 10. (CRITICAL CHILD OF NODE n_i) *Let the end-to-end critical path of node n_i consist of nodes n_0 to n_k. If $n_i \neq n_0$, the critical child of node n_i is n_{i-1}. If $n_i = n_0$ then n_i is the end node of the end-to-end critical path and has no critical child.*

The critical parent of C is S and its critical child is F. Node B has no critical child. Every node has at most one single critical parent and critical child. While the critical parent is the parent through which packets are sent resulting in the longest latency, the critical child is not necessarily the child from which the packets with the highest expected end-to-end latency are received. Node E receives packets from node H, but it has no critical child as these received packets are not critical in the latency of H.

3.1.3 Running Example

Figure 3.1 shows the QoS of the network after one or more dynamic events. For this example, we focus on a single trade-off between end-to-end latency and network lifetime and intuitively show the difficulties and required functionality of QoS provisioning in a dynamic heterogeneous network. Assume we want a run-time reconfiguration approach to maintain the latency of all (end-to-end) critical paths between the *lowerbound* of 900 milliseconds and *upperbound* of 1000 milliseconds, while we maximize the network lifetime.

Three situations can cause the goal of QoS provisioning not to be achieved by the current configuration, i.e., one or more paths have (1) a latency higher than the upperbound, (2) a latency lower than the lowerbound and lifetime of the nodes on the path is not optimal, or (3) a latency within range, but the lifetime of the nodes on the path is not optimal. In the example of Figure 3.1 there is an instance of each of the three cases. With the local estimate of the latency of the end-to-end critical path of node i, $e2elat_i$, node i can determine if its end-to-end critical path is in one of these three cases. Each of these cases should be resolved by the combined effort of collaborating nodes adapting their parameters. The desired functionality of a node to establish this, is discussed next. For trade-offs between other QoS metrics, a similar discussion holds.

(1) $e2elat_i > upperbound$. The latency of the end-to-end critical path from node H to the sink is (much) higher than the *upperbound*. The error with respect

to the required latency range is $(1300 - 1000 =)$ 300 milliseconds. Having a latency higher than the *upperbound* is often more harmful than a lower latency, especially if this results in a latency constraint to be violated (think of the health-monitoring example). Therefore, this is the first thing we want a reconfiguration to resolve. Nodes on the path should collectively reduce the latency to solve the error of 300 milliseconds by trading off a reduction in node lifetime (and thereby potentially the network lifetime), for the reduction in end-to-end latency. Any of the collaborating nodes, C, F, G and H, can adapt to achieve this. Since we want to maximize the minimum lifetime over all nodes, it is preferred that nodes with a longer remaining lifetime take a larger share of the trade-off compared to the nodes with a lower lifetime. We, for example, want G to contribute more to reducing the error than node H.

(2) *$e2elat_i <$ lowerbound and lifetime of the nodes on the path is not optimal* The latency of the end-to-end critical path from node B to the sink is (much) below the *lowerbound*. The error is $(350 - 900 =) -550$ milliseconds. This could signal an excessive use of power, with a negative impact on the lifetime. Adaptations where lifetime is increased at the expense of a higher latency may be possible. For the path from node B to the sink, nodes A and B can adapt and relax their latency. It is preferred for the lower lifetime nodes, in this case node B, to get the largest increase in lifetime.

(3) *lowerbound $\leq e2elat_i \leq$ upperbound, but the lifetime in the network is not optimal.* The latency of the path from E to the sink is within the required bounds, but the lifetimes on the path are not balanced. This potentially causes the maximization of the lifetime of individual nodes to be limited. Node E has a significantly longer remaining lifetime than D, and they are collaborating nodes. Node C, however, is not in the same collaborating set. It has more important things to do, namely improve the latency from H, and should not take part in the adaptation of the path from E to the sink. A better balancing between nodes D and E would allow us to increase the lifetime of node D (while keeping the end-to-end latency within the bounds). This could be achieved by node E reducing its lifetime and latency, while node D reconfigures to increase its lifetime and latency.

Note that the adaptation strategy is not based only on the current value of the local latency to the parent. Instead of providing a good latency for every hop independent of how much power (or lifetime) this requires, our strategy lets nodes collaborate in a distributed fashion to reduce the error in latency, while optimizing the lifetime of the entire network. Parameters are adapted based on the effort (i.e., lifetime) already spent on providing a good end-to-end latency compared to others. This results in a balancing of the effort.

The example suggests the network is in a static situation after some dynamic events. In a real dynamic environment, these events continuously occur and adaptations have to be continuously made using estimates of the metrics. In the next section, we detail how our reconfiguration approach is constructed using the strategy explained above.

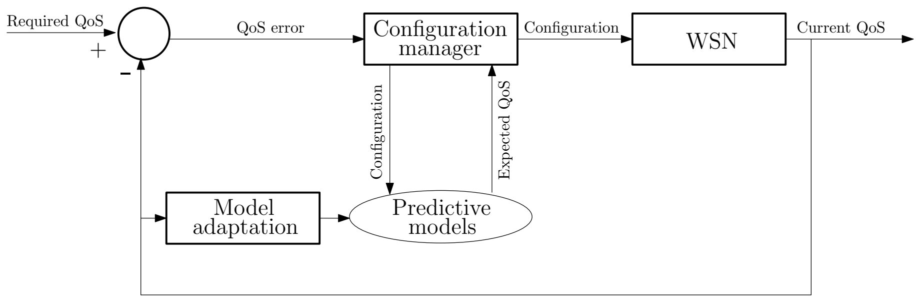


Figure 3.2: Overview of feedback-control-based QoS provisioning approach

3.2 Re-active Reconfiguration for QoS Provisioning

Figure 3.2 shows a conceptual representation of our approach. The goal is to match the current network QoS with the required QoS. The required QoS is in terms of constraints and optimization objectives. The current QoS is defined by the values of all the QoS metrics of interest. Based on the difference between the current and required QoS of the network, a configuration manager determines a new configuration of the network in order to bring the network QoS closed to the required QoS. Using adaptive predictive models, of how different parameters influence QoS, the new configuration is selected by adapting controllable parameters. The new configuration influences the behaviour of the WSN and its provided QoS. The QoS is again compared with the required QoS, closing the feedback loop. Based on observations of the current QoS and QoS before reconfiguration, the predictive models are updated to match the current impact of parameter adaptation. This results in a nested feedback control approach where both the configuration and predictive models are dynamically adapted at run-time.

We want to achieve this (nested) feedback control in a fully decentralized, distributed, manner. For this, we implement the feedback control strategy on every individual node, adapting their own parameters. Adaptations are done such that, together, the adaptations ensure network QoS. In theory, feedback control is a continuous process. In practice, it is realized as a repetitive process. The adaptation of parameters and observing the impact of reconfiguration takes some processing time. Thus, nodes, at a given periodic interval, a *round*, perform an iteration of the reconfiguration approach. As the overhead of comparison and reconfiguration is typically low, we want to set the time between rounds to a low value, for example one second. Note that this sets a maximum frequency of reconfiguration, but does not imply that a node reconfigures every round as this is determined by the actual error and speed, or loop gain, of the controller, as explained later. Furthermore note that every node runs its own instance of feedback control mechanism. In every round, a node decides on how to adapts its

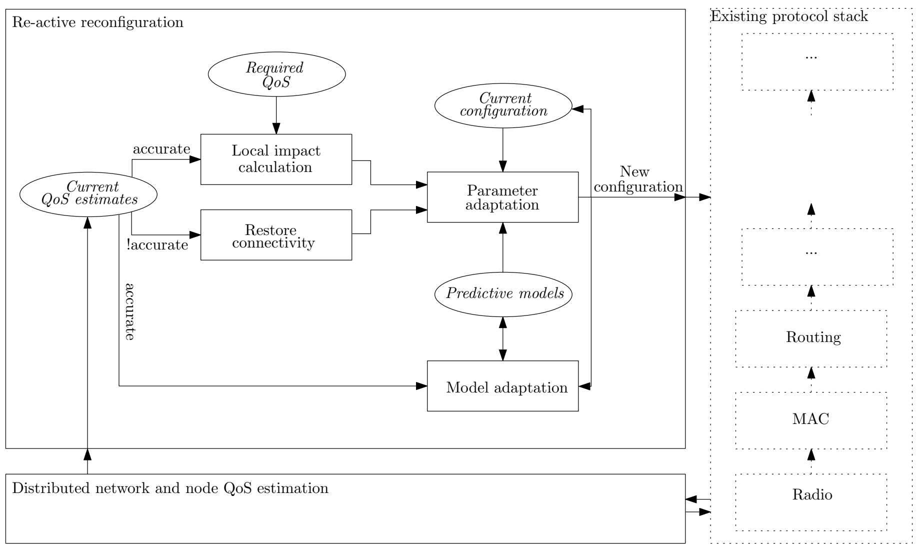


Figure 3.3: Implementation of feedback control mechanism on a node

own parameters based on the locally estimated network QoS error, resulting in a distributed adaptation of the network configuration when needed. Every node has its own notion of a round and the beginning of a round does not need to be synchronized. We do assume that the granularity of adaptation is similar between nodes to avoid that some nodes wait unnecessarily long for other nodes to adapt if possible. For the introduction of our controller, and the analysis done in this chapter, we set the length of a round to the same, low, value for all nodes.

Figure 3.3 shows how the conceptual representation is translated to a practical implementation running on all nodes. As a basis for the reconfiguration approach, our distributed service (Chapter 2) is used to locally estimate the current network QoS. For the discussion in this section we assume a service, instantiated to estimate the required network QoS, to be available. The actual estimation of network QoS, and additionally required information of the node QoS, is discussed in Section 3.3. If the connectivity of the WSN allows for the distribution of information to locally estimate network QoS, the service will provide accurate network QoS estimates to a node. With accurate estimates of the current QoS, in every round, nodes derive how far the current value of every constrained metric is from its intended range. These errors should be resolved by the collaborating set, as explained in the previous section for the trade-off between latency and lifetime. Multiple constrained metrics may be considered at the same time, which all could have a value outside the desired range. This is discussed in Section 3.2.2. How much a given node should exactly contribute to influence every constrained metric is based on the error and the amount of effort spent by collaborating nodes. If a node lacks connectivity to nodes essential for distributed network QoS estimation, e.g., the sink, accurate estimates cannot be determined. Reconfiguration of the node will in this case focus on restoring connectivity to enable re-active reconfiguration. A new configuration to locally influence network QoS is selected using information about the current configuration and the expected impact of changing any of the controllable parameters. Adaptive predictive models that, for every parameter, estimate the expected impact on *local* QoS metrics, e.g., the link latency to the critical parent, which influences related network QoS, e.g., the latency to the sink, are used. These models are discussed in Section 3.2.3. Given the required impact a reconfiguration should have on every constrained metric, and the impact on the metrics given by the predictable model, a new configuration is selected keeping the optimization objective in mind. This is described in more detail in Section 3.2.4. The predictive models are dynamically adapted at run-time to adjust to the dynamics in the network. How this is done is discussed in Section 3.2.5.

3.2.1 Local Impact Calculation

For the ease of explanation, we assume a single constrained metric, end-to-end latency, and focus on how much a node should contribute to resolve an error, given a trade-off with network lifetime. The example of Figure 3.1 is used to support the discussion. The derivation of the local impact on other constrained metrics, or considering different optimization objectives, works is similar way.

We assume that a local estimate of the current latency of the end-to-end critical path is available to node i, $e2elat_i$. Based on this estimate, a node can locally determine if the latency is within the required range. If not, the latency error should be resolved by the combined adaptation of the collaborating nodes. If the latency is within range, the collaborating nodes adapt in order to balance their lifetimes. In Section 3.1.3, we explained which nodes should contribute more to achieve these goals. With the reconfiguration approach being an iterative approach, the local impact is expressed as an adaptation *rate*. The rate defines the (positive or negative) difference in end-to-end latency that a node is expected to achieve in every time unit, e.g., a second or the length of a round, by adapting its parameters. The local rate of adaptation for node i, $rate_i$, is defined as follows.

$$rate_i = \frac{amount_i * frac_i}{k_i}$$

This function states that node i should impact the latency of the end-to-end critical path with a fraction, $frac_i$, of the total amount, $amount_i$. This should be achieved in a given time k_i, defined in the number of time units (or rounds). With parameter k_i, we adjust the overall speed, or loop-gain, of the feedback control used by the node. Reconfiguration cannot achieve a change in latency instantaneously as many aspects, such as the impact of an adaptation on the

QoS and the propagation of the feedback, take time. Therefore, the speed of the controller has an important impact on the behaviour of a feedback control mechanism, such as the stability. An unstable solution, where the latency drifts away from the target latency may occur due to excessive adaptation (for example caused by delayed propagation of latency information). The speed of the controller furthermore plays a role in the trade-off between the accuracy of the controller in providing the required latency and the speed of convergence to the target. This is studied in detail in Sections 3.4 and 3.5.

The values of amount_i and frac_i differ between the nodes in any of the three possible cases, as explained in Section 3.1.1, and are discussed in the remainder of this subsection.

Latency outside required range With the latency of the end-to-end critical path outside the required latency range (cases (1) and (2) of Section 3.1.3), the task of the collaborating nodes is to resolve the latency error. The error observed by node i, error_i, is the difference between the node's own local estimate of the end-to-end critical latency, $\mathit{e2elat}_i$, and the latency range as defined by the *lowerbound* and *upperbound*.

$$\mathit{error}_i = \begin{cases} \mathit{e2elat}_i - \mathit{upperbound} & \text{if} \quad \mathit{e2elat}_i > \mathit{upperbound} \\ \\ \mathit{e2elat}_i - \mathit{lowerbound} & \text{if} \quad \mathit{e2elat}_i < \mathit{lowerbound} \end{cases}$$

To resolve this error, the collaborating nodes have to collaboratively achieve an impact on the end-to-end latency, amount_i, the inverse of this error.

$$\mathit{amount}_i = -\mathit{error}_i$$

The fraction of the total amount of latency, $\mathit{frac}_i, 0 \leq \mathit{frac}_i \leq 1$, that should be taken care of by node i (and thereby also the fraction $(1 - \mathit{frac}_i)$ that is expected to be solved by the other collaborating nodes together) depends on the extent of QoS objective optimization, in this case the remaining lifetime, compared to other collaborating nodes. The fraction is based on several considerations. First of all, we want $\Sigma_{j \in N} \mathit{per}_j = 1$, where N is the set of collaborating nodes. This way, the error is solved by the combined adaptations of the nodes. For the case that $\mathit{e2elat}_i > \mathit{upperbound}$, we furthermore want a higher fraction for a node with a higher expected remaining lifetime. This establishes the balancing of lifetime, thereby optimizing the minimum lifetime in the network. For scalability, packet size, and memory space reasons, we do not want every node to know the lifetime of all individual nodes, but rely on aggregated information. With an estimate of the sum of the lifetimes of the collaborating nodes, $\mathit{sumLife}_i$, we can determine the fraction frac_i by comparing a node's own estimated remaining lifetime, $\mathit{lifetime}_i$ with the sum. This distributed way, not requiring all nodes to know which nodes are in their collaborating set, used to derive all information related to lifetime is discussed in the next section. For the opposite case that $\mathit{e2elat}_i < \mathit{lowerbound}$ we want a higher fraction for a node with a lower expected remaining lifetime. This can be achieved with local estimates of the sum of the inverses of the lifetimes of collaborating nodes, $sumInvLife_i$, and the inverse of the lifetime, $lifetime^{-1}$. For the two cases, $frac_i$ is defined by the following equations.

$$frac_i = \begin{cases} \dfrac{lifetime_i}{sumLife_i} & \text{if} \quad e2elat_i > upperbound \\[2ex] \dfrac{{lifetime_i}^{-1}}{sumInvLife_i} & \text{if} \quad e2elat_i < lowerbound \end{cases}$$

If we look at the path from node H to sink S in the example of Figure 3.1, nodes C, F, G and H are in the same collaborating set and can assist in solving the latency error of 300 milliseconds. Using the equations above we have $per_C = {}^{100}/_{1000}$, $per_F = {}^{350}/_{1000}$, $per_G = {}^{400}/_{1000}$ and $per_H = {}^{150}/_{1000}$. Node G has the longest lifetime and takes the biggest share in solving the error. For the critical path shared by nodes A and B, the fractions are $per_A = {}^{1/300}/_{4/300} = {}^{1}/_{4}$ and $per_B = {}^{3}/_{4}$. Node B has the shortest lifetime and takes the biggest share in the trade-off increasing the latency to increase the lifetime.

Latency within required range As soon as the end-to-end latency is within the required range, i.e., $lowerbound \leq e2elat_i \leq upperbound$, situation (3) of the example of Section 3.1.3, there is no latency error to be solved by the collaborating nodes. Instead, a rate is calculated with the goal of balancing the lifetimes of collaborating nodes. Nodes with a larger remaining lifetime can reduce the end-to-end latency, allowing nodes with a shorter lifetime to improve their lifetime by increasing the latency. To reduce the probability of the end-to-end latency to get outside the target range, the total amount of latency increased should be in line with the amount reduced. We want $\Sigma_{j \in N} per_j = 0$ to avoid a strong fluctuation in end-to-end latency. The latter is achieved by determining the fraction, $frac_i, -1 \leq frac_i \leq 1$, as described below, by subtracting the fraction a node would take in reducing the latency from the fraction it would take in increasing the latency.

$$frac_i = \frac{{lifetime_i}^{-1}}{sumInvLife_i} - \frac{lifetime_i}{sumLife_i}$$

Depending on the predominant factor, the fraction is either positive, indicating that the node wants to increase the latency to increase lifetime, or negative, indicating a reduction. If the lifetimes of all nodes are equal, while the latency in within the suitable range, the fraction is 0 for every node.

For solving a latency error, every collaborating node had the same goal of either reducing or increasing the latency. With balancing, collaborating nodes can have different goals, i.e., a different assumption on what $amount_i$ should be. $amount_i$ should not be too large in this case to avoid strong fluctuations in end-to-end latency due to the discreet and distributed adaptations. We achieve this by the following definition of $amount_i$, where the amount to balance depends on the freedom available between the current latency, $e2elat_i$, and the defined bounds, *lowerbound* and *upperbound*. The used bound depends on whether latency is reduced or increased by the node.

$$amount_i = \begin{cases} upperbound - e2elat_i & \text{if} \quad frac_i > 0 \\ \\ e2elat_i - lowerbound & \text{if} \quad frac_i < 0 \end{cases}$$

Collaborating nodes D and E in the example of Figure 3.1 are on an end-to-end critical path with a latency in the required range, but their lifetimes are unbalanced. The positive fraction for node D, $per_D = {}^6/_8$, indicates that this lowest lifetime node can increase its latency to increase lifetime, while long-lifetime node E, with a fraction $per_E = -{}^6/_8$, reduces its latency.

The rate is expressed in terms of the required impact on the network QoS, i.e., latency of the end-to-end critical path. By adapting parameters, nodes can only impact local QoS, i.e., the latency to the critical parent. Therefore, the adaptation rate is translated to the local impact that is required on the link latency to achieve the desired change in end-to-end latency. In general, a function f is applied to the calculated rate, $rate_i$, to get the local rate, $localrate_i$:

$$localrate_i = f(rate_i)$$

For the end-to-end latency metric this translation is straightforward as the change in the local link latency to the parent is directly proportional to the change to the network latency, i.e., f is the identity function. For other parameters a less trivial translation might be required. For example, a reduction of the delivery ratio to the parent of 10% does not result in a reduction of 10% for the entire path to the sink, unless the remainder of the path has a delivery ratio of 100%. The global rate has to be multiplied with the packet loss on the remainder of the path, *remainderPathLoss*, to get the local rate that is needed to achieve the global rate, i.e., $localrate_i = remainderPathLoss * rate_i$.

3.2.2 Restore Connectivity

A reconfiguration or dynamic event, such as an increased distance between nodes, can result in nodes losing access to the main network. As a consequence, besides not being able to provide application data packets to the sink, no network QoS can be estimated and no local adaptation rates can be calculated by all of the nodes on the unconnected path. To avoid nodes to be in this situation for significant amounts of time, fallback reconfiguration is needed to ensure that nodes can restore connectivity. After being reconnected, network QoS estimates and adaptation rates can be calculated again.

Restoring the connectivity is activated by a node as soon as it is observed that the estimates of the network QoS, as provided using the approach discussed in Section 3.3, are outdated or non-existing. After activation, the configuration selection procedure is, instead of being provided with a rate, requested to adapt parameters which have an expected positive impact on the connectivity of a node. The most obvious one is the transmission power. If more nodes are part of an unconnected path, all of them will try to get reconnected. After being reconnected, network QoS and adaptation rate are calculated again, also allowing to revert any unnecessary reconfigurations.

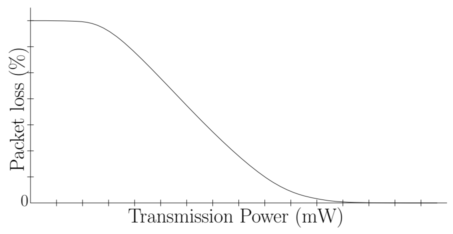
3.2.3 Predictive Model

A set of controllable parameters is available to the node to change its local QoS, e.g., latency to the parent and consequently the network QoS, e.g., end-to-end latency. For a reconfiguration to select appropriate new values for parameters, it needs a model to predict the expected impact of parameter adaptation. The advantage of the feedback control strategy in our reconfiguration approach is that it does not require perfectly accurate models as long as it is known whether a particular change will increase or decrease a particular QoS metric. If the impact is higher or lower than expected this will be observed in the feedback and the controller will continue to adapt the latency to the target range. On the other hand, accurate knowledge of the impact can prevent many unnecessary reconfigurations needed to recover for adaptations that have a significantly different impact than expected. Therefore, it is worthwhile to model as accurate as possible, while being simple enough to be easily stored on the sensor nodes and adapted at run-time.

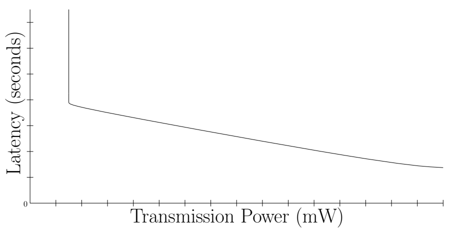
To investigate how to efficiently model the adaptation impact, we start by looking at typical impact characteristics. Figure 3.4 sketches the impact of changing the radio transmission power parameter of a node on the average packet-loss and latency to the critical parent, and expected network lifetime. An increasing transmission power reduces the loss as soon as the power is high enough to be connected and until the point that packet-loss is minimal. Adapting the value after this point will not reduce the packet-loss, but does reduce the lifetime, which is of no interest to a reconfiguration approach. A similar thing holds for the latency metric. We refer to the interval in which a change of parameter shows a significant impact on a trade-off as the *suitable range*. Defining the significance of the impact depends on the considered metric.

Definition 11. (SUITABLE RANGE) *The suitable range of a parameter, with respect to a given QoS metric, is the interval between two parameter values in which parameter adaptation has a significant impact on a trade-off.*

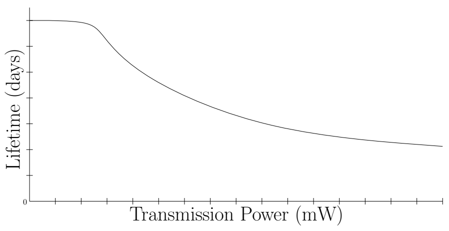
Currently, run-time reconfiguration approaches typically rely on simplistic static impact models, where every adaptation step of a parameter is assumed to have the same impact on the metrics [59]. For such a static model, only a single value needs to be stored for every parameter-metric pair and no run-time overhead is involved with maintaining this model. The main downside is its lack



(a)



(b)



(c)

Figure 3.4: Impact of transmission power on (a) packet-loss, (b) latency and (c) lifetime

of accuracy in dynamic networks. As impact of changing parameters strongly depends on changing run-time properties of the network, such as the topology and the configuration of other nodes, the suitable range can shift and become smaller or larger over time.

We propose an adaptive model for every parameter-metric pair to model the suitable range. The suitable range is approximated by the values, x_{min} and x_{max}, with $x_{min} \leq x_{max}$. The value of the QoS metric corresponding to x_{min} is y_{min} and for x_{max} is y_{max}. As a result, the model is represented by two points, (x_{min}, y_{min}) and (x_{max}, y_{max}), as shown in Figure 3.5 for a typical packet-loss trend. Note that, with a decreasing trend, y_{min} is larger than y_{max}. Inside the suitable parameter range we make a linear approximation of the impact on the QoS metric.

Given these models, solving simple linear equations results in the extraction of the expected QoS given a parameter value, and vice versa. Given the model parameters for a parameter-metric pair, the following formula extracts the predicted value of the QoS metric that will be achieved given a parameter value, *parval*.

$$qos(parval) = y_{min} + (parval - x_{min}) * ((y_{max} - y_{min})/(x_{max} - x_{min})) \qquad (3.1)$$

The formula assumes that $x_{min} \leq parval \leq x_{max}$. The behaviour outside the suitable range of parameter values depends on the considered metric; it could remain the same as for the packet-loss and lifetime, jump to an extreme value as the latency or potentially even invert the trend. The model does not give QoS

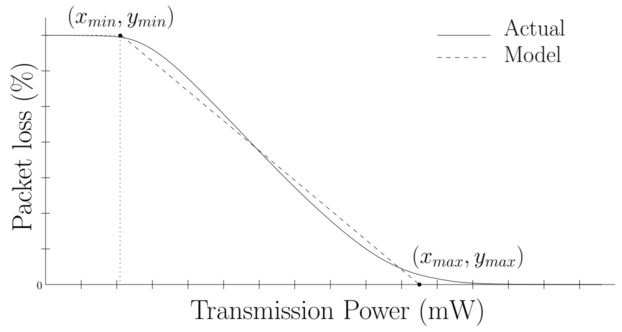


Figure 3.5: Describing adaptation impact on QoS with four parameters

values outside the range, as, by definition, parameter values are expected not to influence any trade-off with other QoS metrics.

Similarly, the following equation extracts the parameter value that is expected to achieve a given value of the QoS metric, *qosval*. It assume that $y_{min} \leq qosval \leq y_{max}$ or $y_{max} \leq qosval \leq y_{min}$, depending on the trend. Achieving a value outside this range is predicted not to be possible with (only) adapting the current parameter.

$$par(qosval) = x_{min} + (qosval - y_{min})/((y_{max} - y_{min})/(x_{max} - x_{min})) \qquad (3.2)$$

In the next subsection, we show how to use the predictive models to select a new configuration, assuming these models are available for every parameter-metric pair. How to initially set and dynamically adapt the model to maintain its accuracy is discussed in Section 3.2.5.

3.2.4 Parameter Adaptation

To achieve the calculated adaptation rate for every constrained metric, parameters are adapted using the predictive models. Instead of directly focusing on finding a set of parameters that achieves the rates for all metrics, we find parameters for every individual metric. The combined adaptations for individual metrics ensures that all metrics are adapted according to their specified rates. In theory, conflicting adaptation options may be suggested in the same round when using this approach, i.e., to resolve the error of one metric, increasing the value of a parameter is suggested, while to resolve the error of another metric, reducing the value of the same parameter is suggested. Given that a suitable configuration exists, another parameter should be available to improve at least one of the two metrics, such that both can converge to their required value range. To avoid that a conflicting adaptation persists over time, nodes could occasionally select the second best adaptation option. An interesting direction for future work is to explore

Table 3.1: Global variables for parameter adaptation

Name	*Explanation*
$curValue_p$	Current value of controllable parameter p
bestParId	Identifier of the (current) best parameter
bestParValue	Best value of parameter *bestParId*
bestValueIn	Whether *bestParValue* is within the suitable range as defined by the predictive model
bestImpactLat	Impact on latency of adapting parameter *bestParId* to *bestParValue*
bestImpactLife	Impact on lifetime of adapting parameter *bestParId* to *bestParValue*

Algorithm 3: Parameter adaptation for node i

```
1  $\Delta Lat := localrate_i$;
2  $rateAchieved := false$;
3  while $\neg rateAchieved$ do
4      for all controllable parameters $p$ do
5          $(value, inside, lat, life) :=$ getBestValueForPar($p, \Delta Lat$);
6          checkWhetherBestPar($p, value, inside, lat, life$);
7      end
8      adaptPar($\Delta Lat$);
9      if $(|\Delta Lat| > |bestImpactLat|)$ then
10         $\Delta Lat := \Delta Lat - bestImpactLat$;
11     else $rateAchieved := true$;
12 end
```

the trade-offs involved with these potential conflicting adaptations performed by the individual maintenance of metrics.

In this subsection, we describe how to select one or more parameters to achieve a given latency rate, $localrate_i$, calculated according to the strategy of Section 3.2.1. The selection strategy is similar for any other metric and optimization objective. Selecting the parameter(s) to adapt in this round in order to achieve a calculated rate consists of three steps. First, the 'best' possible value to for every parameter to achieve the rate is determined. From these adaptation alternatives the 'best' parameter is selected. What defines the 'best' options is explained throughout this subsection. Finally, the selected parameter is adapted such that the adaptation rate is achieved. If adapting a single parameter is not sufficient for achieving the adaptation rate, the steps are repeated.

Table 3.1 states the global parameters used by the pseudo-code descriptions

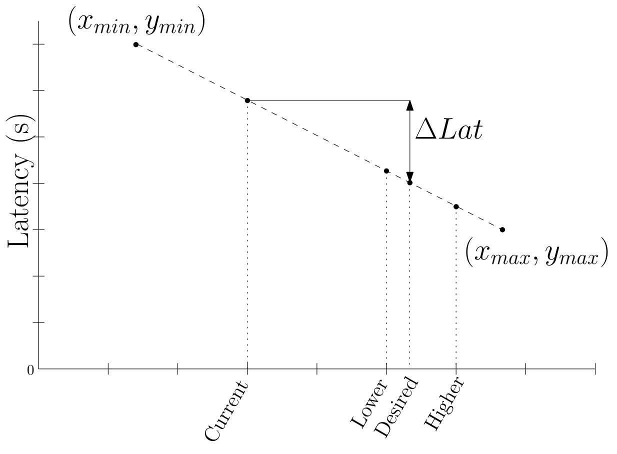


Figure 3.6: Selecting a new parameter value from predictive model

of the three parts. Pseudo-code for the overall adaptation strategy is shown in Algorithm 3. Before determining the parameters to adapt, we define the change in latency, $\mathit{\Delta Lat}$, to be achieved this round. Assuming the rate is defined per round, this change is equal to the (absolute) local rate, $localrate_i$, (line 1). As long as the node is still required to adapt one or more parameters to achieve the (remaining) change in latency, indicated by *rateAchieved* (lines 2 and 3), the best possible parameter is selected from all controllable parameters (line 4). The function getBestValueForParameter (line 5, later explained using the pseudo-code of Algorithm 4) determines to which *value* to adapt parameter *p* given the required change in latency. It furthermore determines the characteristics of the proposed adaptation, i.e., whether the new value is within the suitable range (given by Boolean value *inside*) and the predicted impact on latency, *lat*, and lifetime, *life*. This is needed by checkWhetherBestParameter (line 6, Algorithm 5) which checks whether the current evaluated parameter is the best option so far. Due to discrete adaptation of parameters, the change of latency achieved by adapting the best parameter can either be higher or lower than the remaining required change. A probabilistic adaptation approach, as defined by adaptParameter (line 8, Algorithm 6), is used to determine whether the best parameter should be adapted in this round. Finally, a check whether the parameter adaptation(s) so far are sufficient to achieve the rate is performed by checking the expected impact of the latest adaptation with the remaining required impact on latency (line 9). If the remaining rate is smaller, we repeat the selection of a parameter to adapt for the remaining latency (line 10). Else, the reconfiguration of the node is stopped (line 11).

Algorithm 4: getBestValueForParameter(p, ΔLat)

```
1  $latency$ := getCurrentLatency($curValue_p$);
2  $desired$ := getExactParValue($latency + \Delta Lat$);
3  $lower, higher$ := getDiscreteParValues($desired$);
4  $x_{min}, y_{min}, x_{max}, y_{max}$ := getParsOfPredModel($p, latency$);
5
6  $value := unknown, lowerPossible := false, higherPossible := false$;
7  if $((y_{min} < y_{max} \wedge \Delta Lat > 0) \vee (y_{min} > y_{max} \wedge \Delta Lat < 0))$ then
8      if $(lower \neq unknown \wedge lower \neq curValue_p)$ then $lowerPossible := true$;
9      if $(higher \neq unknown)$ then $higherPossible := true$;
10     if $(lowerPossible)$ then $value := lower$;
11     else if $(higherPossible)$ then $value := higher$;
12 else
13     if $(higher \neq unknown \wedge higher \neq curValue_p)$ then
       $higherPossible := true$;
14     if $(lower \neq unknown)$ then $lowerPossible := true$;
15     if $(higherPossible)$ then $value := higher$;
16     else if $(lowerPossible)$ then $value := lower$;
17
18 if $(value \neq unknown)$ then
19     $inside := x_{min} \leq value \leq x_{max}$;
20     $lat$ := getLatencyImpactFromModel($value, curValue_p$);
21     $life$ := getLifetimeImpactFromModel($value, curValue_p$);
22 return $(value, inside, lat, life)$
```

Selecting the best value for a parameter The pseudo-code for the selection of the best value of parameter p given a required change in latency, ΔLat, is shown in Algorithm 4. We illustrate the code using a particular instance of parameter p where the parameter has a decreasing impact trend for the latency metric, $y_{min} > y_{max}$, shown in Figure 3.6. For this example, we furthermore assume $\Delta Lat < 0$. We start the derivation of the best value by initializing required variables. Given the current parameter value, the predicted current latency value is provided by getCurrentLatency (using Equation 3.1) (line 1). Similarly, the exact desired parameter value to achieve ΔLat is provided by getExactParValue, using Equation 3.2, (line 2). Parameter values can typically not be selected from a continuous space, but are discrete, such as the number of retransmissions. This leaves two options for the selection of the best new parameter value, i.e., lower or higher than desired (see Figure 3.6). Note that the selected value for adaptation can potentially be outside the suitable range. Given the definition of the suitable range this might be an adaptation we want to avoid. On the other hand, it is important to keep in mind that a predictive model is just an approximation and different impact on the QoS than predicted might be observed in practice, especially in the presence of dynamics. While we initially want to focus on adaptation within the suitable range, selecting outside the range should not be prohibited. It is required to explore the boundaries of the modeled suitable range in the presence of dynamic events, as explained in more detail in Section 3.2.5. The lower and higher value are assumed to be provided by getDiscreteParValues based on knowledge of the available discrete adaptation options (line 3). If any of the values does not exist, for example because the desired value is already higher than the highest possible option, the value is considered to be unknown, referred to by a special value *unknown*. We furthermore use the parameters of the predictive model that describes the impact of parameter p on the latency, assumed to be proved by function getParsOfPredModel (line 4). The best adaptation option is selected from the lower and higher value, starting by checking whether the options are viable. Initially, both values are not possible and there is no best adaptation option (line 6). Selecting either the lower and higher value depends on the impact trend of the predictive model and the change in latency to achieve. For our example, $y_{min} > y_{max}$ and $\Delta Lat < 0$ (line 7), the desired value is higher than the current parameter value. The lower value is a possible adaptation option if it exists ($\neq$ *unknown*) and is not equal to the current parameter value (line 8). The higher value is possible if it exists (line 9). No check on equality with the current value is needed as the higher value is either higher than the current value (potentially outside the suitable range) or not existing. If possible, the lower value is selected as the best option (line 10). Using this conservative approach we first adapt parameters within the suitable range with at most the required change in latency. Any remaining rate will be achieved by repeating the selection of the best parameter value. If the lower value is not possible, the higher value is selected if possible (line 11). The above discussion also holds for the symmetric case that $y_{min} < y_{max}$ and $\Delta Lat > 0$ (added to line 7). For the case that the newly selected value is lower than the current parameter, things are similar, but the role of the lower and higher value interchange (lines 12-16). If an adaptation option is available (line 18), its characteristics are determined. It is checked whether the value is within the suitable range (line 19). Furthermore, the predicted change in latency (line 20) and lifetime (line 21) is determined using the predictive models and Equation 3.1. To allow to compare two adaptation options that are outside the suitable range, it is assumed that the model trend continues outside the suitable range allowing to calculate the expected impact. If both the lower and higher value are not possible, the returned best value is *unknown* (and its characteristics are not defined) and the parameter will not be considered in the rest of the adaptation process.

The above discussion implicitly assumed that ΔLat can be achieved within the current suitable range of the considered parameter. If not, the desired value (calculated in line 2) is considered to be the closest value within the suitable range (leaving all remaining rate to be achieved by adapting other parameters). Furthermore, we have to consider that the current value of the parameter is not

Algorithm 5: checkWhetherBestParameter(p, *value*, *inside*, *lat*, *life*)

```
betterOption := false;
if (bestParId = unknown) then betterOption := true;
else if ($\neg$bestValueIn $\wedge$ inside) then betterOption := true;
else if (bestValueIn $\vee$ $\neg$inside)) then
    adaptRatio := lat/life;
    bestRatio := bestImpactLat/bestImpactLife;
    if ($\Delta Lat < 0 \wedge$ adaptRatio > bestRatio) then betterOption := true;
    if ($\Delta Lat > 0 \wedge$ adaptRatio < bestRatio) then betterOption := true;

if (betterOption) then
    bestParId := i, bestParValue := value;
    bestValueIn := inside, bestImpactLat := lat, bestImpactLife := life;
```

within the suitable range at all. The parameter value could previously be selected outside the suitable range and the range might not be adapted to contain this value, as discussed in Section 3.2.5. By the definition of the suitable range, no trade-off is expected by adapting a value outside the suitable range to the closest boundary value of the suitable range, i.e., x_{min} or x_{max}. Given this, we set the current value (as used in line 1) equal to this closest boundary to be able to determine an impact on the latency with the model and derive the 'best' value of the parameter using the approach described above.

Determining the best parameter to adapt The pseudo-code to check whether the current option is the best so far is shown in Algorithm 5. The current best parameter value is assumed to be stored in the global parameter *bestParId* and is *unknown* if non existing. Initially, it is assumed that the current option is not better (line 1). If no best option currently exist, then the first option given is considered to be the best (line 2). With a best adaptation option, we first prefer adaptations within the suitable parameter range. If the current best adaptation option suggests to adapt the value outside the suitable range, defined by global variable *bestValueIn*, while the value for i is adapted within range, adapting i is considered to be best (line 3). If both the current best and suggested adaptation option are outside or inside the parameter range (line 4), we determine the best option based on the ratio between the expected impact on latency and lifetime of the current parameter (line 5) and currently known best parameter (line 6). This ratio defines how much lifetime is needed to achieve a change in latency. If the required adaptation rate is negative, the best parameter to be adapted should have the lowest negative impact on lifetime to achieve this reduction of latency. In other words, higher impact ratios are better (line 7). For a positive rate, we want the highest possible increase in lifetime, i.e., the lowest impact ratio (line 8). If the newly suggested

Algorithm 6: adaptPar(ΔLat)

```
1 if $(|bestImpactLat| \leq |\Delta Lat|)$ then
2     Adapt($bestParId$, $bestParValue$);
3 else
4     $prob := \Delta Lat / \Delta bestImpactLat$;
5     if $(randomFloat(0, 1) < prob)$ then
6         Adapt($bestParId$, $bestParValue$);
```

adaptation option is found to be better, this is stored (lines 10, 11 and 12) for future comparison.

(Probabilistic) Adaptation of best parameter Algorithm 6 shows the steps involved in determining if the best parameter should be adapted in this round. If the predicted impact is less than the required impact, the parameter is directly adapted by informing the protocol stack (lines 1 and 2). We assume function Adapt to be an interface to the protocol stack to adapt controllable parameters. In the case that predicted impact is more than required, a probabilistic approach is used. The probability of adaptation is equal to the ratio of the desired step and the discrete step (line 4). For example, with a required adaptation rate of 10 milliseconds per round and an expected impact of adaptation of 100 milliseconds the probability of adaptation is set to 10%. With the determined probability the parameter is adapted (lines 5 and 6). In practice, we see that the required change in latency in a single round is typically low and a probabilistic adaptation is applied.

Besides to achieve a calculated rate, reconfiguration might be needed to restore connectivity. If the restoration of connectivity is triggered, as explained in Section 3.2.2, parameters are adapted that have an expected positive impact on the connectivity. This might either be fixed if known at design-time or based on the current predictive models. In the extreme case that adapting parameters is not possible anymore and a node is still unconnected to the main network, duty cycling between the most power efficient and most power hungry configuration (with higher probability of finding outbound neighbours) is performed to avoid an extensive drain of the battery. Even though situations like this may always occur in practice, they should be avoided as much as possible by design of the WSN, e.g., sufficient density of nodes.

3.2.5 Model Maintenance

We require initial predictive models to be available at the start of WSN operation. For example, simulations can be used to identify an initial model. Our strategy for the run-time model maintenance is visualized in Figure 3.7, where the horizontal time-line is divided into rounds. After a reconfiguration of the node, the predictive model is not adapted until the impact of reconfiguration is reflected in the local

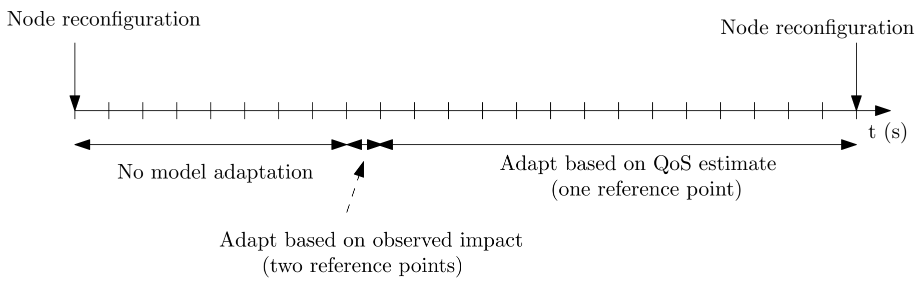
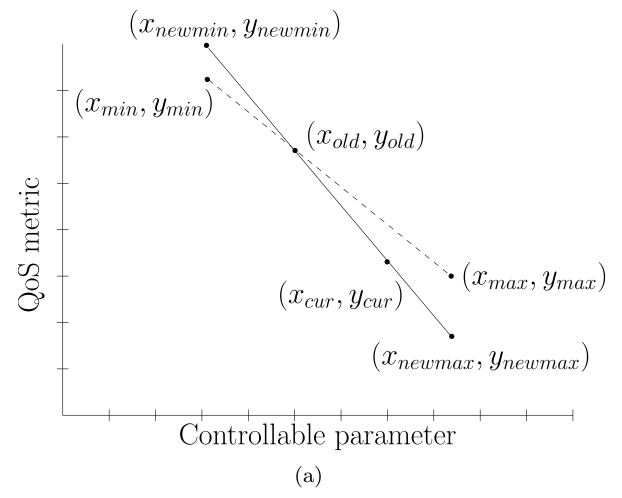

Node reconfiguration
Node reconfiguration
t (s)
No model adaptation
Adapt based on QoS estimate (one reference point)
Adapt based on observed impact (two reference points)

Figure 3.7: Model maintenance strategy over time for a given node

estimated network QoS. This takes a particular amount of rounds, depending on time needed for the propagation of network QoS estimations and the potential averaging of QoS values. If latency is averaged over the last minute, at least a minute is needed to see the full effect of a reconfiguration. After this, the model is adapted based on the two available reference points; the QoS for the value before reconfiguration and the QoS for the value after reconfiguration (as explained later in more detail). Parameter adaptations may be infrequent, but allows to update the model using these two reference points. Afterward, only the local estimates of QoS metrics for the current parameter value are used to update the predictions of the model. In this phase, we have only one reference point for model adaption, but it can be performed whenever accurate estimates are available.

Adapting the model based on observed impact of reconfiguration Figure 3.8(a) visualizes the adaptation of the model based on two reference points; the observed QoS, y_{old}, for the previous parameter value, x_{old}, and the current QoS value, y_{cur}, for the current parameter value, x_{cur}. The new model parameter values are such that the two points are predicted by the model. Algorithm 7 shows the pseudo-code for this step. To save space, only the adaptation for a model with a decreasing trend, $y_{min} > y_{max}$, is discussed. For the symmetric case, where $y_{min} < y_{max}$, the same approach is applied, but the role of min and max interchange. The new model parameters are initially set to the current values (line 1). The first check is whether the previous value, x_{old}, is within the suitable range (line 2). If not, the model is adapted based on only the observed QoS for the current parameter value, x_{cur}, because no latency predictions can be made for x_{old} (line 3, calling the function in Algorithm 8). The following statements (lines 4 and 6) check whether the observed impact is in conflict with the expected trend (in this case decreasing). The trend is assumed to continue if there is at least a predefined minimum impact observed, *minDif*, after reconfiguration. If not, there is no continuation of the trend and the suitable range is reduced, if needed. The suitable range is reduced to exclude the current parameter value by adapting x_{max} or x_{min}, respectively according to the value of the current parameter value with respect to the



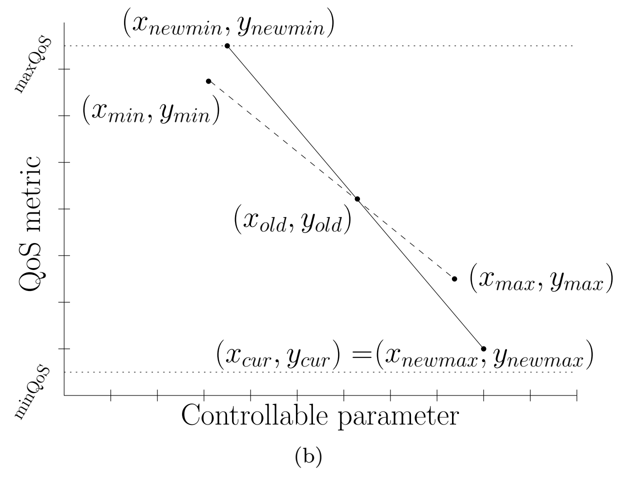


Figure 3.8: Model before (dashed line) and after (solid line) adaptation. (a) Base case. (b) Current value outside suitable range and considering the minimum and maximum achievable QoS

Algorithm 7: adaptForImpact(), for $y_{min} > y_{max}$ (decreasing trend)

```
1  $x_{newmin} := x_{min}, y_{newmin} := y_{min}, x_{newmax} := x_{max}, y_{newmax} := y_{max}$;
2  if $(x_{old} < x_{min} \vee x_{old} > x_{max})$ then
3      adaptForCurrentQoS();
4  else if $(x_{old} < x_{cur} \wedge y_{old} - y_{cur} < minDif \wedge x_{cur} < x_{max})$ then
5      $x_{newmax} := x_{cur}$;
6  else if $(x_{old} > x_{cur} \wedge y_{cur} - y_{old} < minDif \wedge x_{cur} > x_{min})$ then
7      $x_{newmin} := x_{cur}$;
8  else
9      $s := (y_{cur} - y_{old})/(x_{cur} - x_{old})$;
10     $y_{newmin} := y_{old} + (x_{newmin} - x_{old}) * s$;
11     $y_{newmax} := y_{old} + (x_{newmax} - x_{old}) * s$;
12     if $(y_{newmin} > maxQoS)$ then
13         $x_{newmin} := maxQoS/s - y_{old}/s$;
14         $y_{newmin} := y_{old} + (x_{newmin} - x_{old}) * s$;
15     if $(y_{newmax} < minQoS)$ then
16         $x_{newmax} := minQoS/s - y_{old}/s$;
17         $y_{newmax} := y_{old} + (x_{newmax} - x_{old}) * s$;
18     if $(x_{cur} < x_{min})$ then
19         $x_{newmin} := x_{cur}$;
20         $y_{newmin} := y_{cur}$;
21     if $(x_{max} < x_{cur})$ then
22         $x_{newmax} := x_{cur}$;
23         $y_{newmax} := y_{cur}$;
24 AdaptModel($x_{newmin}, y_{newmin}, x_{newmax}, y_{newmax}$);
```

old one (lines 5 and 7). If the trend continues, the model is adapted such that both reference points are predicted by the model, by adapting y_{min} and y_{max}. With the adaptation of the model, we can additionally take into account (trivial) bounds, *minQoS* and *maxQoS*, on the achievable QoS, such as 0% and 100% for the packet-loss metric. We limit the extrapolation of predicted latency, as shown in Figure 3.8(b), for x_{newmin} and y_{newmin} (lines 12 until 17). If the current parameter value is outside the suitable range, while the impact trend is observed to continue, the range is extended to include the new parameter value (lines 18 until 23). This is shown in Figure 3.8(b), for x_{newmax} and y_{newmax}. Finally, the model is adapted based on the newly calculated parameter values (line 24). This function can either directly adapt the model to the new parameter values or take a stepwise approach where the model is adapted with a fraction of the deviation between the current and new model. The latter avoids that a single observed extreme QoS for the current parameter value strongly influences the model, as multiple observations are needed to adapt the model.

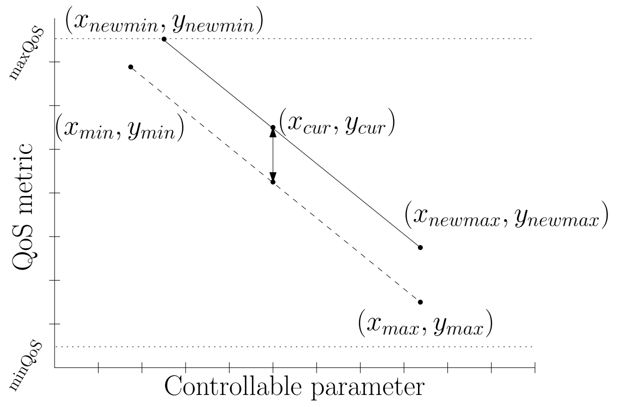


Figure 3.9: Adapting model parameters using current parameter value

Adapting the model based on current QoS Figure 3.9 visualizes how to adapt the predictive model using only the observed latency, y_{cur} for parameter value x_{cur}. The model is translated in the vertical direction, maintaining the slope of the impact trend. Algorithm 8 shows the pseudo-code. Again, the adaptation for a model with a decreasing trend, $y_{min} > y_{max}$, is discussed. Initially, the new model parameters are equal to the old model (line 1). The model is only adapted if the current parameter value is within the suitable range (line 2). By extracting the predicted latency for x_{cur} from the current predictive model (line 3, using Equation 3.1), we can determine an error with the actual observed latency (line 4). This error cannot be determined with a parameter value outside the suitable range. The model is adapted to resolve the error by fitting the observed point on the line predicted by the model, by adapting the minimum and maximum QoS value (lines 5 and 6). The expected QoS values are changed without extending the suitable range. We do reduce the suitable range based on (trivial) lower and upper bounds on the QoS (lines 7 until 12). This step is applied to x_{newmin} and y_{newmin} in Figure 3.9. Finally, the model is adapted to the newly calculated parameter values (line 13).

3.3 Quality-of-Service Estimation

Both network and node QoS metric information are needed by the nodes to support our distributed re-active reconfiguration approach. This network QoS information is determined and disseminated using our generic distributed service (Chapter 2). We again focus on the single trade-off between latency and lifetime and instantiate the service accordingly. Estimating information for other trade-offs works in a similar way. As explained in the previous section, a node needs

Algorithm 8: adaptForCurrentQoS(), for $y_{min} > y_{max}$ (decreasing trend)

```
1  $x_{newmin} := x_{min}, y_{newmin} := y_{min}, x_{newmax} := x_{max}, y_{newmax} := y_{max}$;
2  if $(x_{min} \leq x_{cur} \leq x_{max})$ then
3      $y_{expected}$ := getLatencyFromModel($x_{cur}$);
4      $r := y_{expected} - y_{cur}$ ;
5      $y_{newmin} := y_{min} + r$;
6      $y_{newmax} := y_{max} + r$;
7      if $(y_{newmin} > maxQoS)$ then
8          $x_{newmin} := x_{min} + (y_{newmin} - maxQoS) * (\frac{x_{max}-x_{min}}{y_{max}-y_{min}})$;
9          $y_{newmin} := maxQoS$;
10     if $(y_{newmax} < minQoS)$ then
11         $x_{newmax} := x_{max} - (minQoS - y_{newmax}) * (\frac{x_{max}-x_{min}}{y_{max}-y_{min}})$;
12         $y_{newmax} := minQoS$;
13 AdaptModel($x_{newmin}, y_{newmin}, x_{newmax}, y_{newmax}$);
```

estimates of the latency of its end-to-end critical path. Additionally, both the sum and the sum of the inverses of the remaining lifetime of the collaborating nodes, simply referred to as the sum and inverse sum, are needed. Section 3.3.1 summarizes the relevant aspects of the service and shows the equations characterizing our required network metrics. The local estimation of node QoS required to compute network QoS is discussed in Section 3.3.2.

3.3.1 Network QoS Estimation

The estimation of the required network QoS information involves several steps. The first step is for nodes to estimate their maximum latency to the a given sink, $lat_{i,s}$, which describes the estimated maximum time it takes to send a data packet from the node (over one or more paths) to the sink. For the end nodes of every end-to-end critical path, the latency to the sink is equal to the latency of their end-to-end critical path, $e2elat_i$, which is the value we want to keep within certain bounds. This value is distributed allowing all other nodes to estimate their latency of the end-to-end critical path to the sink. As soon as the end-to-end critical path is locally known by all nodes, collaborating nodes can be defined and the sum and inverse sum of their lifetimes, $sumLife_i$ and $sumInvLife_i$, can be estimated.

Latency-related QoS estimation Figure 3.10 shows the flow of information needed for nodes of a 5 node network (including sink S) to locally estimate the maximum latency to the sink and the latency of the end-to-end critical path (to the same sink). The solid lines are links and the number next to them the latency to communicate over the link. The dashed lines show how the information is communicated and the order in which this is done. The sink (with known fixed

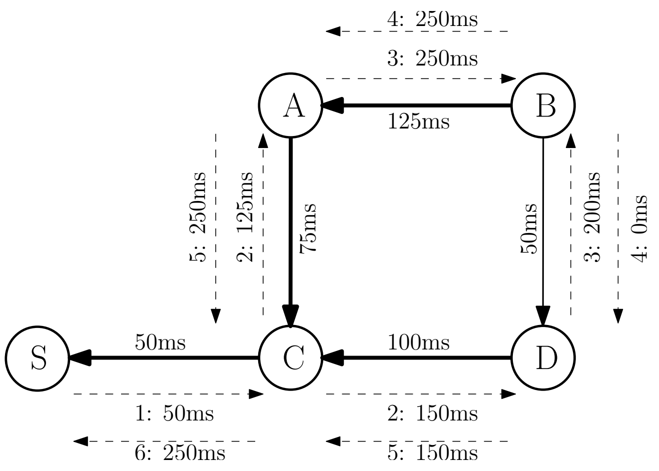


Figure 3.10: Distribution of the latency to sink S and latency of the end-to-end critical path

maximum latency to the sink of 0) starts by observing the latency of the link from C to be 50 milliseconds and communicates this value added to its own maximum latency to C (step 1). With this information, node C determines its critical parent to be S and its maximum latency to the sink to be 50 milliseconds. Using this maximum latency and the observed latencies of nodes A and D, node C informs A en D about their maximum-latency to the sink when using C as a parent (125 and 150 milliseconds respectively, step 2). With this information, they determine their critical parent to be C. Node A and D both individually inform B about its maximum latency when using them as its parent (step 3). With the latency through node A being the highest, node B selects A as its critical parent and its maximum latency to be 250 milliseconds. Note that the maximum is considered critical as we require the latency of every packet sent to be in between the required bounds, which is ensured if the path with the longest latency is within bounds.

This procedure of estimating the maximum latency, $lat_{i,s}$, of node i to sink s, can be characterized by the fixed point of the following recursive definition, with which nodes can estimate their maximum latency to the sink based on both the maximum-latency to the sink of all used parents, P_i, and the link-latency to send packets directly to these nodes, i.e., $ll_{i,x}$ to neighbour x. P_i is assumed to be available by the routing protocol, and p_i is the critical parent.

$$p_i = \underset{x \in P_i}{\operatorname{argmax}}(ll_{i,x} + lat_{x,s})$$

$$lat_{i,s} = \begin{cases} 0 & \text{if} \quad i = s \\ ll_{i,p_i} + lat_{p_i,s} & \text{if} \quad i \neq s \end{cases}$$

It is straightforward to show that the equations have a unique solution.

Using the estimates of maximum latency to the sink, and the critical parent, nodes can estimate the latency of the end-to-end critical path. For the example of Figure 3.10, node B does not receive packets from other nodes, making its latency of the end-to-end critical path equal to its own maximum latency to the sink, i.e., 250 milliseconds. Its critical parent A is part of the same end-to-end path and is informed about this value. At the same time, a latency value of 0 is communicated to the non-critical parent, D, to indicate that B is not on their critical path (step 4). Subsequently, node A determines node B to be its critical child and forwards the received end-to-end latency to C. At the same time, D observes that is has no critical child(ren), i.e., it is the end node of its end-to-end critical path, and sends its own maximum latency to its critical parent C (step 5). C has learned that it is on multiple end-to-end critical paths. The maximum latency path is of interest and hence the critical child is selected to be A. A informs its own critical child S about this value (step 6).

The following recursive equation characterizes the local estimates of the latency of the end-to-end critical path, $e2elat_i$. Let Ch_i be the set of children of node i from which it receives application packets. C_i is the set of nodes for which node i is the critical parent. From these nodes, the critical child, c_i, is selected. For a node at the end of the critical path, i.e., that has no children which consider it as their critical parent, the latency of the end-to-end critical path is equal to its own latency to the sink, $lat(i, s)$. Starting from these end nodes, maximum end-to-end latency is communicated, allowing connected nodes to determine their value based on the end-to-end latency of their critical child, c_i.

$$C_i = \{x \in Ch_i | p_x = i\}$$

$$c_i = \underset{x \in C_i}{\operatorname{argmax}}(e2elat_x)$$

$$e2elat_i = \begin{cases} lat_{i,s} & \text{if} \quad C_i = \emptyset \\ e2elat_{c_i} & \text{if} \quad C_i \neq \emptyset \end{cases}$$

Lifetime-related QoS estimation Given local knowledge of the critical parent and child, the sets of collaborating nodes are known and the related sum and inverse sum of their lifetimes can be estimated. Figure 3.11 shows the flow of information needed to locally estimate the sum and inverse sum for the same 5-node example. The number next to the nodes represents the remaining lifetime of the node in hours. Starting at the leaves of the end-to-end critical paths, B and D, having no critical child, lifetime and inverse lifetime are sent to their critical parent (node A and C respectively, step 1). Node C responds to the information received by (non-critical child) node D with a sum and inverse sum of 0 to indicate it is considering a different end-to-end critical path, and is therefore not a collaborating node of D. At the same time, node A adds its own lifetime and inverse lifetime to the

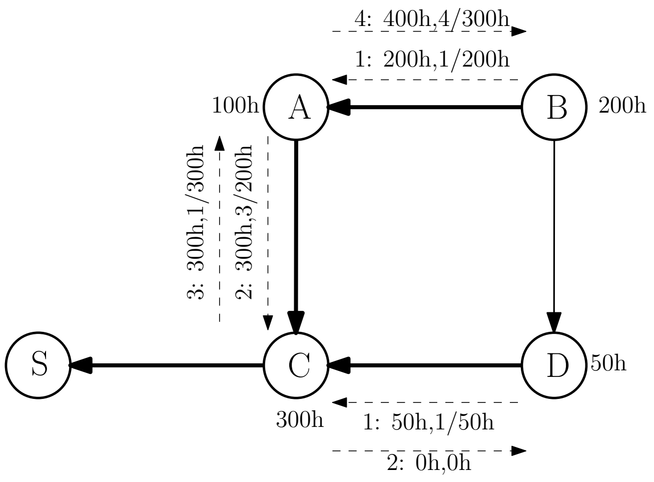


Figure 3.11: Distribution of the sum and inverse sum of lifetimes of collaborating nodes

values received from its critical child, and hence its collaborating node, B, and forwards this information to critical parent C (step 2). Now node D knows it is not collaborating with any other node and has sufficient information to determine its sum to be 50 hours and inverse sum to be $1/50$ hours. Node C has sink S as its critical parent, which is assumed to not take part in the reconfiguration and cannot be a collaborating node. As a result, node C is the end of the path with collaborating nodes starting at node B. It sends its own lifetime and inverse lifetime to node A allowing it to determine the sum to be 200 (node B) + 100 (own lifetime) + 300 (node C) = 600, and inverse sum $1/200 + 1/100 + 1/300 = 11/600$ (step 3). Node A provides node B with the remaining information needed to determine the sum and inverse sum of all other collaborating nodes, i.e., A and C, to that node (step 4).

In contrast to the latency to the sink and the end-to-end critical path, distributed estimation of the sum and inverse sum, $sumLife_i$ and $sumInvLife_i$, requires information about the sum and inverse sum from both the collaborating nodes on the inbound part of the critical path, $sumIn_i$ and $sumInvIn_i$, and outbound part of the end-to-end critical path, $sumOut_i$ and $sumInvOut_i$. This is expressed by the following formulas.

$$sumLife_i = sumIn_i + lifetime_i + sumOut_i$$
$$sumInvLife_i = sumInvIn_i + lifetime_i^{-1} + sumInvOut_i$$

Recursive equations for estimating the sum and inverse sum of the collaborating nodes of the inbound and outbound part of the critical path are as follows. Recall that c_i and p_i are the critical child and critical parent of node i respectively, with c_{p_i} thus indicating the critical child of the parent of node i (which is not necessarily node i itself).

$$sumIn_i = \begin{cases} 0 & \text{if} \quad C_i = \emptyset \\ sumIn_{c_i} + lifetime_{c_i} & \text{if} \quad C_i \neq \emptyset \end{cases}$$

$$sumOut_i = \begin{cases} 0 & \text{if} \quad P_i = \emptyset \vee p_i = s \vee c_{p_i} \neq i \\ sumOut_{p_i} + lifetime_{p_i} & \text{if} \quad P_i \neq \emptyset \wedge p_i \neq s \wedge c_{p_i} = i \end{cases}$$

$$sumInvIn_i = \begin{cases} 0 & \text{if} \quad C_i = \emptyset \\ sumInvIn_{c_i} + lifetime_{c_i}^{-1} & \text{if} \quad C_i \neq \emptyset \end{cases}$$

$$sumInvOut_i = \begin{cases} 0 & \text{if} \quad P_i = \emptyset \vee p_i = s \vee c_{p_i} \neq i \\ sumInvOut_{p_i} + lifetime_{p_i}^{-1} & \text{if} \quad P_i \neq \emptyset \wedge p_i \neq s \wedge c_{p_i} = i \end{cases}$$

The example suggests the network is in a static situation, with bidirectional links. Presence of any asymmetric, and potentially unidirectional, communication links [27, 84], requires the information to be disseminated over multiple hops. The service provides solutions for that using a controlled n-hop forwarding approach where information is efficiently propagated to the n-hop neighbourhood. The data dissemination is repeated at a given update interval to avoid stale information due to dynamic changes in link qualities.

3.3.2 Node QoS Estimation

Node i should know its own link latency to all parents x, $ll_{i,x}$ to derive maximum latency information and a critical parent. An estimated remaining lifetime, $lifetime_i$, is needed for the derivation of the sum and inverse sum of the lifetimes of the collaborating nodes.

The latency of a packet can be determined by receiving nodes based on time stamps added to the packet by the sending side, assuming the presence of a time synchronization protocol. For the sending nodes to know their latency to neighbouring nodes, communication of the observed information is needed which can be combined with the distribution of the latency to the sink, as discussed in the previous subsection.

Estimating the remaining lifetime, defined by the time until the node runs out of battery, is non-trivial, because of the unpredictable changes in power consumption due to, among others, future reconfigurations. For our approach, we estimate the remaining lifetime by dividing the estimated remaining battery capacity (in mWh) by an estimate of the current power consumption (in mW). This provides the remaining lifetime, assuming the network stays in the current configuration (and the estimation of the power is accurate). With our approach, we are interested in the relative differences between the lifetime of nodes. The absolute value is of less interest, but may be more accurately estimated using more elaborate models which, for example, make predictions on the future average power if available. The remaining battery capacity is determined based on recorded average power consumption since powering up the node and the initial battery capacity, e.g., an AA battery of 1500mAh at 1.5V. For a node to determine its own power consumption at run-time is a difficult task. Estimates can be achieved by monitoring the amount of time spent in the different radio states, as the radio is the dominant part of the power consumption [49] and the most important part of the power controlled by reconfiguration. In TinyOS [70] this monitoring can be done by adding counters at the appropriate places in the MAC protocol code. Using knowledge about the power spent in different radio states for the used radio chip (e.g. [68]) an estimate of the power consumption can be made. This approach is used for all experiments in this chapter. Note that the node with the highest power consumption might not have the lowest expected remaining lifetime, for nodes with different battery capacities which are potentially also powered on at different moments in time. This is our main motivation to focus on lifetime, instead of power consumption, when reconfiguring a heterogeneous network.

3.4 Controller Performance Analysis

We start this section by discussing the approach used for performance analysis in more detail in Section 3.4.1. In Section 3.4.2, we discuss the parameters that influence the behaviour of the reconfiguration approach in more detail. We continue with a qualitative analysis of the behaviour of the feedback control strategy by looking at the so-called step-response in Section 3.4.3. It gives us important insight in the behaviour of our approach and its ability to control the QoS in a distributed manner, including its interaction with the adaptive predictive model. Additional experiments focus on the impact of using this particular dynamic model, by comparing its accuracy with that of a static model and an 'oracle' model using knowledge of the future dynamics.

In Section 3.4.4, we analyse a dynamic monitoring WSN for which the reconfiguration approach constantly needs to respond to dynamic events, such as moving nodes. Here we explore to what extent the amount of the dynamics influences the behaviour of our approach. Furthermore, we compare our approach with existing (re-)configuration approaches. Finally, in Section 3.4.5, we summarize the main conclusions. In this section, we use extensive simulations to evaluate the performance. Performance analysis using an actual deployment is performed in the next section.

3.4.1 Introduction

During the distributed feedback control process of our reconfiguration approach many actions happen at the same time, including the adaptation of parameters, propagation of network QoS information, and dynamic changes to the predictive model. The adaptation of parameters is discrete, and propagation of feedback takes time and is unreliable. The resulting behaviour of the entire network, and impact on the network QoS, is therefore non-trivial to determine in advance. Due to the differences between the environment in which we use our controller and discrepancies with assumptions of common control theory, i.e., fully distributed control, discrete adaptation steps, feedback propagation that is delayed and unreliable and the use of an adaptive predictive model, we cannot resort to the analytical reasoning of classical control theory, but we use an experimental analysis approach using simulations in this section and an actual deployment in the next section. A simulator gives us, in contrast to an actual deployment, a controllable environment where we can easily repeat and compare experiments using the same environmental characteristics. This helps us to get a good insight in the behaviour of the approach and the impact of its parameters. For the experiments in this section, we implemented the complete reconfiguration approach, including the node QoS estimation and service to propagate the required network QoS, in OMNeT++ [44], a discrete-event simulation environment, with the use of MiXiM [37], a modeling framework for wireless networks.

3.4.2 Parameters

An important parameter of our reconfiguration approach is the speed of every node i, k_i, of the feedback control strategy (see Section 3.2.1). Note that the length of a round also impacts the (maximum) speed of reconfiguration (and model adaptation), but we assume it to be fixed to a low value such that we can control the speed with only parameter k_i. The speed determines how fast individual nodes respond to changes in QoS. Increasing k_i reduces the speed of the controller. Intuitively it is expected that if a controller is slower, the configuration will be sub-optimal for a longer time and QoS may differ too much from the required QoS. On the other hand, if it is too fast, feedback may not have propagated yet and large fluctuations in QoS may occur due to adaptations of many nodes at the same time, potentially resulting in never settling of the required QoS, or even instability of the network.

In our approach, local estimates of network QoS are provided by an underlying service. The parameters of the service determine the speed at which information for local estimates becomes available to the nodes. As the availability of network QoS information forms the basis of our reconfiguration approach, the speed of propagation is important for the stability, as we show in this section.

3.4.3 Step Response

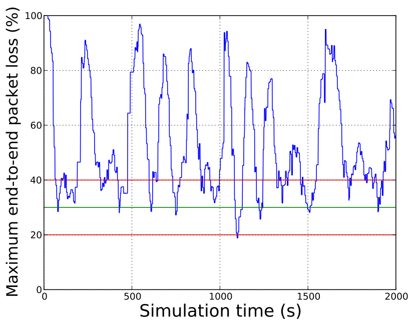
The step-response is a well-known from of analysis from control theory that investigates the behaviour of a feedback controller after a single dynamic event [67]. We consider end-to-end packet-loss to be the constrained metric, while we want to maximize lifetime. This shows the applicability of our approach for other metrics than the latency metric considered so far, but also has practical advantages as, in contrast with latency measurements, no time synchronization is required. Of main interest is whether the controller is able to resolve the impact on packet-loss caused by the dynamic change, i.e., whether and how it gets the network back into a stable situation. In an unstable situation, the packet-loss diverges from its target and may fluctuate between its extreme values. In a stable situation this still needs time, referred to as the convergence time. The amount of variation in QoS during the stable situation is a measure of the accuracy of the system.

For the analysis of the stability, convergence speed and accuracy we investigate a network consisting of a 5x5 grid of 25 TelosB nodes [68]. They communicate to a sink over one or multiple hops using a (dynamic) minimum-cost routing protocol where the neighbour resulting in the lowest possible packet-loss is selected as parent. Furthermore, the B-MAC [48] protocol, available in MiXiM, is used to manage communication over the shared medium. The link quality is modeled by the commonly used log-normal shadowing path loss model. The objective used in the experiments is to maximize the minimum remaining lifetime of the nodes in the network, while the packet-loss to the sink, averaged over the packets in the last minute, is maintained between 20% and 40%. The nodes send an application packet to the sink every 5 seconds. We assume the nodes to have the same initial battery-capacity of 2 x 1500 mAh at 1.5 V. The initial network has 10 meter distance between neighbouring nodes in the grid and a suitable configuration and predictive model parameters are selected for this initial situation. These optimal values are determined using simulations. At the start of the analysis of the step response ($t = 0$ for the figures in this section), the distance between adjacent nodes is increased to 15 meter. This large step enforces a significant change in QoS for which the network needs to reconfigure. After this step no other external dynamics are introduced and nodes are only affected by internal dynamics due to changing their parameters.

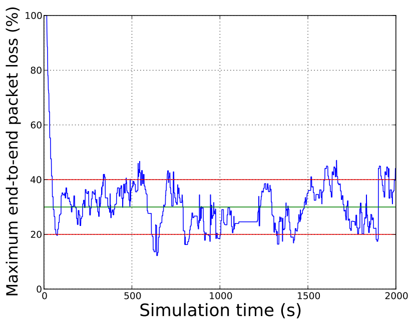
We set the length of a round to be 1 second and the service propagates network QoS information at an update interval of 5 seconds. We assume that this is the highest update frequency possible given the maximum allowed overhead of the service. The parameters that we allow to be controlled are the radio transmission power (any of the values specified in [68]), number of transmissions of a single packet (between 1 and 10 times) and the number of receive buffer spaces (between 1 and 10). This gives us a useful range of parameters to influence the trade-off between packet-loss and lifetime.

Impact of controller speed Figure 3.12 shows the step responses for the first 2000 seconds for three experiments with different speeds, i.e., values of k_i, for all nodes in the network.

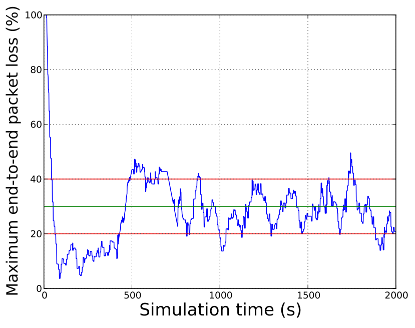
Figure 3.12(a) shows the step-response for $k_i = 1$, for all nodes. This value means that the collaborating nodes together try to solve the observed error in a single round of one second. The step response shows large fluctuations in packet-



(a)



(b)



(c)

Figure 3.12: Packet-loss with controller speed that is (a) fast ($k_i = 1$), (b) sufficient ($k_i = 90$) and (c) slow ($k_i = 250$)

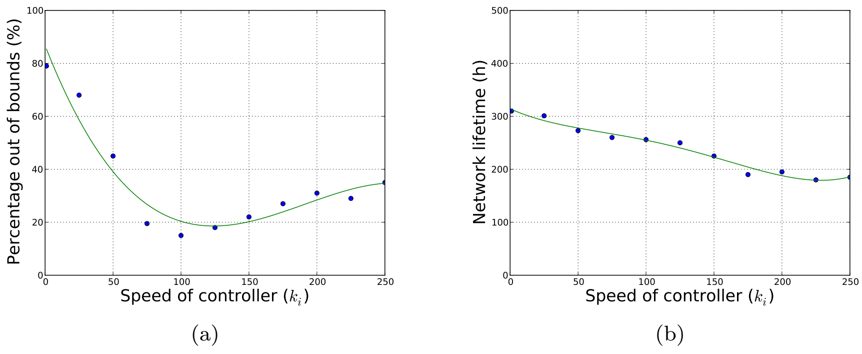


Figure 3.13: Impact of controller speed on packet-loss behaviour and expected network lifetime

loss. Parameters are quickly adapted completely until one end of the suitable range to reduce the loss, because the impact of a single adaptation step is not fully observed before the next adaptation. Due to the use of a suitable range in the predictive model, we do not observe both a repeated under and overshooting of the target QoS, as was observed with using a static model which allows the parameter to take every possible value from its range [59]. The dynamic model does not have the opportunity to adapt (as the time between adaptations is less than the averaging time of 60 seconds), and the model used for the old situation is constantly used. This results in instability of the packet-loss and should be avoided.

Figure 3.12(b) shows the step-response for $k_i = 90$. We see that the distributed adaptation of parameters is able to keep the packet-loss within the target range, with only occasional outliers. It quickly converges to a stable situation. Small fluctuations remain due to discrete steps in the adapted parameter values and random variations in the packet-loss of the links. Furthermore, inaccuracy in the predictive model can result in adaptations causing deviations from the target packet-loss range.

Figure 3.12(c) shows the step-response for $k_i = 250$. The controller converges to an accurate stable situation, but requires a long time to converge. Every time dynamics occur that cause the packet-loss to be outside the required range, a long time is needed to reduce the resulting packet-loss error. Therefore, like a controller which is too fast, a controller which is too slow is not preferred.

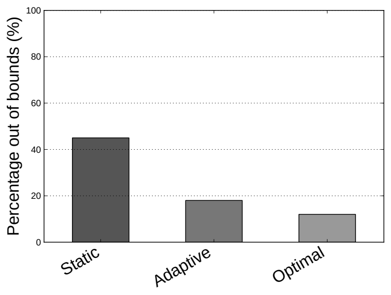
Figure 3.13 shows the percentage of time for the 2000 seconds experiments that the maximum packet-loss is outside the required range and the expected remaining lifetime of the network, for a larger range of k_i. The results are averaged over 10 simulations per value of k_i. The line is a polynomial approximation to show the trend. The combination of this information is used as a measure of the quality of QoS provisioning. The remaining lifetime is what we want to optimize in the end. Note that our approach focuses on the optimization for the given scenario. If lifetime is still deemed to be insufficient, the scenario itself should be adapted (reduce amount of communications and size of packets using, for example, data fusion) to allow a better optimization, which is outside the scope of this work. Low k_i favors lifetime, at a cost of a larger percentage out of bounds. For high k_i, we have a slow convergence where most of the time a configuration is used with a lower packet-loss than required, costing too much lifetime.

There is a range of speeds with approximately the same low percentage and high lifetime, in this case between around 75 and 125. For a deployment, we need to select the speed such that we get the desired behaviour as in Figure 3.12(b), i.e., to select k_i in the range providing the best possible provisioning of QoS. Closer analysis of the results reveals that the step response, and hence a suitable value of k_i, strongly relates to the (average) time needed to propagate a change in network QoS information. For a fast controller, with a step-response similar to Figure 3.12(a), nodes repeatedly adapt based on inaccurate local information of the network QoS as time between adaptations is often lower than the time needed to propagate information updates. In other words, instability is caused by a mismatch between the speed of network QoS propagation of the service and the controller speed. Furthermore, with the averaging of network QoS information, e.g., average packet-loss over one minute, it takes time for the impact of a parameter adaptation to be fully reflected in the network QoS. In case that the speed is too slow, local estimates of network QoS are present on the nodes, but nodes are simply respond too slowly to an observed error.

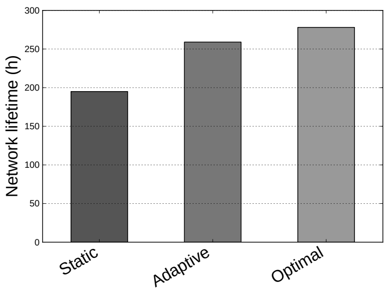
Consider the case that on the longest critical path in the network, the node closest to the sink experiences a change in link-latency. Propagation of this change along the entire path is needed to update the (maximum) latency to the sink of all nodes (since the maximum occurs at the end of the path). From the end node of the path, both the end-to-end latency and the sum and inverse sum of the inbound part of the critical path are communicated till the other end of the path. At that point, the sum and inverse sum of the outbound part of the path are propagated over the entire path. In other words, given a length l_i of the end-to-end critical path of node i (which has the same value for all nodes on the same end-to-end critical path) and an update interval of network QoS information of u seconds, the time needed to update all estimates on the path is $3 * l_i * u$, assuming bidirectional links and no packet-loss. Losing packets and using multiple hops to communicate over asymmetric/uni-directional links may increase this time. On the other hand, several factors also reduce the time needed to propagate a change. If the critical parent and child are known and do not change due to the observed packet-loss change, lifetime information is propagated with the latency information. No lifetime update is needed at all if the collaborating set of nodes remains the same and lifetime has not significantly changed. Furthermore, the update intervals do not need to be synchronized between nodes potentially resulting in less time on average than a complete interval to forward a packet to the next node. Which factors are dominant depends on network characteristics. The factors combined determine a scaling factor s, which is 1 if the time increasing and decreasing factors are in balance. Resulting optimal speed of the controller is $s * 3 * l_i * u$. Design-time analysis allows us to approximate s. For the deployment considered for our simulations, we observe a maximum path-length of 8 and use an update interval of 5 seconds. A suitable k_i is expected to be in the order of $3 * 8 * 5 = 120$, with $s = 1$. From simulations we see that k_i can be set lower for the considered type of networks, since the propagation time is lower on average. We approximate $s = 2/3$ and thereby an optimal speed of $k_i = 2/3 * 3 * l_i * u$, and $k_i = 80$ for our experimental analysis.

At run-time, the impact of the factors influencing the optimal speed may change. The equation assumes knowledge of the worst-case end-to-end path length. In practice we may only be able to determine one which may be very conservative. This results in a controller which is stable, but unnecessary slow. Therefore, it may be interesting for nodes to set the speed of the controller depending on the current length, or hop-count, of its end-to-end critical path. As a result, the speed can differ between nodes. Compared to the fixed (worst-case) speed approach, critical paths consisting of a fewer number of nodes than the longest path in the network have a speed that better matches the speed at which network QoS information is available. They thereby adapt earlier than with the worst-case speed approach avoiding unnecessarily long periods of suboptimal QoS. For the experimental analysis, we use a fixed $s = 2/3$ and dynamically changing l_i based on the current hop-count of the end-to-end critical path. The approach can be extended by measuring other run-time factors at run-time, such as the average packet-loss of a link, to have a more fine-grained influence on the controller speed by dynamically adapting s as well. With a changing s, the update interval u can be adapted to maintain the speed of the controller. Additional analysis is needed to explore the issues involved with this more complex run-time tuning of the controller speed. Besides using knowledge of the current dynamics in the network, knowledge of upcoming dynamics helps to pro-actively set the speed and adapt for upcoming impact of dynamics. One could even think about temporally stop the re-active reconfiguration, and rely on a single worst-case configuration, if dynamics get extremely high. In Chapter 4, we go into detail on how to exploit a-priori knowledge of dynamics to adapt the configuration, including, for example, the speed of re-active reconfiguration.

Impact of the predictive models An adaptive predictive model is expected to have a positive impact on the behaviour of the controller (compared to a static model) at the expense of maintaining the model parameters. The following experiments focus on the accuracy of the predictive model compared to the static impact model calibrated for the situation before the step. We furthermore compare to an 'oracle' model which predicts the impact of adaptation based on extensive simulations with all the possible configurations. This model is used to determine



(a)



(b)

Figure 3.14: Impact of predictive models on packet-loss behaviour and expected lifetime

the remaining inaccuracy of the dynamic model.

Figure 3.14 shows the average percentage of time the end-to-end packet-loss is outside the required range and the resulting lifetime using the three predictive models, for the same set of simulations.

The static model results in a fairly large percentage of time in which the packet-loss is out of bounds, i.e., either higher or lower than the predefined packet loss range. With the dynamic approach, the percentage reduces and lifetime increases compared to the static approach as bounds on both the parameter values and expected QoS are considered. This results in less over and under-shooting, as adaptation focuses on parameter values within the range, while the static approach will continue to adapt parameters even though no significant impact on the metric is observed. The dynamic model still has some inaccuracy compared to the oracle model as time is needed to find accurate model parameters, but the deviations are small, both in packet loss and lifetime.

3.4.4 Dynamic Setup

In the remainder of this section, we consider a dynamic scenario by adding 5 mobile nodes to the setup, which we assume to have half the battery capacity of a static node. This results in network characteristics of monitoring scenarios, such as the health monitoring. With the heterogeneity in battery capacity, balancing the effort to provide the required QoS becomes even more important. The distance between (adjacent) static nodes is 15 meters and is not changed.

We first investigate the impact of dynamism on the efficiency of our reconfiguration approach. We assume that the mobile nodes move at walking speed (2 m/s) for 10 seconds in a random direction within the grid and remain at the resulting location for a given fixed amount of time. The amount of dynamism is

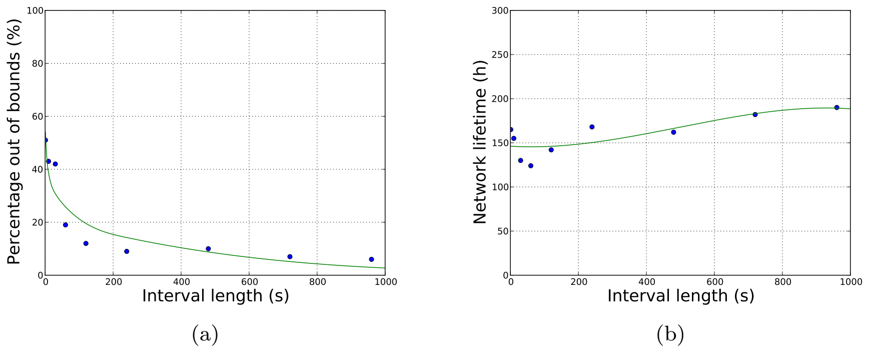


Figure 3.15: Impact of amount of dynamics on packet-loss behaviour and expected lifetime

determined by the length of this time interval between two node movements. It is expected that the slower and less frequent the dynamics, the more efficient our approach will be as more time is available to determine network QoS and adapt to a suitable configuration. We verify and analyze this expectation below. Finally, we perform simulations to compare the efficiency of our approach with existing (re-)configuration approaches. For this, we compare with a worst-case static configuration approach, often applied in current practical deployments, and what we consider to be the most suitable run-time configuration approach, which focuses on providing local QoS instead of network QoS.

Impact of the amount of dynamics Figure 3.15 shows both the average time the observed maximum packet-loss to the sink is outside the target range and the expected network lifetime, for various lengths of the interval between two mobile node movements. Simulations have a length of 5000 seconds.

With a larger amount of dynamics, the ability of our controller to keep the maximum packet-loss within the target range reduces. A lower amount of dynamics has a positive effect on the remaining lifetime of the network. For the short interval lengths there is no clear lifetime trend, as the remaining lifetime depends on the fluctuating ratio between time spent in situations where the packet-loss is too low (spending too much lifetime) and too high (not spending enough lifetime).

For the explanation of this behaviour, we have to look at the estimation error of the packet-loss at the moment an adaptation is done. If a change is not fully reflected in estimates of other nodes, reconfiguration is done based on outdated QoS estimates. Moving nodes cause many and large changes in QoS, due to nodes joining and leaving critical paths. As a result, end-to-end critical paths change and collaborating sets need to be updated. For a given speed of the controller, faster movements cause a larger difference between the actual and estimated QoS.

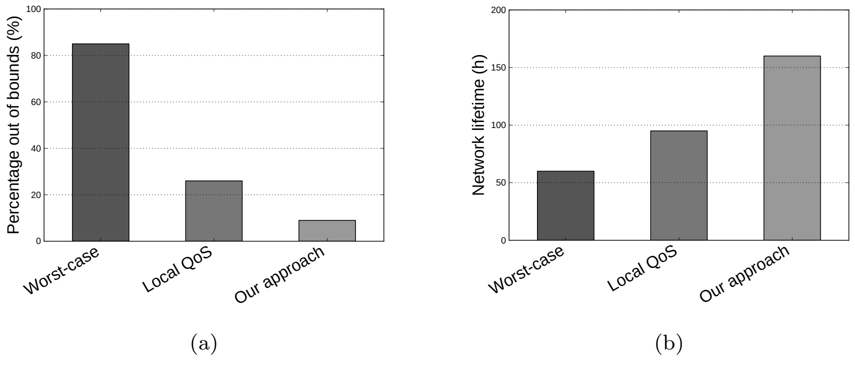


Figure 3.16: Reconfiguration approach comparison

For this particular case, if the time between significant movements of nodes is at least two minutes, as is the typical case for, for example, health-monitoring, the controller is able to efficiently provide QoS.

Comparison with existing (re-)configuration approaches To compare our approach with existing (re-)configuration approaches, we first compare with a single, static, configuration approach using a configuration based on the worst-case dynamics. Then, we compare with what we believe is the best approach based on existing run-time reconfiguration strategies. This approach focuses on local QoS instead of network QoS. Because there exists no approach directly usable from current literature that considers adapting multiple parameters to control multiple conflicting network quality metrics, we construct one with a straightforward integration of a local QoS provisioning technique. It uses the same predictive model as our approach but, instead of adapting based on network QoS, the end-to-end packet-loss in this case, it adapts to provide local QoS, the packet-loss to the immediate parent, independent of the remaining lifetime and condition of other links. The number of hops to the sink determines the minimum packet-loss every link should independently have to have a resulting packet-loss to the sink within the range between 20% and 40%. As a local approach does not consider this network information, we have to make assumptions on this. We assume a maximum number of hops of 8 to reach the sink and therefore focus on providing the packet-loss of every link to be as low as 5%. The experiments confirm that this level is necessary and sufficient.

Figure 3.16 shows the results for simulations of the dynamic set-up with an interval between movements of 10 minutes, using the three approaches. For the single, worst-case, configuration, a large percentage of time the packet-loss is outside the target range; packet delivery ratio is higher than required. The goal of the worst-case approach is not to keep the packet-loss in the target range, but to have a packet-loss lower than the target in all cases. As a consequence of the trade-off with lifetime, we observe a very low remaining lifetime. As expected, very power expensive configurations are used to provide a low packet-loss in all possible situations, while worst-case situations, for which these configurations are useful, are rare. The worst-case approach is clearly not a viable option for a dynamic environment requiring a long lifetime.

The other two approaches consider the trade-off between packet-loss and lifetime, and keep packet-loss in the target range. We do still see the packet-loss to be occasionally outside the required range. For the local QoS provisioning approach this percentage is slightly higher, as the packet-loss is often found to be lower than necessary. This is because local decisions rely on worst-case assumptions about path-length.

Looking at the remaining lifetime, we see a clear advantage of our approach from focusing on network QoS, including the overhead involved with network QoS propagation, instead of local QoS. The main reason is that focusing on controlling the packet-loss to the parent, lower lifetime nodes, and the mobile nodes in particular, have the same packet-loss constraint as the higher lifetime nodes. Low lifetime nodes spend lifetime on providing a low packet-loss to the parent, independent of the ability of other nodes on the path to do this more efficiently. Our approach balances the lifetime, forcing the higher lifetime nodes to spend more on obtaining a low packet-loss to the sink providing more flexibility to the lower lifetime nodes to extend their lifetime.

3.4.5 Conclusions from Simulations

From the simulations we can draw several conclusions. First, our reconfiguration approach is able to provide sufficient QoS with a proper speed of the feedback control. Optimal speed is found to be related to the time needed for network QoS estimates propagation. As the required speed depends on run-time properties, such as the path length, we suggest to dynamically adjust the speed of a controller to these run-time properties.

The introduced adaptive predictive model is more accurate than the previously introduced static approach allowing a better QoS provisioning at the expense of maintaining only four model parameters for every parameter-metric pair.

There is a relation between the amount of dynamics that is present in a deployment and efficiency of QoS provisioning. Our reconfiguration approach is found to be most suitable for deployments with frequent changes with small impact and infrequent changes with immediate larger impact on network QoS. This makes our approach useful for the typical dynamics found in (health-)monitoring scenarios. Compared to current reconfiguration approaches, our approach shows significant improvement in the ability to provide QoS, defined by the ability to keep the packet-loss in a given range, while maximizing the network lifetime.

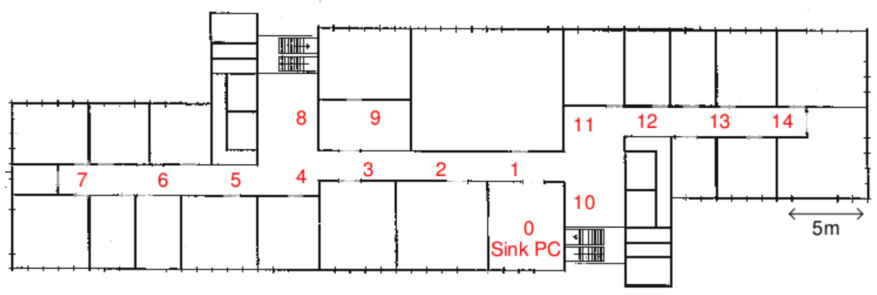
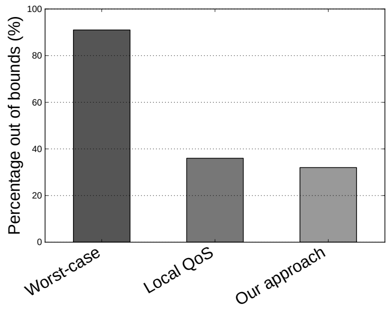


Figure 3.17: Deployment set-up static nodes

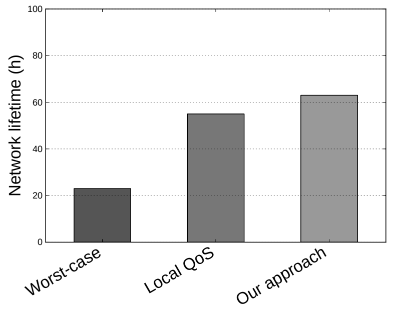
3.5 Experimental Analysis

We implemented our reconfiguration approach, together with the distributed network QoS estimation service, in TinyOS [70] for a WSN deployed in our office building. It consists of 15 static TelosB [68] nodes deployed as shown in Figure 3.17. Furthermore, 5 persons, which occasionally move between different locations in the office building, have a BSN [8] sensor node attached which sends application packets at a rate of one every 2 seconds to the sink (node 0), using the static nodes when needed. Together with the protocol stack, the code requires around 7KB RAM and 40KB ROM for both the TelosB and BSN implementations, which fits easily on those nodes. Currently, a large amount of data storage is reserved for link quality estimation. Code optimization may be possible to reduce the code size. We want to establish a packet delivery ratio between 60% and 70%. Occasional misses of the constraint are allowed in exchange of a significant increase in lifetime of the network, which we want to maximize.

The MAC protocol for all nodes is the Low-Power-Listening MAC [38] available in TinyOS. It is a basic asynchronous MAC protocol which reduces idle listening by sampling the medium at a given interval. On top of this we use an approach inspired by [5] where we always retransmit a packet a given number of times and instead of relying on the receiver to send acknowledgments. This avoids to adapt the acknowledgment mechanism to deal with asymmetric links. Static nodes use a fixed (static) node to route data to. The mobile nodes use an approach of repeatedly requesting packet-loss information from nodes in range. The node resulting in the lowest packet-loss to the sink is used as parent. The protocol stack has several controllable parameters and for run-time adaptation we focus on two of them. Firstly, we consider the transmission power of the radio. An increase in transmission power can potentially reduce the packet-loss to a given parent and result in mobile nodes finding neighbouring nodes with a lower packet-loss, at the expense of a shorter lifetime. Secondly, we look at the number of retransmissions of the MAC protocol. Extra retransmissions could reduce the link packet-loss,



(a)



(b)

Figure 3.18: Comparison of configuration approaches for (a) percentage out of bounds and (b) network lifetime

but the radio will be spending more time in transmission and listening modes requiring more power compared to being idle, resulting in shorter lifetime.

Before the nodes can perform their task, controllable parameters and parameters of the reconfiguration approach, including the distributed service, have to be initially set. Based on small scale experiments, we select the length of a round of the service to be 10 seconds and the number of hops used to forward packets to be 1 to deal with asymmetric links. We set an initial value for the controllable parameters and determine suitable parameter ranges for the predictive models using simulations, small scale experiments and experience with the hardware. The selected values and ranges may be different for every node and may not be optimal. This is not a problem since the configuration and predictive model will be updated at run-time as needed. However, a good setting does reduce the initialization time in which the approach settles itself in a good configuration. Given the moderate set-up size and regular structure of the network, manual parameter selection is feasible. Future work could focus on an automated phase where nodes calibrate the suitable range. Making use of accurate models of the deployment could also benefit guided selection of parameter values.

The simulations in the previous section give us good insights in the functionality of the approach. The goal of our experiments with the deployment are mainly to show the feasibility of the reconfiguration approach to be implemented on resource constrained nodes and to be used in practice. We compare the performance of our controller with existing strategies, also used for the simulations. We assume a maximum number of 4 hops to reach the sink and therefore steer the packet-loss of every link to around 10% in the local QoS approach.

Figure 3.18(a) shows the percentage of time that the maximum packet-loss, of any of the nodes, to the sink is out of the target range, for a 3 hour experiment of the three approaches. To derive packet-loss information at the sink (connected to a PC), information about the observed packet-loss to the parent of the sending node is added to the application packets. Using knowledge of the maximum amount of received (application and service) packets and interpolating for lost packets, the sink determines the end-to-end packet-loss for every node every 10 seconds. Note that the figure considers the maximum packet-loss over all nodes in the network, which does not necessarily refer to the same node during the entire experiment.

Figure 3.18(b) shows the resulting predicted network lifetime of the static and mobile nodes in the network. The estimated lifetime is locally calculated and added to the application packets, similarly to the packet-loss, to be collected by the sink. The expected lifetime is calculated based on the average power spent in the last minute and an assumption on the initial capacity of the used battery. The average power spent by a single node is determined by locally monitoring the time spent in every radio state, i.e., sending, receiving and idle, and the used transmission power. This way to determine the power consumption takes all the packet communication into account, including the additional overhead incurred by the service used for the propagation of QoS information in our approach. For the TelosB nodes we assume the initial energy available to be 2 full batteries at 1500mAh at 1.5V and a smaller full 750mAh at 1.5V battery for the BSN nodes.

We can make several observations. First, we have shown that our reconfiguration approach is simple enough to be implemented on resource constrained nodes and can successfully respond to changes in network QoS in a distributed manner. Similar to the simulations, compared to a single worst-case configuration approach we see a clear improvement in network lifetime, despite the negative impact caused by the overhead of network QoS estimation, as we avoid QoS over-provisioning for the non worst-case situations. Compared to the local QoS provisioning approach, our approach has a higher network lifetime of about 15%, while the percentage out of bounds is slightly reduced (around 5%). The local approach does not consider knowledge of the current state of the system, such as the actual number of hops a packet has to travel and existing heterogeneity in (remaining) node lifetime, in the QoS provisioning. As a result, shorter paths typically show QoS over-provisioning and effort spent by the short lifetime mobile nodes is the same as the long lifetime static nodes. Compared to the set-up used for the simulations, this set-up shows less advantage of our approach over the local QoS approach. This is mainly caused by the more stable single-path routing to the sink, compared to the multi-path grid deployment used in simulations. The local approach trade-offs lifetime to ensure a good quality of every link, independent of whether it is used for routing. Difference between the approaches is expected to be larger when using more parameters and an increased heterogeneity in the network. More extensive experiments are required to confirm this.

Looking closer at the experiments we can make more detailed observations about the functionality of our approach and potential improvements. For both the run-time reconfiguration approaches we often observe switching between two configuration. A fairly large difference in packet-loss is sometimes observed between these two configurations, due to a coarse grained discrete parameter value selection. This results in a good packet-loss on average, but shows consecutive moments of good and bad packet-loss. Ideally we want the behaviour to be smooth over time, also when averaging over a longer period. Selecting from a set of parameter values with a smaller granularity of the impact on packet-loss could support this. On the one hand, this may be achieved by using a larger set of parameters to select from. On the other hand, more fine-grain scalable platforms could be used. A typical example is the transmission power. The used radio only support a given set of parameter values, with potentially large impact on the considered metric between two subsequent values. We observe that in our scenario mobility has an important impact on the optimality of the used configuration. Mobile nodes occasionally move between different locations in the network, thereby influencing the critical path of one or more nodes. The potentially large impact on network metrics is not directly observed by the nodes, due to both the averaging of packet-loss over time and the time needed to distribute network QoS information. The configuration is only slowly adapted and, as a result, potentially sub-optimal configurations are used, until the mobile node resides at a more or less static location, i.e., the person stays at a given location. In that case the reconfiguration approach settles for the new situation and reconfigures accordingly. For our current scenario this is still an efficient solution as the time for a node to be mobile is small compared to the time spent in a relative static situation. To reduce the time of the response needed the re-active approach can be made faster, i.e., faster propagation of network QoS and controller speed, thereby increasing the overhead of the approach. This is no practical solution as the overhead during the times with less dynamics is much more than needed. Furthermore, a sufficient increase of speed may not be possible due to limitations on the maximum allowed overhead. If it is important to quickly respond to infrequent relatively fast dynamics, with a large impact on network QoS, a re-active approach is inherently less efficient. For that case, one might resort to a pro-active reconfiguration strategy, such as discussed in the next chapter. With such an approach, observable network dynamics trigger reconfiguration before their actual impact on the network QoS. Parameters could be proactively changed in response to node mobility, such as the validity time of outbound neighbours, or a completely different routing strategy could be used. Alternatively, the speed of the controller could be temporarily increased to adapt for the faster dynamics, if possible. The presented feedback controller can then fine-tune this proactively selected configuration.

3.6 Related Work

Configuring a WSN to provide a given QoS has become an important topic of research in the last few years. Many approaches exist that incorporate the provisioning of QoS in the design of the used protocols before the actual deployment [64]. Other focus on the design-time derivation of a single (worst-case) configuration independent of the used protocol stack [16, 22]. Design-time approaches require knowledge of the worst-case characteristics of the network and typically focus on static networks. The need for (additional) run-time reconfiguration, or adaptation, to respond to unpredictable dynamism in dynamic heterogeneous networks is recognized by many researchers. Our run-time reconfiguration approach distinguishes itself from existing work by its combination of being fully *distributed*, considering *multiple QoS metrics* at the *network-level* and its applicability in *dynamic heterogeneous WSN*.

With a centralized run-time reconfiguration approach, such as [26], information on the quality of the network is collected and reconfiguration decisions are made at a central location. The practical applicability of centralized approaches is limited for WSNs that are heterogeneous, require a low overhead and fast response to dynamic changes. With distributed approaches, such as our approach, nodes locally decide if and how to adapt their own parameters.

The focus of most current distributed run-time reconfiguration approaches is to optimize for a single network QoS metric, usually power, or only focus on local metrics instead of the network-level metrics that are of interest to the end-user. In [69] a feedback control approach is used to guarantee that each node maintains an average delay for packets transiting a node. In [12] the transmission power and routing decisions are dynamically adapted based on packet deadlines. Several MAC protocols exist that adapt their duty-cycle based on the amount of traffic observed by the node [63, 81]. Considering only a single metric ignores the fact that important trade-offs exist between different QoS metrics that should all be considered for the correct execution of the task. The approaches that do not directly steer the network QoS, but local metrics only, do not consider heterogeneity in the network and it is unclear if network QoS constraints are met.

There is little existing work on run-time adaptation techniques that consider multiple QoS metrics at the network-level. The recent work of [86] confirms the need to consider multiple metrics and proposes a centralized approach determining how to adapt MAC protocol parameters based on centrally collected network QoS information. A homogeneous configuration is assumed where all nodes use the same MAC parameters. We focus on a distributed solution where nodes control their parameter values independent of each other, which allows for heterogeneity among nodes. In [46] a distributed adaptive algorithm that minimizes power consumption, while guaranteeing a given successful packet reception probability and delay is proposed. It only considers the adaptation of the parameters of the IEEE 802.15.4 MAC. The relation between parameter values and the metrics is accurately modeled by mathematical expressions. This model is a suitable predictive model for the particular case in which our approach is used for the IEEE 802.15.4 MAC. Local optimization of the expression is repeatedly done to find ‘optimal’ parameter values. This results in the optimization of local QoS, while our approach considers, global, network QoS. For this, nodes need sufficient information about the network QoS and the status of other nodes, to decide on adaptation of their parameters. In our approach, the nodes receive this information using our distributed service (Chapter 2), which can efficiently propagate information through dynamic heterogeneous WSNs.

Our approach is furthermore independent of the protocols used and number of controllable parameters considered. With the cross-layer adaptation of parameters we exploit the fact that the parameters from all protocols influence the behaviour of the network [35]. While we adapt controllable parameter to fine tune the functionality of protocols, and thereby the behaviour of the node, one could also focus on replacing complete components. With component adaptation, the functionality of a protocol is modified by adding or replacing a component of the protocol or the complete protocol itself [39, 65]. Unless the different components and protocols are known at design-time and stored on the nodes, significant overhead is needed to consistently perform component adaptation. For infrequent dynamics with a large impact on network QoS, component adaptation may be a suitable reconfiguration strategy. Additional analysis is needed to explore the benefits of combining component adaptation with parameter adaptation.

We adopt ideas from the area of control theory and (decentralized or distributed) model predictive control [50]. In our case we control a system which constantly experiences unpredictable dynamics and disturbances. A number of aspects make this control problem hard. Due to the use of wireless communication the feedback propagation is delayed and unreliable. Furthermore, we employ a fully distributed approach where every node uses a nested feedback control strategy to adapt both the configuration and predictive model. For the reconfiguration we need to rely on discrete adaptation of the parameters. These aspects are all subjects of active research and the optimal controller synthesis problems have not yet been solved. We develop a pragmatic solution with a strong focus on the practical implementation on sensor nodes. We cannot use the existing analytical analysis approaches in this field of research to analyse our approach and therefore rely on experimental analysis and evaluation.

3.7 Summary

To ensure that the QoS, expressed by multiple metrics, is maintained at run-time, adaptation of controllable protocol parameters is needed. In this chapter, we introduced a re-active run-time distributed reconfiguration approach that actively maintains the required QoS of the network using feedback control. Nodes adapt their parameters based on deviations between the required and locally estimated current network QoS. To estimate QoS in a distributed manner, our generic distributed service is used and instantiated such that estimates of the current network QoS are available to the nodes. The impact of parameter changes on the QoS is predicted using an adaptive model. This model is updated using observations of the impact of reconfiguration on QoS. With simulations, we explored the parameters and characteristics of the approach. We showed, with analysis of the step-response, that our distributed control approach is stable, if the speed of the controller, and the underlying service, is set in accordance with the deployment characteristics. Experiments with an actual deployment showed that we are able to implement the controller on resource-constrained nodes and that it provides appropriate QoS for a dynamic and heterogeneous deployment. Compared to a worst-case single configuration, and a local adaptation approach which does not take network QoS knowledge into account, we are able to maintain required packet delivery ratios and extend the lifetime of the network.

Chapter 4

Pro-active Reconfiguration

In this chapter, we introduce a *pro-active* run-time reconfiguration approach which exploits design-time knowledge of the application scenario dynamics for efficient QoS provisioning. The approach anticipates for the impact that known dynamic events can have on the QoS by pro-actively reconfiguring the network immediately at the moment that changing dynamics will impact the QoS. This approach is complementary to our re-active approach which is generally applicable without knowledge of the application scenario, but adapts only in response to QoS errors instead of anticipating for them. We start this chapter by introducing two practical monitoring scenarios. We show how these scenarios can benefit from a pro-active reconfiguration strategy and use them as a case-study throughout this chapter. Section 4.2 introduces our pro-active run-time reconfiguration strategy and discusses the implementation details for the two practical scenarios. In Section 4.3, we discuss the results of extensive simulations and experiments with an actual deployment. In Section 4.4, we take a look at related work. Section 4.5 concludes.

4.1 Illustrative Scenarios

We use two different scenarios to support the introduction of our pro-active reconfiguration approach; a cow-health monitoring scenario and an office monitoring scenario. They differ in their protocol stack, controllable parameters, deployment and application characteristics, and QoS requirements. In this section, we discuss the details of these scenarios and how they can benefit from pro-active reconfiguration.

Cow-health monitoring Health problems of cows often result in decreased mobility. Therefore, mobility monitoring can lead to an early detection and treatment of health problems, improving animal well-being and lowering production losses.

A health-monitoring WSN was proposed in the WASP project [30, 74] where a sensor node is attached to the leg of a cow. We consider the monitoring of thirty-eight dairy cows with an attached BSN sensor node [8], in a barn of 40 by 60 meters. Twelve static BSN nodes are placed in a grid to support communication to a sink location (being a static node at one corner of the barn). When a cow is walking, acceleration is sampled and transmitted every second, via the static network, to the sink. This information is used to detect mobility problems, such as limping. When a cow is stationary, the sink is only informed about the change of posture of the cow, e.g., standing or laying, to allow activity monitoring. The required network QoS is a loss of the packets sent to the sink lower than 20%, while, as it is very inconvenient to regularly remove the nodes from the cows, the network lifetime is maximized.

The nodes use a basic TDMA MAC protocol with a fixed slot assignment. Every TDMA frame consists of an active period in which every node has a fixed slot for sending, while it listens to all other nodes in the other slots. Furthermore, there is a sleep period in which all the nodes turn off their radio. To get the packets to the sink, we use Gradient-Based Routing (GBR) [54], where nodes maintain their hop-count by periodic flooding initiated by the sink. Packets are only forwarded if received from nodes with a higher hop-count. We consider several controllable parameters of this protocol stack: the TDMA slot-size, used for sending and receiving packets, sleep-time, the radio transmission (TX) power and the flooding interval of GBR, used to update hop-counts.

When looking at the behaviour of the monitored dairy cows, we observe that during most of the day they stay in the barn with limited mobility and twice a day, at regular times, the farmer leads them to a milking parlor. The milking procedure takes up to two hours including the walking. This a-priori knowledge allows us to identify moments in time after which significant changes to the QoS may arise, and reconfiguration could help to avoid this change. When we look at the expected behaviour of the entire network, the process of milking the cows heavily impacts the traffic load in the entire network and network topology, as during this period all cows will (slowly) walk to the milking parlor. An increased amount of traffic and changing topology can result in the overloading of buffers, collisions during communication, and loosing the option to communicate to intended neighbouring nodes. As a consequence, packet-loss increases and may potentially violate the constraint. To avoid this, we can adapt parameters to positively affect the amount of data able to be handled by the network, for example reducing the TDMA sleep time. We can furthermore increase the responsiveness to a changed topology by, for example, increasing the flooding frequency of GBR. These adaptations will typically increase the power consumption of the nodes. Therefore, we do not want to constantly use this configuration, but use a low power configuration able to handle less data, when the cows are not being milked. On a smaller scale, we can look at the behaviour of individual nodes. The nodes attached to the cows can be stationary or mobile depending on the activity of the cow. Independent of whether the cows are being milked of not, these changing dynamics (locally)

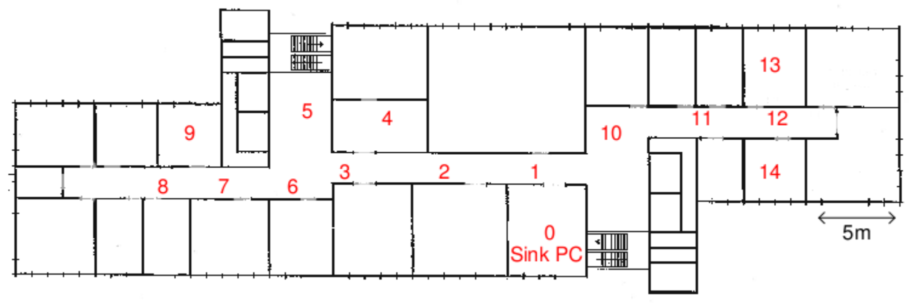


Figure 4.1: Deployment of 15 static TelosB nodes

change the behaviour of the network. When moving, a larger interference is observed and communication to neighbouring nodes is less reliable. Adapting parameters, such as the transmission power, can positively influence the reliability of communicating to neighbouring nodes and prevent an increase in packet loss while the cow is walking.

Office monitoring From monitoring the locations of persons in a building we could, for instance, automatically control the lights in every room of the building based on the presence of persons, potentially reducing the power consumption compared to the manual (de-)activation of lights by the employees. We can similarly imagine control of temperature, humidity or other environmental aspects. We consider a deployment with a static backbone consisting of 15 TelosB nodes [68], which monitor light intensity and forward packets to the sink, as shown in Figure 4.1. To monitor the location of employees in the building, a BSN node is carried by employees, in our case 5 persons, when they start working. They use the parent node to which they communicate, as an indicator of their location, and communicate it to the sink every 10 seconds. For this deployment we want an end-to-end packet delivery ratio of at least 50%, while network lifetime is optimized.

Nodes use the Low-Power-Listening (LPL) MAC [38], available in TinyOS [70], a basic asynchronous MAC protocol which reduces idle listening by sampling the medium at a given interval. We furthermore use a tree-based routing approach where the static nodes have a fixed parent. Mobile nodes determine their parent (to forward packets to and as an indication for their location) by iteratively broadcasting a request for hop-count information from the static nodes in range, and subsequently select the lowest hop-count neighbour. The controllable parameters are the sampling interval of the MAC protocol, the frequency at which the mobile nodes check their parent for the routing protocol, and the transmission power of the radio.

Similarly as for the cow-health monitoring scenario, the dynamics of employees in the office monitoring scenario show several interesting changes for which we could pro-actively reconfigure. Whether the monitored employee is static or mobile, impacts the mobility of the node attached to the person. The topology, and thereby neighbouring nodes to which communication can take place, changes over time. The request frequency of the routing protocol is an obvious candidate to increase in case the person starts to walk and updates of routing parents are needed. If the person is not walking, less effort has to be put in finding a new parent and the configuration should aim at saving power instead. Furthermore, there is an obvious daily pattern of office hours and non-office hours. Outside office hours no or just a limited number of employees need to be monitored, which drastically reduces the amount of data that is communicated within the WSN. As for the cow-health monitoring, with the knowledge of a very low data amount communicated in the network, adapting parameters to trade off the amount of data that can be handled in the network for a reduction of power, such as the sampling interval of the MAC protocol, is possible. This can significantly reduce the power consumption, while the packet-loss constraint is still met.

4.2 Method Details

In this section, we introduce our reconfiguration approach which pro-actively adapts protocol parameters at run-time to ensure the required network QoS is provided in a dynamic heterogeneous WSN. We describe the details of our pro-active approach and discuss the integration into the two illustrative scenarios given in the previous section. Section 4.2.1 describes the steps involved for designing and using our pro-active reconfiguration approach. It gives an overview of the design-time steps, discussed in more detail in Section 4.2.2, the deployment-time steps, Section 4.2.3, and run-time steps, Section 4.2.4. The integration of the approach for the two scenarios are analysed, and compared to current (re-)configuration approaches, in the next section.

4.2.1 Overview

Figure 4.2 gives an overview of the design-time, deployment-time and run-time steps involved with the construction of our pro-active reconfiguration approach. The basic concept is as follows. Instead of using a single configuration during the entire run-time (worst-case) or adapt the configuration once the network QoS is insufficient (re-active), we define a set of configurations from which the most appropriate is selected based on expected dynamics in the network. By observing events that indicate a change in the dynamics of the network, the configuration is adapted to anticipate for the impact of these changed dynamics. In other words, the run-time of every node is partitioned into a finite number of *modes* of operation which classify the dynamics experienced. Every mode has one associated

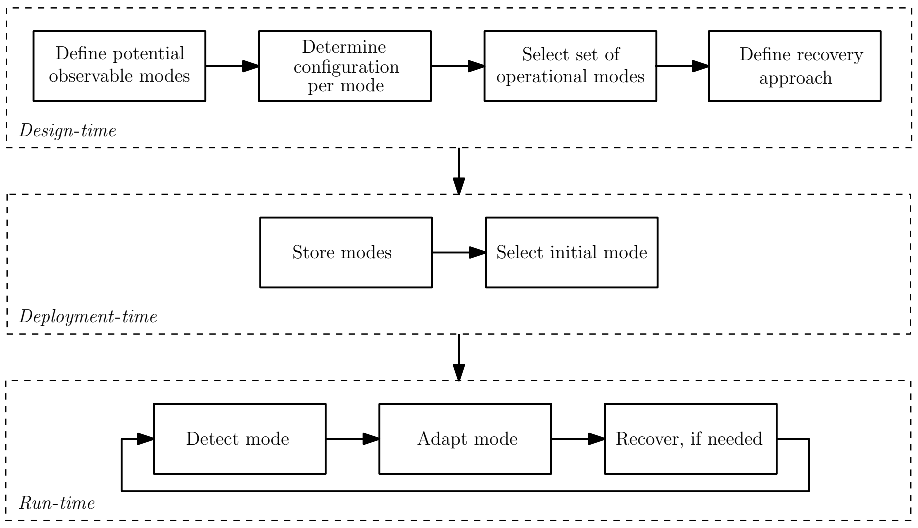


Figure 4.2: Steps involved in pro-active reconfiguration

configuration. These modes are defined at design-time and stored on the nodes during deployment. At run-time, the approach ensures that every node can detect the mode in which it should operate. These individual steps of our approach are discussed in detail, using the two illustrative scenarios introduced before, in the remainder of this section.

4.2.2 Design-time Steps

At design time, we perform the following four steps to instantiate the pro-active approach for a given application scenario.

Define potential observable modes The first step is to define potential modes by analysing the dynamics of the deployment and application scenario. We use a hierarchical approach to define the different modes in which the network could operate. First, we identify expected run-time situations which significantly differ in the dynamics experienced by the entire network, i.e., *network modes.* Network modes can, for example, be day and night. Within these network modes, every node has its own *node mode*, classifying the dynamics of the individual node. Typical node modes are, for example, whether a node is moving or stationary. At any moment in time, the network is operating in one of its potential network modes, and the node in one of its node modes. The node modes for a given node can potentially differ per network mode, e.g., a node might experience different kind of dynamics during the day compared to during the night. From experience with actual deployments we see that modes can often easily be derived using the designer's high-level understanding of the WSN deployment and the target application. Detailed information on, for example, the internal functionality of the network protocols, is often not required.

To allow run-time switching between modes, we require the beginning of a defined mode to be observable at run-time by the detection of a predefined event. Observable events can be related to a large variety of information, for example, sensor information from one or more nodes, the amount of traffic observed and/or the current time. For example, significant changes in accelerometer data can indicate the start of a person walking. The beginning of a node mode is required to be observable by every node itself. The beginning of a network mode is required to be detected at a central location, the sink (potentially using information from other nodes), allowing a coordinated change of the network mode to be initiated by this central location, as explained later. Observing the start and end of office hours can, for example, be done by monitoring the wall-clock time of the PC attached to the sink.

When we look at the cow-monitoring scenario, we can identify at least two interesting, run-time detectable, network modes; milking and not milking. As milking periods start and end every day around the same time, observing the beginning of both network modes based on the wall-clock time is the easiest solution. An alternative would be to, for example, use a dedicated sensor informing the sink as soon as the milking parlor is used. While in one of these network modes, every cow can be either walking or stationary, which classify as two node modes. It is expected that during the milking network mode, the majority of the cows is in the walking node mode. For the (mobile) nodes to detect whether the monitored cow is changing its posture from walking to stationary and vice versa, we can use thresholding on the accelerometer values, obtained from the BSN node. For the office monitoring scenario, the obvious daily pattern of office hours and non-office hours results in two potential network modes. We can furthermore define two node modes: walking and stationary. For detecting modes, the same approach as for the cow-health monitoring scenario is used. Network modes are observed at the sink based on the wall-clock time, and the nodes modes are detected using accelerometer data.

For the integration of the pro-active approach into the two scenarios, we consider the modes described before, but one could think of distinguishing even more modes. For the cow-monitoring scenario we could, for example, think of additional modes related to the activity of the cow, e.g., standing, lying or running. For the office-monitoring scenario, one might also be interested in reconfiguring for the case that, for example, many persons are in a single room.

Determine configuration per mode After analysing the dynamics of the scenario and defining the possible modes of operation, we determine a suitable configuration for every combination of network mode and node mode, simply referred to as the mode. For every mode we determine a suitable configuration at design-time, where we rely on existing techniques, such as analytical approaches [16, 22], simulations [37, 44], and experience with practical WSN deployments. The configuration consists of defining a value for the controllable parameters of the protocol stack. This can include the parameters of complementary run-time adaptation strategies, such as our re-active reconfiguration approach. One could think of increasing the speed on the re-active approach, and underlying service, with increased dynamics, which improves on the efficiency compared to having the fast speed during the entire run-time of the WSN. Some situations may also require the re-active approach to be completely suspended, such as short periods of very high dynamics where the speed of re-active reconfiguration cannot be set high enough and we have to rely on a fixed non-adaptive configuration instead. While our approach is independent of the design-time analysis techniques and controllable protocol parameters considered, they are important for the overall improvement obtained from our pro-active reconfiguration approach.

Similarly as the hierarchical definition of network and node modes, we consider two kinds of controllable parameters. A *global parameter* is set consistently for every node in the network, while a *local parameter* can be set independently for every node in the network. A consistent value is needed for global parameters for, for example, the proper functioning of a protocol, such as a MAC protocol. In the cow monitoring scenario, for instance, we consider the radio transmission power to be a local parameter, while the TDMA slot-size and sleep-time, and GBR flooding interval are global parameters. In the office monitoring scenario, the parent request interval of the mobile nodes and the transmission power are local parameters, while the MAC protocol sampling interval is a global parameter. Since changing a global parameter affects the behaviour of every node in the network it typically has a larger impact on the QoS compared to changing a local parameter. On the other hand they require more time and effort to change. A change of global parameter should be coordinated and synchronized to maintain proper functioning. We keep this in mind with the selection of a configuration, by only adapting global parameters in response to (infrequent) network mode changes. As network mode changes, and thereby potential changes to global parameters, are initiated at a central location, a coordinated and synchronized adaptation approach is possibly at run-time, as discussed later.

For the cow-health monitoring scenario, the configuration per mode is selected with the use of simulations. We implemented the scenario in OMNeT++ [44], a discrete-event simulation environment, with the use of MiXiM [37], a modeling framework created for wireless networks. We simulated with different parameter values to get an understanding of the impact of changing different parameters. Table 4.1 shows the values for the parameters resulting from the configuration-space exploration to get a packet-loss lower than 20%, while lifetime is optimized, for the different possible modes. Note that we distinguish different node modes for the static and mobile nodes. For the static nodes we use only a single node mode and retain the same local parameter values at all time. The first column shows the

Table 4.1: Configurations for cow-health monitoring scenario

Network mode	-	*Not milking*	*Not milking*	*Milking*	*Milking*
Node mode (mobile nodes)	-	*Stationary*	*Walking*	*Stationary*	*Walking*
Node mode (static nodes)	-	*Static*	*Static*	*Static*	*Static*
TDMA slot-size	0.1 s	0.05 s	0.05 s	0.1 s	0.1 s
TDMA sleep-time	0 s	10 s	10 s	0 s	0s
GBR update interval	30 s	60 s	60 s	30 s	30 s
TX power (mobile nodes)	-10dBm	-10dBm	-5dBm	-15dBm	-15dBm
TX power (static nodes)	-15dBm	-15dBm	-15dBm	-15dBm	-15dBm

Table 4.2: Configurations for office monitoring scenario

Network mode	-	*Office*	*Office*	*Non-office*	*Non-office*
Node mode (mobile nodes)	-	*Stationary*	*Walking*	*Stationary*	*Walking*
Node mode (static nodes)	-	*Static*	*Static*	*Static*	*Static*
LPL sampling interval	0.5 s	0.5 s	0.5 s	2 s	2 s
Request interval (mobile nodes)	3 s	60 s	3 s	60 s	3 s
TX power (mobile nodes)	-15 dBm	-15 dBm	-10 dBm	-15 dBm	-10 dBm
TX power (static nodes)	-15 dBm	-15 dBm	-15 dBm	-15 dBm	-15 dBm

values that we would use when no reconfiguration approach is applied, but just a single static (worst-case) configuration is used over time. This single configuration approach is the current state-of-the-art as has been implemented in the WASP project, and later used as one of the references for performance analysis of the pro-active approach. With a change to the milking mode, parameters are adapted to anticipate for the increased amount of data communicated in the network. In this case, the TDMA slot-size is increased and sleep-time reduced. This allows more data to be communicated, but reduces the time spent by the radio in the a low-power idle mode. Furthermore, to anticipate for the changing topology, the GBR update interval is reduced. In the milking mode, we observe that a lower transmission power is sufficient compared to the not-milking mode, as cows are typically closer together. During the milking mode a change of parameters is found to be not needed when the activity of the cow changes. While a cow is not being milked, an increase of transmission power is found to be useful when a cow starts to walk to compensate for increased interference.

For the office monitoring scenario, we determined the parameter values to use based on experimenting with the actual deployment. With the deployment in place we determined the impact of changing the parameters on the packet-loss and network lifetime. This resulted in the parameter value selection shown in Table 4.2. The first column of the table shows the single configuration used for a static configuration approach. With the switch between office hours and non-office hours we adapt the global Low-Power-Listening (LPL) sampling interval parameter, such that nodes are able to handle the amount of traffic experienced during that mode. While walking, the request interval is set to 3 seconds, which is enough to quickly respond to changing connectivity based on the distance between the nodes and walking speed. While stationary, nodes do not experience this change in connectivity and a high value of 60 seconds is selected to reduce the packet-load in the network. The transmission power is chosen such that a node can always connect to at least one static node in the monitored environment. When walking, we observed a higher interference on the sent packets, for which we compensate by a slightly higher transmission power. Note that we have chosen not to use this higher transmission power for the single configuration approach. This is because this higher transmission power is only found to be useful during the limited time a person is walking, which does not outweigh the disadvantage of having a high transmission power during all office hours. This shows an additional benefit of using a set of modes; it allows for a more fine-grained exploitation of performance trade-offs.

Select set of operational modes After classifying interesting node and network modes and determining the appropriate configuration per mode, a trade-off between the benefits of mode switching and the overhead incurred is made to select the set of modes to actually use at run-time. The local detection of a node mode is cheap, while a distributed detection where, for example, the sensor data of multiple sensors is collected to determine that a person starts to move, can be expensive. Similarly, adapting a local parameter is cheap, while the adaptation of a global parameter is relatively expensive. The frequency of events plays an important role in the overall configuration overhead. Due to the design-time definition of the modes, we can already analyse the (expected or maximum) frequency of events for a given deployment at design-time. In an office monitoring scenario we know there are two switches between office and non-office hours, while a person starting to walk occurs much more frequently (but maybe still infrequent enough to change a global parameter depending on the deployment). Besides reasoning, design-time techniques, such as simulations, can again be used to get an idea about the overhead and benefits of changing modes. As a result of this step, initially defined modes that were expected to be beneficial, but require too much overhead for limited benefit, are not considered for run-time reconfiguration.

For the cow-health monitoring scenario, the network mode is only changed for the milking period which happens two times a day at fixed times. The change of node mode can be completely observed locally and requires only limited run-time overhead for the detection of events indicating the mode change. In the case that the cow is milked, no detection is needed at all, as the derived configurations used when walking or stationary are equal (i.e., effectively there is only one node mode during the milking network mode). The expected overhead of the reconfiguring for the considered node modes is therefore negligible and we reconfigure for all of them. Similarly for the office monitoring scenario, a change of node mode is observed by every node individually, and the network mode change is observed by the sink and happens only two times per day. The benefits of reconfiguring for these modes is outweighing the overhead of detection and reconfiguration.

Define recovery approach In a WSN deployment, nodes may experience heavy external interference for a longer time, may loose connection and need to re-join the network. This may result in nodes missing a request to change the network mode. Nodes may also freshly join an existing network and have no knowledge of the current network mode. Incorrect assumptions on the network mode may result in the use of inconsistent global parameters leading to nodes not being able to effectively communicate, e.g., when using different radio channels, or to reduced efficiency of the communication, e.g., when using different radio sampling intervals. For the practical applicability of our approach, we define a recovery approach that recovers the appropriate network mode in the case a mismatch with other nodes arises. After being unable to communicate to any other node for a certain amount of time, nodes start recovering the network mode. In general, nodes can cycle through their different possible network modes and send packets to confirm the current network mode. In specific cases, nodes may have more efficient means to observe the global parameter values (and derive the current network mode) and avoid any communication or time overhead, for example by overhearing communication while changing radio frequencies in order to find the appropriate globally used radio channel. Discovering an erroneous assumed network mode is less obvious when nodes are still able to communicate, but less efficiently. A node may not be able to make a distinction between using a wrong mode or experiencing increased interference. In this case nodes should either regularly synchronize the assumed network mode with neighbouring nodes or observe it based on overhearing communication. In this work, we want to emphasize the need of a recovery approach for any practical application. The concrete approach is likely application specific. Additional experiments are needed to further explore the trade-offs involved in designing a recovery approach and the impact of the used protocol stack.

4.2.3 Deployment-time Steps

At the time the network is deployed, the configurations used for the collection of modes are stored on every node. Every mode is identified by a unique identifier. This allows easy referencing when reconfiguration of the node is externally triggered, such as with network mode changes. During deployment, the configuration is initially set for every node such that all nodes use a consistent network mode as is most appropriate for the dynamics experience at start-up of the WSN. Given the network mode, the initial node mode is set for every individual node. Note that even if the modes are selected incorrectly for the dynamics at deployment time, observing the run-time events will eventually lead to the nodes to be in the correct network and node mode.

4.2.4 Run-time Steps

Observable events allowing to detect the beginning of a mode are assumed to be identified at design-time with the selection of the potential modes. After observing these events, controllable parameters are adapted according to the configuration locally stored per mode. For the introduction of the pro-active approach we implicitly assume the configuration per mode to be static, but our approach is not limited to this. Additional run-time tuning of the configurations used per mode may be possible on top of the pro-active approach, for example by using a complementary re-active reconfiguration approach.

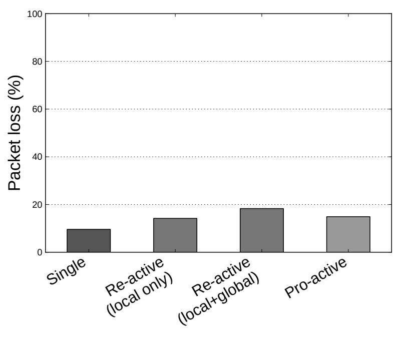
Node mode changes are detected by individual nodes themselves. Nodes may use information from other nodes, but often only need locally observed (sensor) information. A node detecting a node mode change can immediately adapt its local parameters based on the locally stored mode information without the need to inform or synchronize with other nodes. For example, a node may change its transmission power after observing significant changes in the acceleration of the node without any significant overhead. For detecting the events indicating the changes of the network mode, we rely on a single central location. Unless the detection of the required events is already part of the application, small procedures need to be written for the nodes to detect the events triggering node and network mode changes. Once decided to change the network mode, all nodes in the network are informed. As the parameter values used for every mode are locally accessible, the central location only needs to communicate the predefined unique identifier of the network mode to all nodes. Because of the small amount of information to communicate, we can resort to a simple and low overhead flooding approach, piggybacking on other packets if possible, to inform all nodes about the mode change. The flooding starts from the central location, the sink. The mode identifier is broadcast to all of its neighbours and every node receiving the information for the first time will broadcast it. With this procedure the information is disseminated to all nodes in an iterative fashion. After providing neighbouring nodes with the new mode identifier, the node adapts its (global and/or local) parameters to the values defined for the new network mode. For the cow-health monitoring scenario, we observed during simulations that a single flood is enough to inform all nodes about the change in network mode, due to the small size of the information packet, the used contention free MAC and large density of the network. Because of this and the fact that nodes remain connected to the network throughout the different network modes, a recovery approach is not required for this scenario. For the office scenario we observed that, a single broadcast is often not enough to inform all nodes about a changing network mode due to the lack of redundancy in the deployment and protocols. With more retransmissions we quickly converge to successfully informing all (connected) nodes. Based on these results, we conservatively flood five times to ensure a sufficiently small probability of connected nodes requires recovery. Since the network mode changes only two times a day, this repeated flooding induces a negligible amount of the total traffic load, i.e., 10 packets per node during the entire day. As nodes (re-)joining the network is an integral part of the application, we do require a recovery approach for these nodes to determine the globally used parameter values. We implemented the recovery approach where the node cycles through the different possible global parameter values, while requesting their parent for the actual network mode, as soon as they observe that they are entering the network after being disconnected. Experiments with this approach show that the time needed to find the network mode is typically less than a second when in range of the network. We did not implement a repeated check of the network mode for this scenario, accepting the small probability of a connected node experiencing limited connectivity if a mode change request is not received.

4.3 Performance Evaluation of Pro-active Reconfiguration

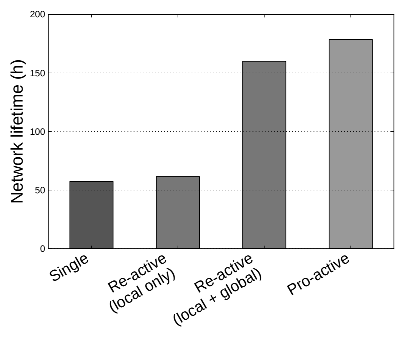
In this section, we discuss results of simulations for the cow-health monitoring scenario in Section 4.3.1 and experiments with the office building monitoring deployment in Section 4.3.2. For both scenarios we compare with a worst-case static configuration approach. We furthermore compare with a re-active approach and show situations in which pro-active reconfiguration allows more efficient QoS provisioning compared to the re-active approach.

4.3.1 Cow-health Monitoring Scenario

For the evaluation of the pro-active approach integrated in our cow-health monitoring scenario we use an implementation in the OMNeT++ simulator for determining suitable configurations per mode, as discussed in the previous section. We compare the pro-active approach with a single worst-case configuration and a re-active approach, in terms of the maximum end-to-end packet-loss of any of the nodes to the sink and network lifetime. The configuration for the single configuration approach and the modes for the pro-active approach are defined in Table 4.1. For the re-active approach we consider the same configurations, but a change between these configurations is performed based on re-active response to deviations in the packet-loss, instead of dynamic events. To meet the packet-loss constraint of 20%, the re-active approach aims at providing a packet-loss between a lower bound of 5% and upper bound of 15%. More specific, the transmission power is increased from -15 dBm to -10 dBm as soon as the observed packet-loss to the parent is higher than 15%. It is lowered as soon as the loss is below 5% to trade-off packet-loss for a reduction in power consumption. We respond to the periods of high data traffic, when milking, by monitoring the amount of data processed at the sink. Nodes are informed to adapt their global parameter values, TDMA slot-size, sleep-time and GBR update interval, to the values related to the milking network mode (see Table 4.1) when it is observed at the sink that at least 80% of the cows is sending at the rate related to walking.



(a)

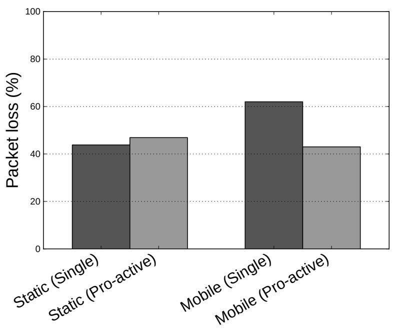


(b)

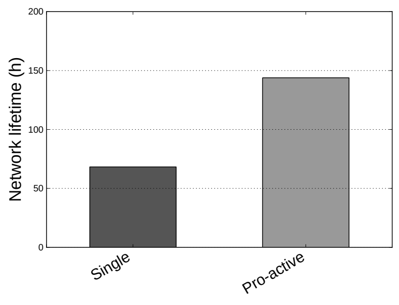
Figure 4.3: (a) Packet-loss and (b) network lifetime for cow-health monitoring scenario

The scenario is simulated for a day, in which two milking periods of two hours occur, for four approaches, i.e., a single (worst-case) configuration, re-active (adapting only the local parameters and adapting both the local and global parameters) and pro-active using a set of modes. To have statistically reliable results, every experiment was repeated 10 times. The maximum end-to-end packet-loss and network lifetime are shown in Figure 4.3. The results are the averages over all runs. The packet-loss is the percentage of the number of packets sent by the mobile nodes, but not received by the sink. The lifetime of a node is determined from its simulated power consumption and initial battery capacity. For the power consumption we focus on the main power consuming part of the node, the radio [49, 68]. Power consumption, in this case, depends on the TDMA schedule. We assume an initial battery capacity of the BSN nodes of 750 mAh at 1.5 V.

We observe that the maximum packet loss is lower than 20% for all approaches. A significant increase of network lifetime is observed when using the pro-active approach instead of the single static configuration, as the high-power worst-case configuration is not constantly used. When we look at the results of the re-active approach we see only a limited improvement, compared to the single configuration, if we only consider adapting local parameters. This is because the transmission power is just slightly adapted over time and does not allow a significant impact on the overall power consumption. As soon as global parameters are re-actively adapted as well, the network lifetime is increased significantly and is close to the pro-active approach. By adapting the global parameters, the time spent in the power-efficient idle mode of the radio is influenced. This shows the importance of adapting global parameters for this scenario. As the re-active approach is created to optimize the quality metrics averaged over a long time it is found to have a similar impact on the metrics due to relatively limited network dynamism. The most profound difference can be seen when looking in more detail at the



(a)



(b)

Figure 4.4: (a) Packet-loss and (b) network lifetime for office monitoring scenario

times that network dynamics occur. With the re-active approach, there is a delay between observing the high network load and the adaptation of the global parameters, i.e., TDMA schedule. In this period, packets are lost due to the limited capacity reserved by the TDMA schedule. The re-active approach first tries to compensate for a high packet-loss by increasing the transmission power, which increases power consumption, while the solution lies in the adaptation of the TDMA schedule. What we furthermore see is that the coordinated change of the global parameters becomes less efficient due to the high traffic, and more than one retransmission is needed to successfully propagate the change. This increases the time until the required adaptation of the network mode. This shows another advantage of our pro-active approach; it anticipates and adapts for the high traffic just before it actually occurs, avoiding reconfiguration in the period in which coordinating a network mode change is more challenging and many packets can be lost. The difference between the re-active and pro-active approach in their behaviour around dynamic events is investigated and discussed in more detail for the office monitoring scenario in the next subsection.

4.3.2 Office Monitoring Scenario

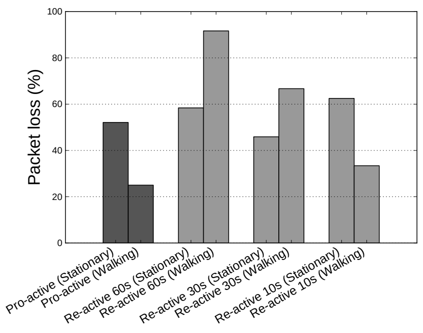
We implemented the office monitoring scenario, and integrated our reconfiguration approach, using TinyOS for both the (static) TelosB and (mobile) BSN nodes, as discussed in more detail in the previous section. We did a one day experiment using the modes and parameter values shown in Table 4.2. We assumed the office hours to be from 9 AM, after which most employees start working, until 7 PM, the time at which most employees have ended their working day.

Figure 4.4 shows the results of the pro-active approach compared to the worst-case single configuration approach, with respect to the maximum end-to-end packet-loss, where we differentiate between the static and mobile nodes, and network lifetime. The comparison with the re-active approach is later performed and discussed in detail. For the static TelosB nodes we assume the initial energy available to be 2 full batteries at 1500 mAh at 1.5 V and a smaller full 750 mAh at 1.5 V battery for the (mobile) BSN nodes. Due to the smaller battery, network lifetime is dominated by the mobile nodes. The results are based on experiments starting at 8 AM, before all monitored employees started working and 1 hour before the network mode switch, until 8 PM, after all employees stopped working and 1 hour after the network mode switch. For practical reasons we could not leave the nodes over night. For the remaining 12 hours we therefore determine the lifetime using the packet-loss and power consumption based on experiments with only the static network in place and assumed it would be similar during the entire night.

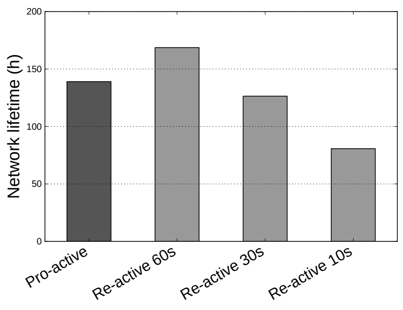
The results first of all show the feasibility of the reconfiguration approach to be implemented on resource constrained nodes and to be used in practice. The pro-active reconfiguration furthermore results in a significantly better network lifetime, while the average packet-loss is maintained, as constantly using a power inefficient worst-case configuration is avoided. For the mobile nodes, a reduction of packet-loss is observed due to the increase of transmission power when the node is mobile, compensating for increased interference.

To focus on the main difference between the re-active and pro-active approach, their response to network dynamics, we perform additional experiments. We constructed a scenario in which we control the moments in time at which dynamics occur. Using the same deployment of static nodes, a single mobile node moves in 1 minute from one side of the corridor to the other, stays there for 2 minutes, moves to the other side in 1 minute and stays there for 2 minutes. During the experiment, the mobile node sends packets to the sink once every 5 seconds, while the static nodes only forward these packets. The node starts to move and results are collected after an initialization period of 1 minute. For this experiment, we integrated both a pro-active and re-active approach in the routing protocol, which provides the parent to use for communication. The pro-active approach adapts the local parameters to the walking mode, i.e., the radio transmission power is increased to -10dBm and the routing request interval reduced to 3 seconds (see Table 4.1), as soon as significant changes in acceleration data are observed. The re-active approach adapts to the configuration as used in the walking mode as soon as a change of parent is observed. If no change of the parent is observed in the last 30 seconds, the activity of the protocols is relaxed by reducing TX to -15dBm and the interval to 60 seconds. With our experiment we focus on local events and retain the same global parameters, i.e., an LPL sampling interval of 500 milliseconds.

Figure 4.5 shows the packet-loss and network lifetime of the mobile node for the different activities, i.e., walking and stationary, when using the pro-active approach compared to the re-active approach, where we vary the time the re-active approach spends on the local analysis of the performance, i.e., a request interval of 10, 30 and 60 seconds. With the re-active approach, delay in adaptation



(a)



(b)

Figure 4.5: (a) Packet-loss and (b) network lifetime for the mobile node in the small-scale experiment

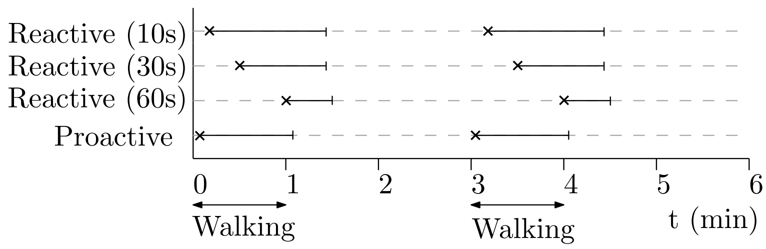


Figure 4.6: Adaptation responsiveness

results in nodes using an unreachable parent. For the re-active approach we can therefore see a clear trade-off between the responsiveness to topology changes, which is reflected in the packet-loss when walking, and network lifetime. Putting more effort in observing the local performance, during the entire run-time, results in a better responsiveness to dynamic events, but also requires more power, and thereby lifetime, during the period in which no dynamics occur. With the pro-active approach we actively observe the dynamics and this trade-off does not need to be made, resulting in both a lower packet-loss (when walking) and longer network lifetime.

To explain these results we take a closer look at the behaviour of the re-active and pro-active approach during this experiment. Figure 4.6 shows the reconfiguration moments in time of the different adaptation approaches. While the pro-active approach accurately adapts during the times it is required, using a re-active approach introduces a delay for the adaptation. After the node starts moving a delay arises until a parent change is observed. After the period of walking, there is still a small delay due to the introduced interval to be sure that the topology stopped changing. Reducing this interval can reduce the delay, but can result in a more frequent, unnecessary, adaptation for slow topology changes and even instability. For the short and predictable dynamics with a fairly large impact on the QoS, such as the periods of mobility in this small-scale experiment, pro-active reconfiguration allows a more efficient QoS provisioning compared to a re-active strategy.

4.4 Related Work

Our pro-active run-time reconfiguration approach differentiates itself from existing related work by the combination of its (i) pro-active approach as opposed to the common re-active approach, (ii) use of both local and global parameters and (iii) independence of protocol stack.

Existing run-time reconfiguration approaches in the area of WSN react to the impact on QoS caused by changes in dynamic behaviour. With the use of run-time techniques, such as feedback control [12, 26, 59, 63, 81], or machine learning [18, 71], these approaches converge to parameter values suited for the current situation. This makes them generally applicable, but ignores the additional knowledge we gain from the actual deployment to be reconfigured. We introduce a generic approach that can be instantiated to exploit the knowledge of the dynamics of a particular scenario, complementary to these re-active strategies.

In our approach, we consider adapting both local and global parameters, as well as the issues involved in coordinating and maintaining consistent network modes, and global parameters. Nodes (frequently) adapt their local parameters in a distributed manner, while we take a centralized approach to coordinate (infrequent) changes of global parameters. With approaches, such as a re-active reconfiguration approach, the (maximum) amount of reconfigurations and thereby the overhead involved with changing network parameters is unpredictable. With our method, the overhead of the coordination can be assessed and kept under control with design-time knowledge of the frequency of events.

While many reconfiguration approaches depend on specialized adaptive protocols, our approach considers the existing protocol stack as a black box with an interface of changeable protocol parameters. This is common for design-space exploration approaches which are used to find a single set of parameter values [16, 22]. It makes our method independent of the used protocol stack and allows us to change parameters for different protocol layers at the same time, increasing the applicability for new and existing WSN deployments. The adaptation techniques used by existing approaches are not generally applicable to other protocols and induce a (possibly large) run-time overhead. The use of design-time defined modes results in a very low run-time overhead for reconfiguration, because we only need a lightweight detection of the mode to use and a switching mechanism for informing on a changing network mode. This makes it very well suitable for integration in diverse practical WSN deployments. The results of the integration for a small-scale cow-health demonstrator for the WASP project are discussed in [73].

In our approach, we explicitly exploit a-priori knowledge and analysis of the application scenario for pro-active reconfiguration. In the area of real-time embedded systems the benefits of using a pro-active approach is already recognized. In [19], scenarios, related to what we call a mode, are defined at design-time in order to reduce resource consumption by the system by predicting in which scenario the system operates.

Finally, the integration of a recovery approach is emphasized for the use in any practical deployment. The need of such a recovery approach is recognized [7, 62], but its current application in practice induces a large overhead as complete chunks of code need to be communicated and/or stored on the nodes. Compared to approaches as described in [7, 62] we require a smaller scale recovery of only a single network mode identifier, thereby limiting the overhead involved with the recovery.

4.5 Summary

In this chapter, we introduced a pro-active run-time reconfiguration approach for dynamic heterogeneous WSNs. This approach explicitly exploits a-priori knowledge of the application scenario dynamics to guide a reconfiguration process.

At design time, dynamics affecting the behaviour of the nodes in the network are identified. Given these dynamics, we define multiple modes in which a node can operate. We use a hierarchical approach where we differentiate between the network mode, equal for every node in the network, and the mode of operation of the node, which can be different for every node. A changing mode indicates changing dynamics of one or more nodes that will potentially impact the network QoS. Every change of mode is assumed to be observable at run-time by an event, such as sensor readings or wall-clock time. Using existing design-time analysis techniques, a suitable configuration, defined by the values of controllable parameters, for every mode is determined. We distinguish parameters that can be adapted locally and those that should be considered globally. A change of node mode results in the adaptation of local parameters, while a change of network mode can also affect the value of global parameters. The modes are selected at design time keeping in mind the benefit of using them and the run-time switching overhead involved in the synchronized adaptation of global parameters. When deployed, the configuration used per mode is stored locally on the node and an initial mode is selected. At run-time, nodes locally observe the design-time defined events which signal the beginning of a node mode and adapt their configuration accordingly. A central location, such as a sink, observes and initiates the change of network mode by communicating to all nodes in the network, for which a simple, low overhead, flooding approach suffices. After being notified about a network mode change, nodes adapt their configuration, potentially including global parameters. Our reconfiguration approach is made robust for practical deployments by a recovery approach that allows nodes to determine the current network mode after, for example, (re-)joining the network or missing a network mode request.

We discussed the integration of our pro-active reconfiguration approach for cow-health monitoring and office monitoring scenarios. With simulations and experiments with an actual deployment, we have shown a significant positive effect on the network lifetime, while there is no negative effect on other QoS metrics, i.e., the packet-loss to the sink, when using a pro-active strategy compared to a static worst-case configuration. The experiments furthermore show the feasibility of the reconfiguration approach to be implemented on resource constrained nodes and to be used in practice. Due to the controlled number of global parameter changes and design-time definition of the modes, there is only a very low run-time overhead incurred by our approach. We also compared the performance of pro-active reconfiguration with a re-active reconfiguration approach, demonstrating their complementary nature. Re-active reconfiguration is a generally applicable reconfiguration strategy, but does not exploit any additional knowledge we have of the application scenario. We have shown that for predictable dynamics with a fairly large impact on the QoS, pro-active reconfiguration allows a more efficient QoS provisioning as it avoids phases in which the performance of the network is limited and re-active reconfiguration would require a significant amount of time to resolve the QoS error.

Chapter 5

Conclusions

Since the emergence of Wireless Sensor Networks (WSNs) around fifteen years ago, a significant amount of research has been done on specialized hardware and software. Many types of sensor nodes and protocols are readily available and are used for an increasing number of applications. Typical applications are the monitoring of elderly in an assisted living facility and controlling the environmental conditions in an office building based on the presence of employees. With the increased practical applicability of WSNs, end-user expectations on the performance are increasing and the WSN deployments get more complex. Different types of nodes are used and work together in the same WSN, resulting in heterogeneous networks. Furthermore, WSNs are integrated in more and more dynamic environments, where events such as moving persons and external interference have a fluctuating impact on the behaviour of the network. This makes the task of Quality-of-Service (QoS) provisioning, where the network is configured such that the application is efficiently and accurately executed, an increasingly important and challenging field of research. QoS provisioning for WSNs that are dynamic and heterogeneous is a huge and rather unexplored challenge.

This thesis provides an efficient QoS provisioning approach for dynamic heterogeneous WSNs. We aim at WSNs for monitoring applications consisting of both static and mobile nodes. Nodes with different characteristics may be used and can show significant dynamic behaviour. The dynamic behaviour of the WSN and application requires run-time reconfiguration to support efficient QoS provisioning. By a combination of re-active and pro-active run-time reconfiguration techniques, controllable parameters are adapted such that the network provides sufficient QoS to successfully perform the required application. To allow distributed reconfiguration decision making, we introduce and use an efficient generic distributed service to provide nodes with local estimates of network metrics. The network metrics are defined by a converging recursive equation expressed in terms of information exchanged with neighbouring nodes in the network. The validity and applicability of the individual parts of the QoS provisioning approach are confirmed with simulations and experiments with practical deployments. Practical experiments furthermore demonstrate that the approach can be implemented on resource constrained nodes. Using our QoS provisioning strategy, a significant improvement in the efficiency of QoS provisioning is shown compared to existing state-of-the-art configuration approaches, e.g., delivery ratio constraints are satisfied with a significantly longer lifetime of the network.

The rest of this chapter gives an overview of the contributions made in this thesis and discusses interesting aspects to consider for future work.

5.1 Contributions

In Chapter 2, we introduced a generic distributed service that allows nodes to accurately estimate network metric information in dynamic heterogeneous wireless sensor networks. To instantiate the service, a recursive local update procedure is defined that converges to a fixed point representing the desired metric. The recursive procedure is defined in terms of information that is locally measured and provided by neighbouring nodes. For several instantiations of our service, we showed that the iterative update function is stable and converging. Controlled n-hop forwarding is used to communicate information across asymmetric links, which are often found to be present in heterogeneous networks. To maintain accuracy under dynamic changes, the information dissemination is repeated at a given interval to allow nodes to keep estimates up-to-date. This interval is the main parameter to influence the trade-off between accuracy and overhead of the service. It is set in accordance to the amount and speed of dynamics in the deployment. With extensive simulations and experiments with actual deployments, we explored this trade-off and showed a significant increase in accuracy of the estimated network metric compared to the typically used local broadcasting approach. We showed the integration into various protocol stacks requiring different kinds of network metric estimates.

In Chapter 3, we introduced a re-active reconfiguration method that actively maintains the required QoS of the network using a distributed feedback control approach. Nodes adapt their parameters based on observed differences between the current QoS level (as estimated with the service) and QoS required by the end-user. The approach is generally applicable in any dynamic scenario as it directly steers the network QoS the end-user is interested in. As multiple nodes can influence the network QoS, nodes collaborate to efficiently resolve any QoS error. This can imply that one node changes its behaviour more actively compared to others. For example, when considering to meet a packet-loss constraint, while network lifetime is optimized, nodes with a low expected remaining lifetime should spend less power on reducing packet-loss compared to longer lifetime nodes on the same path. We have shown how to instantiate our distributed service to collect network QoS information needed for efficient collaboration. Nodes reconfigure by adapting controllable parameters of which the impact on the QoS is predicted using an adaptive model. This model is updated at run-time using information on observed impact of reconfigurations and the current QoS, to maintain accuracy of the predicted impact. Step-response analysis showed that the distributed feedback control approach is stable, if reconfiguration decisions are based on accurate feedback. The speed of the controller should therefore be set in line with the speed of feedback propagation performed by the service. Experiments with an actual deployment showed that it is possible to implement the controller on resource constrained nodes and to provide QoS for a dynamic and heterogeneous WSN deployment. Compared to a worst-case single configuration, and a local adaptation approach which does not take network QoS knowledge into account, we are able to maintain the same required packet delivery-ratio, while significantly extending the lifetime of the network.

In Chapter 4, we introduced a pro-active reconfiguration method for QoS provisioning. Complementary to the re-active approach, this approach explicitly exploits a-priori knowledge of the application scenario to guide the reconfiguration process. At design-time, multiple modes in which a node can operate are identified and observable events that signal the start of a mode are defined. For the definition of the modes we take a hierarchical approach. The mode of operation of the network, shared by every node in the network, defines situations affecting the dynamic characteristics of the entire network. Network modes can, for example, be day and night. Within the network mode, the mode of operation of the node, which can be different for different nodes, defines the dynamic behaviour of the node. Typical node modes are, for example, whether the node is mobile or stationary. A change of (node or network) mode indicates changing dynamics of one or more nodes, potentially impacting network QoS. Using existing design-time analysis techniques, a suitable configuration for every mode is determined. For our pro-active approach, we distinguish between parameters that can be adapted locally and those that should be considered globally. A change of global parameter should be coordinated and synchronized to maintain proper functioning. We keep this in mind with the selection of a configuration, by only adapting global parameters in response to (infrequent) network mode changes. As network mode changes, and thereby potential changes of global parameters, are initiated at a central location, a coordinated and synchronized adaptation approach is required at run-time. The modes to use at run-time are selected keeping in mind the benefit of using them and the run-time switching overhead involved in the synchronized adaptation of global parameters. At the time the network is deployed, the configurations for the defined modes are stored on the nodes and an initial configuration is selected. At run-time, nodes locally observe events which signal the change of a node mode and adapt the configuration accordingly. A centralized location, such as a sink, observes and initiates a change of network mode by communicating to all nodes in the network, for which a simple, low overhead flooding approach suffices. After being notified about a network mode change, nodes adapt their configuration, potentially including global parameters. Our reconfiguration method is made robust by using a recovery approach that allows nodes to determine the current network mode after, for example, (re-)joining the network or missing a network mode change request. We integrated our pro-active reconfiguration approach into practical cow-health monitoring and office monitoring scenarios. The use of a pro-active approach results in more efficient QoS provisioning, e.g., a significant positive effect on the network lifetime, while constraints on the packet-loss are still met. By comparing with a re-active approach, we have seen that being pro-active allows an early reconfiguration anticipating for network dynamics. It thereby avoids a phase in which the performance of the network is limited. Due to the controlled number of global parameter changes and design-time definition of the modes, the approach incurs only a very low run-time overhead.

5.2 Recommendations for Future Work

While this thesis provides a complete and efficient run-time reconfiguration strategy for the QoS provisioning problem for dynamic heterogeneous WSNs, there is room for extensions.

An interesting step to increase the applicability of our QoS provisioning approach is to construct an efficient calibration phase which relieves the designer from the potentially complex task or finding appropriate initial configurations and parameter values for the re-active and pro-active approaches. We currently rely on manual setting of the parameters using observations from simulations and small-scale deployments. This works fine for relatively small networks with a reasonable amount of heterogeneity. Especially when size and heterogeneity in the network increase, a simply strategy for the initial setting of the WSN is preferred which does not require the manual setting of every individual node. One could think of run-time techniques that, for example, require every node to scan its possible range of parameter values to search for interesting trade-offs and initial parameter settings.

The distributed service for network QoS estimation in our QoS provisioning approach assumes the update interval to be identical for every node in the network. A heterogeneous setting of the update interval significantly increases the design space, but may be beneficial for reducing the overhead of the service by making some paths update their network QoS faster than others. For example, for paths on which nodes do not regularly show changes in their behaviour less updates may be needed compared to paths with nodes that often affect the path QoS.

If we look at the re-active approach, it aims at keeping values of the constrained metrics between their respective lower and upperbounds. The relative distances of these bounds from the constraint are assumed to have been defined, but their selection involves making a trade-off between the optimization freedom and the probability of fluctuating metric values to occasionally violate the constraint. The defined distance between the bounds and the constraint could for example be influenced by the impact of the expected dynamics. The distance between the lower and upperbound is also an interesting aspect to investigate. It should be large enough to avoid the QoS to frequently be outside this range, but it should be low enough to allow exploiting the performance trade-offs. Exploring the trade-offs involved with selecting the lowerbound and upperbound for a particular deployment, and constraint, is an interesting subject.

The optimal speed of the re-active approach is influenced by various run-time aspects, such as path length and amount of external interference. These aspects impact the time needed to propagate network QoS information and thereby the maximum speed for a stable feedback controller. Currently, we consider using the path length of the end-to-end critical path to dynamically adapt the speed, allowing shorter paths to reconfigure faster than longer paths. One could think of various other aspects, such as the quality of links or amount of expected dynamism on a path, to influence the speed. More extensive analysis of the aspects influencing the optimal controller speed can give more insight in the possibility to automatically, and even more efficiently, set or adapt the controller speed at run-time.

For the pro-active approach, we rely on design-time analysis, mostly using the developers and/or end-user's knowledge of the application scenario, for the identification of modes. There is a probability of overlooking less obvious, but exploitable, modes. The more useful modes that are detected, the more efficient the use of a pro-active reconfiguration can be. More elaborate techniques for the automated selection of modes, for example, by design-time techniques such as simulations, or analysis of collected data, or the run-time identification of modes, are interesting subjects.

As a final recommendation, we suggest to further explore the relation between the re-active and pro-active approach. The re-active approach is generally applicable for any kind of application scenario, while the pro-active approach explicitly uses knowledge of the scenario. We discussed the possibility of the re-active approach using information of the scenario, similarly as done for the pro-active approach, to adapt the speed of the controller to the mode (or even suspend it for some time knowing that dynamic changes in a mode are too frequent to follow adaptively). Furthermore, the re-active approach can be used to tune the configurations used in every mode. A hierarchical organization can be used in which a re-active approach is used in every mode, and can potentially be adapted as soon as a mode changes. To which extent the two approaches can further benefit from each other, and an analysis of the trade-offs involved, given a particular application, are interesting subjects for future work.

5.3 Concluding Remarks

In this thesis, we provided techniques that can be adopted by WSN researchers and designers to help them construct efficient dynamic heterogeneous WSNs. With our techniques, we aim at the important task of providing a sufficient level of QoS for the entire network, allowing the successful execution of a given application. The contributions of this thesis can form the basis for future research, for example along the lines of the future work recommendations given above. Our efficient QoS provisioning paves the way for more practical WSN deployments and takes the use of sensor networks to aid us in our daily lives a step forward.

Bibliography

[1] ALwEN (Ambient Living with Embedded Networks) project website. http://www.alwen.nl. Last visited: June 2013.

[2] N. Baccour, A. Koubaa, M. Ben Jamaa, H. Youssef, M. Zuniga, and M. Alves. A comparative simulation study of link quality estimators in wireless sensor networks. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on*, pages 1–10, 2009.

[3] D. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Athena Scientific, 1997.

[4] D. Bertsekas, J. Tsitsiklis, and M. Athans. Convergence theories of distributed iterative processes: A survey. In *Lecture Notes in Control and Information Sciences*, volume 76, pages 107–139, 1986.

[5] M. Blagojevic, M. Nabi, M. Geilen, T. Basten, T. Hendriks, and M. Steine. A Probabilistic Acknowledgment Mechanism for Wireless Sensor Networks. In *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*, pages 63–72, 2011.

[6] P. Boonma and J. Suzuki. Monsoon: A coevolutionary multiobjective adaptation framework for dynamic wireless sensor networks. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pages 497–506, 2008.

[7] S. Brown and C. J. Sreenan. Software Update Recovery for Wireless Sensor Networks. In *Proceedings of the conference on Sensor Applications, Experimentation, and Logistics*, volume 29 of *SAEL'10*, pages 107–125, 2010.

[8] BSN website, ICL London. http://vip.doc.ic.ac.uk/bsn/m621.html. Last visited: June 2013.

[9] A. Cerpa, J. Wong, M. Potkonjak, and D. Estrin. Temporal properties of low power wireless links: modeling and implications on multi-hop routing. pages 414–425, 2005.

[10] J.-H. Chang and L. Tassiulas. Energy conserving routing in wireless adhoc networks. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 22 –31, 2000.

[11] J.-H. Chang and L. Tassiulas. Maximum lifetime routing in wireless sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 12(4):609–619, 2004.

[12] O. Chipara, Z. He, G. Xing, Q. Chen, X. Wang, C. Lu, J. Stankovic, and T. Abdelzaher. Real-time power-aware routing in sensor networks. In *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on*, pages 83 –92, 2006.

[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[14] J. Du, W. Shi, and K Sha. Asymmetry-aware link layer services in wireless sensor networks. *Journal of Embedded Computing*, 3:141–154, 2009.

[15] EAGER (An In-Home Health Alert System with Remote Care Coordination) project website. http://www.eldertech.missouri.edu/docs/Skubic - EAGER.html. Last visited: June 2013.

[16] K. P. Ferentinos and T. A. Tsiligiridis. Adaptive design optimization of wireless sensor networks using genetic algorithms. *Computer Networks*, 51(4):1031 – 1051, 2007.

[17] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo. TinyOS TEP 123: The Collection Tree Protocol. August 2006.

[18] A. Forster. Machine Learning Techniques Applied to Wireless Ad-Hoc Networks: Guide and Survey. In *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on*, pages 365 –370, 2007.

[19] S. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. De Bosschere. System-scenario-based design of dynamic embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 14(1):1–45, 2009.

[20] S. Hedetniemi and A. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.

[21] W.B. Heinzelman, A.P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *Wireless Communications, IEEE Transactions on*, 1(4):660 – 670, oct 2002.

[22] R. Hoes, T. Basten, C.-K. Tham, M. Geilen, and H. Corporaal. Quality-of-service trade-off analysis for wireless sensor networks. *Performance Evaluation*, 66(3–5):191 – 208, 2009.

[23] E. Hyytiä and J. Virtamo. Random waypoint mobility model in cellular networks. *Wireless Networks*, 13:177–188, 2007.

[24] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *Networking, IEEE/ACM Transactions on*, 11(1):2–16, 2003.

[25] S. C. Kleene. On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3(4):150–155, 1938.

[26] S. Kogekar, S. Neema, B. Eames, X. Koutsoukos, A. Ledeczi, and M. Maroti. Constraint-guided dynamic reconfiguration in sensor networks. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, IPSN '04, pages 379–387, 2004.

[27] D. Kotz, C. Newport, and C. Elliott. The mistaken axioms of wireless network research. Technical Report TR2003-467, Dartmouth College, July 2003.

[28] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 64–75, 2005.

[29] S. Lin, J. Zhang, G. Zhou, L. Gu, John A. Stankovic, and T. He. Atpc: adaptive transmission power control for wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 223–236, 2006.

[30] C. Lokhorst, P.H. Hogewerf, R.M. de Mol, R. Verhoeven, M. Steine, J.J. Lukkien, and M. Bennebroek. Wireless sensor application for dairy cow activity monitoring. In *Proceedings of the 5th European Conference on Precision Livestock Farming*, ECPLF 11, pages 17–26, 2009.

[31] A. M. Lyapunov. The general problem of the stability of motion. *International Journal of Control*, 55(3):531–534, 1992.

[32] M.K. Maggs, S.G. O'Keefe, and D.V. Thiel. Consensus Clock Synchronization for Wireless Sensor Networks. *Sensors Journal, IEEE*, 12(6):2269–2277, 2012.

[33] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 88–97, 2002.

[34] P. J. Marron, S. Karnouskos, D. Minder, and A. Ollero. *The Emerging Domain of Cooperating Objects*. Springer, 2011.

[35] T. Melodia, M. C. Vuran, and D. Pompili. The state of the art in cross-layer design for wireless sensor networks. In *Proceedings of the Second international conference on Wireless Systems and Network Architectures in Next Generation Internet*, EURO-NGI'05, pages 78–92, 2006.

[36] A. Milenković, C. Otto, and E. Jovanov. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Computer Communications*, 29(13-14):2521–2533, August 2006.

[37] MiXiM website. mixim.sourceforge.net. Last visited: June 2013.

[38] D. Moss and P. Levis. BoX-MACs: Exploiting Physical and Link Layer Boundaries in Low-Power Networking. Technical report, Stanford Information Networks Group Technical Report, 2008.

[39] L. Mottola, G. P. Picco, and A. A. Sheikh. FiGaRo: fine-grained software reconfiguration for wireless sensor networks. In *Proceedings of the 5th European conference on Wireless sensor networks*, EWSN'08, pages 286–304, 2008.

[40] M. Nabi, M. Blagojevic, T. Basten, M. Geilen, and T. Hendriks. Configuring multi-objective evolutionary algorithms for design-space exploration of wireless sensor networks. In *Proceedings of the 4th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, PM2HW2N '09, pages 111–119, 2009.

[41] E. F. Nakamura, A. A. F. Loureiro, and A. C. Frery. Information fusion for wireless sensor networks: Methods, models, and classifications. *ACM Comput. Surv.*, 39(3), 2007.

[42] D. Niculescu and B. Nath. Ad hoc positioning system (APS). In *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, volume 5, pages 2926–2931, 2001.

[43] R. Olfati-Saber, J.A. Fax, and R.M. Murray. Consensus and Cooperation in Networked Multi-Agent Systems. *Proceedings of the IEEE*, 95(1):215 –233, 2007.

[44] OMNeT++ website 2013. www.omnetpp.org. Last visited: June 2013.

[45] V. Pareto. Manuale di economia politica. piccola biblioteca scientifica, milan, 1906. translated into english by ann s. schwier (1971), manual of political economy. *Manual of Political Economy.*

[46] P. Park, P. Di Marco, C. Fischione, and K.H. Johansson. Modeling and Optimization of the IEEE 802.15.4 Protocol for Reliable and Timely Communications. *Parallel and Distributed Systems, IEEE Transactions on*, 24(3):550 –564, 2013.

[47] E.M. Petriu, N. D. Georganas, D.C. Petriu, D. Makrakis, and V.Z. Groza. Sensor-based information appliances. *Instrumentation Measurement Magazine, IEEE*, 3(4):31–35, 2000.

[48] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 95–107, 2004.

[49] A. Prayati, Ch. Antonopoulos, T. Stoyanova, C. Koulamas, and G. Papadopoulos. A modeling approach on the telosb wsn platform power consumption. *Journal on Systems and Software*, 83(8):1355–1363, 2010.

[50] A. Richards and J. How. A decentralized algorithm for robust constrained model predictive control. In *American Control Conference, 2004. Proceedings of the 2004*, volume 5, pages 4261 –4266, 2004.

[51] Roessingh Research and Development company website. http://www.rrd.nl. Last visited: June 2013.

[52] K. Romer and F. Mattern. The design space of wireless sensor networks. *Wireless Communications, IEEE*, 11(6):54 – 61, 2004.

[53] L. Sang, A. Arora, and H. Zhang. On link asymmetry and one-way estimation in wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 6(2):12:1–12:25, 2010.

[54] C. Schurgers and M.B. Srivastava. Energy efficient routing in wireless sensor networks. In *Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE*, volume 1, pages 357 – 361, 2001.

[55] K. Seada, M. Zuniga, A. Helmy, and B. Krishnamachari. Energy-efficient forwarding strategies for geographic routing in lossy wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 108–121, 2004.

[56] K. Srinivasan and P. Levis. RSSI is Under Appreciated. In *Proceedings of the 3th International Workshop on Embedded Networked Sensors*, EmNets '06, 2006.

[57] J. A. Stankovic, Q. Cao, T. Doan, L. Fang, Z. He, R. Kiran, S. Lin, S. Son, R. Stoleru, and A. Wood. Wireless sensor networks for in-home healthcare: potential and challenges. *Proceedings of High Confidence Medical Device Software and Systems Workshop*, 2005.

[58] M. Steine, M. Geilen, and T. Basten. Distributed Maintenance of Minimum-cost Path Information in Wireless Sensor Networks. In *Proceedings of the 6th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, PM2HW2N '11, pages 25–32, 2011.

[59] M. Steine, M. Geilen, and T. Basten. A Distributed Feedback Control Mechanism for Quality-of-Service Maintenance in Wireless Sensor Networks. In *Digital System Design, Architectures, Methods and Tools, 2012. DSD '12. 15th Euromicro Conference on*, pages 739–742, 2012.

[60] M. Steine, C. Viet Ngo, R. Serna Oliver, M. Geilen, T. Basten, G. Fohler, and J.-D. Decotignie. Proactive Reconfiguration of Wireless Sensor Networks. In *Proceedings of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '11, pages 31–40, 2011.

[61] I. Stojmenovic and X. Lin. Power-Aware Localized Routing in Wireless Networks. *Parallel and Distributed Systems, IEEE Transactions on*, 12(11):1122–1133, 2001.

[62] M. Strasser and H. Vogt. Autonomous and distributed node recovery in wireless sensor networks. In *Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, SASN '06, pages 113–122, 2006.

[63] Y. Sun, O. Gurewitz, and D. B. Johnson. RI-MAC: a receiver-initiated asynchronous duty cycle MAC protocol for dynamic traffic loads in wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 1–14, 2008.

[64] P. Suriyachai, U. Roedig, and A. Scott. A survey of mac protocols for mission-critical applications in wireless sensor networks. *Communications Surveys Tutorials, IEEE*, 14(2):240–264, 2012.

[65] M. Szczodrak, O. Gnawali, and L. P. Carloni. Dynamic reconguration of wireless sensor networks to support heterogeneous applications. In *DCOSS 2013. The ninth IEEE International Conference on Distributed Computing in Sensor Systems. Proceedings. IEEE*, pages 52 –61, 2013.

[66] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 214–226, 2004.

[67] T.T. Tay, I.M.Y. Mareels, and J.B. Moore. *High Performance Control*. Springer, 1998.

[68] TelosB Datasheet, Crossbow Inc. www.xbow.com. Last visited: June 2013.

[69] H Tian, J.A. Stankovic, L. Chenyang, and T. Abdelzaher. SPEED: a stateless protocol for real-time communication in sensor networks. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 46–55, 2003.

[70] TinyOS website. www.tinyos.net. Last visited: June 2013.

[71] S. Tomforde, I. Zgeras, J. Hähner, and C. Müller-Schloer. Adaptive control of sensor networks. In *Proceedings of the 7th international conference on Autonomic and trusted computing*, ATC'10, pages 77–91, 2010.

[72] Population ageing and development. Report of United Nations Department of Economic and Social Affairs, 2009. http://www.un.org/esa/population/publications/ageing/ageing2009.htm.

[73] C. Viet Ngo, R. Serna Oliver, and G. Fohler. SIMOSP: A Simple Mode Switch Protocol for Wireless Sensor Networks. Technical Report TR10 09, Technische Universtät Kaiserslautern, October 2010.

[74] WASP (Wirelessly Accessible Sensor Populations) project website. http://www.wasp-project.org. Last visited: June 2013.

[75] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 381–396, 2006.

[76] G. Wittenburg, K. Terfloth, F. L. Villafuerte, T. Naumowicz, H. Ritter, and J. Schiller. Fence monitoring: experimental evaluation of a use case for wireless sensor networks. In *Proceedings of the 4th European conference on Wireless sensor networks*, EWSN'07, pages 163–178, 2007.

[77] A. Woo, T. Tong, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 14–27, 2003.

[78] A. Wood, J.A. Stankovic, G. Virone, L. Selavo, Zhimin He, Qiuhua Cao, Thao Doan, Yafeng Wu, Lei Fang, and R. Stoleru. Context-aware wireless sensor networks for assisted living and residential monitoring. *Network, IEEE*, 22(4):26–33, 2008.

[79] F. Ye, A. Chen, S. Lu, and L. Zhang. A Scalable Solution to Minimum Cost Forwarding in Large Sensor Networks. In *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, pages 304–309, 2001.

[80] F. Ye, G. Zhong, S. Lu, and L. Zhang. GRAdient broadcast: a robust data delivery protocol for large scale sensor networks. *Wireless Networks*, 11(3):285–298, 2005.

[81] W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *Networking, IEEE/ACM Transactions on*, 12(3):493 – 506, june 2004.

[82] M. Younis and K. Akkaya. Strategies and techniques for node placement in wireless sensor networks: A survey. *Ad Hoc Networks*, 6(4):621 – 655, 2008.

[83] O. Younis and S. Fahmy. HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *Mobile Computing, IEEE Transactions on*, 3(4):366 – 379, 2004.

[84] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 1–13, 2003.

[85] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, MobiSys '04, pages 125–138, 2004.

[86] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele. ptunes: runtime parameter adaptation for low-power mac protocols. In *Proceedings of the 11th international conference on Information Processing in Sensor Networks*, IPSN '12, pages 173–184, 2012.

[87] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimisation, and Control*, pages 95–100. CIMNE, Barcelona, Spain, 2002.

Curriculum Vitae

Marcel Steine was born in Rotterdam, The Netherlands, on May 5, 1985. In 2006, he received a Bachelor's degree in Computer Science and Engineering from the Eindhoven University of Technology (TU/e). In 2008, he received a Master's degree (with honors) in Embedded Systems from the TU/e, for a project on predictable scheduling for multiprocessors performed at NXP Research, Eindhoven. During his Master program he visited the National University of Singapore for a project on quality-of-service analysis for sensor networks.
In January 2009, he started working towards a Ph.D. degree within the Electronic Systems group at the department of Electrical Engineering of the Eindhoven University of Technology. His research was in part funded by the European Commission through project WASP. The results of his research are presented in this thesis.

List of Publications

First Author

- M. Steine, M.C.W. Geilen, T. Basten. A Distributed Feedback Control Mechanism for Quality-of-Service Maintenance in Wireless Sensor Networks. In *15th Euromicro Conference On Digital System Design*, DSD '12, Proceedings, pages 739-742. Cesme, Izmir, Turkey, 5-7 September 2012. IEEE Computer Society Press, 2012.

- M. Steine, C. Viet Ngo, R. Serna Oliver, M.C.W. Geilen, T. Basten, G. Fohler and J.-D. Decotignie. Proactive Reconfiguration of Wireless Sensor Networks. In *14th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '11, Proceedings, pages 31-40. Miami, FL, USA, 31 October - 4 November 2011. ACM Press, 2011.

- M. Steine, M.C.W. Geilen, T. Basten. Distributed Maintenance of Minimum-cost Path Information in Wireless Sensor Networks. In *6th ACM International Workshop on Performance Monitoring, Measurement and Evaluation of Heterogeneous Wireless and Wired Networks*, PM2HW2N '11, Proceedings, pages 25-32. Miami, FL, USA, 31 October 2011. ACM Press, 2011.

- M. Steine, M.J.G. Bekooij and M. Wiggers. A Priority-based Budget Scheduler with Conservative Dataflow Model. In *12th Euromicro Conference on Digital System Design*, DSD '09, Proceedings, pages 37–44. Patras, Greece, 27-29 August 2009. IEEE Computer Society Press, 2009.

- M. Steine, M.C.W. Geilen, T. Basten. A Distributed Reconfiguration Approach for Quality-of-Service Provisioning in Dynamic Heterogeneous Wireless Sensor Networks. *Under review for journal publication.*

- M. Steine, M.C.W. Geilen, T. Basten. Distributed Estimation of Network Metrics in Dynamic Heterogeneous Wireless Sensor Networks. *Under review for journal publication.*

Co-author

- M. Blagojevic, M. Nabi, M.C.W. Geilen, T. Basten, T. Hendriks and M. Steine. A Probabilistic Acknowledgment Mechanism for Wireless Sensor Networks. In *6th IEEE International Conference on Networking, Architecture, and Storage*, NAS '11, Proceedings, pages 63–72. Dalian, Liaoning, China, 28-30 July 2011. IEEE Computer Society Press, 2011.
- C. Lokhorst, P.H. Hogewerf, R.M. de Mol, R. Verhoeven, M. Steine, J.J. Lukkien and M. Bennebroek. Wireless Sensor Application for Dairy Cow Activity Monitoring. In *5th European Conference on Precision Livestock Farming*, ECPLF '11, Proceedings, pages 17–26. Prague, Czech Republic, 11-14 July 2011.

Acknowledgments

Pursuing a PhD is something you cannot do on your own. This thesis would never been completed without the support of many colleagues, family and friends.

First of all, I like to thank my promotor, Twan Basten, and daily supervisor and copromotor, Marc Geilen. Twan has been a great source of inspiration and motivation during the years of my PhD, but also in the years before when he supervised my internship and master thesis. I really respect and appreciate his close involvement with the work of all his PhD's, even with his increasingly busy schedule. Our discussions greatly improved the details of my work and helped me to get a better understanding of the bigger picture. Twan, thanks for your invaluable guidance. I greatly appreciate all the work that went into my supervision. Thanks for creating the supportive, friendly and professional environment to pursue my PhD. I'm very glad to have had Marc as my daily supervisor. With his immense knowledge he guided me and helped me to improve my research and engineering skills. Marc, thanks for your patience and allowing me to drop by your office whenever I needed feedback on my work. Our discussions always provided me with renewed motivation, solutions and ideas for improvements. Thanks for showing me very nice mountainbike tracks in Limburg and that I need more practice to keep up with you next time.

I would like to thank Johan Lukkien, Koen Langendoen and Jean-Dominique Decotignie for being part of the reading committee and providing detailed comments on the draft version of my thesis. Thanks to Sonia Heemstra for being part of the defense opposition and Ton Backx for being the chairman for the defense.

My research was partly conducted within the European project WASP. I want to thank all the members of the WASP project, and in particular Johan Lukkien, Richard Verhoeven, Jérôme Rousselot, Jean-Dominique Decotignie, Cuong Viet Ngo, Ramon Serna Oliver and Gerhard Fohler, for the fruitful discussions and collaboration contributing to an important part of my thesis.

During my PhD, I had the pleasure to work in the Electronic Systems group. I would like to thank all of its members for creating the perfect working atmosphere. During regular group meetings, ongoing work and new ideas were discussed. I want to thank all the members of the NES group for their valuable contributions. In particular, I want to thank Majid Nabi and Milos Blagojevic.

We started to work on our PhD around the same time, on similar topics in a field which was relatively new within our group. Thanks for the very pleasant collaboration over the years. I furthermore want to thank the colleagues with which I had regular coffee breaks and lunches. The discussions we had during these times were always interesting and relaxing. In particular, I want to thank Raymond Frijns and Luc Vosters for their valuable contributions to these discussions, and the great times we had together during many game nights. I'm pleased to have Raymond and Luc assisting me as my paranymphs during the day of my defense.

I'm glad to have many friends who have been a great support during my PhD. Friends from the student tennis association Fellenoord with which I spent many hours in training, tournaments, competition and activities next to the tennis court. Friends I met during my study. Friends from Zeeland. Colleagues who became very good friends. Thanks for being there through hard times and for providing well needed distraction by spending great holidays, weekends, evenings, sports and parties together. While our future careers and personal lives may diverge, I will make sure to keep in touch.

Last, but certainly not least, my family. The ones that made me what I am and provide unconditional support. I especially want to thank my mother and sister for their endless love, and my father, who often told me how proud he was of my achievements, but unfortunately cannot see the end result of all the hard work. I dedicate this thesis to them.

Marcel Steine
September 2013