Technische Universiteit
Eindhoven
University of Technology

A Distributed Reconfiguration Approach for Quality-of-Service Provisioning in Dynamic Heterogeneous Wireless Sensor Networks

Marcel Steine, Marc Geilen and Twan Basten

ES Reports
ISSN 1574-9517

ESR-2013-02
20 June 2013

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems

http://www.es.ele.tue.nl/esreports
esreports@es.ele.tue.nl

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems
PO Box 513
NL-5600 MB Eindhoven
The Netherlands

A Distributed Reconfiguration Approach for Quality-of-Service Provisioning in Dynamic Heterogeneous Wireless Sensor Networks

MARCEL STEINE, Eindhoven University of Technology
MARC GEILEN, Eindhoven University of Technology
TWAN BASTEN, Eindhoven University of Technology and TNO-ESI

Wireless Sensor Networks (WSNs) are commonly deployed in dynamic environments where events, such as moving sensor nodes and changing external interference, impact the performance, or Quality-of-Service (QoS), of the network. QoS is expressed by the values of multiple, possibly conflicting, network quality metrics, such as network lifetime and maximum latency of communicating a packet to the sink. A sufficient QoS should be provided by the WSN to ensure that the end-user can successfully use the WSN to perform its application. Current run-time reconfiguration approaches optimize only a single network QoS metric, focus on node quality metrics and/or consider a homogenous configuration. They thereby ignore existing trade-offs between important network QoS metrics, the need for nodes to collaboratively achieve the network QoS or heterogeneity in the network. We propose a distributed reconfiguration approach that actively maintains a sufficient QoS at run-time for a heterogeneous WSN in a dynamic environment. To resolve differences between the current and required QoS of the network, nodes reconfigure themselves by adapting controllable parameters of the protocol stack, such as the transmission power or maximum number of packet retransmissions. They take the opportunities of other nodes affecting the same network QoS metric into account. The behaviour of the reconfiguration approach and the trade-offs involved are analyzed in detail. With the use of simulations and experiments with an actual deployment we show that our approach allows a better optimization of QoS objectives while constraints are met, e.g., it achieves the same packet-loss with a significant longer lifetime, compared to current (re-)configuration approaches.

1. INTRODUCTION

A WSN consists of a, often heterogeneous, set of small autonomous devices, called *nodes*, that are capable of sensing, processing and wireless communication and collaborate to perform a task. A typical task is health monitoring of elderly people [Stankovic et al. 2005]. The (projected) percentage of the population that will be aged 60 or over is steadily increasing [United Nations Report 2009]. Furthermore, the ratio between workers and retirees is projected to fall from 9 in 2009 to 4 persons of working age per person aged 65 or over in 2050. This will result in large costs to support the current health-care system. By extending the monitoring of health from the hospital to the elderly (nursing) home, expensive hospital care can be limited by early detection and prevention of health issues. A wireless sensor network is a suitable technology enabling this in-home monitoring. Nodes can be equipped with several types of sensors to (unobtrusively) monitor the health of a person by attaching it to the body, clothes or accessories such as a watch [Milenković et al. 2006]. These sensors continuously measure vital signs such, as heartbeat rate, blood pressure and stress level. Data from one or more sensors is propagated to a central location using a backbone of static nodes deployed throughout the entire area in which the monitored person(s) can be located. The static nodes could, besides forward sensor data, furthermore be used to collect information on environmental properties, such as temperature and humidity. At the central location, the sink, all data is collected and combined to derive a complete picture for diagnosis. Care takers can thereby adjust the treatment of the monitored persons if needed and are informed about (upcoming) health issues. In this paper, we focus on such monitoring systems consisting of both static and mobile nodes.

Meeting explicit performance targets by the WSN is important for the correct execution of a task, especially for health-monitoring. The power consumption of nodes is typically heavily constrained as nodes rely on batteries or energy harvesting and frequently changing batteries is inconvenient or simply not possible. It is furthermore

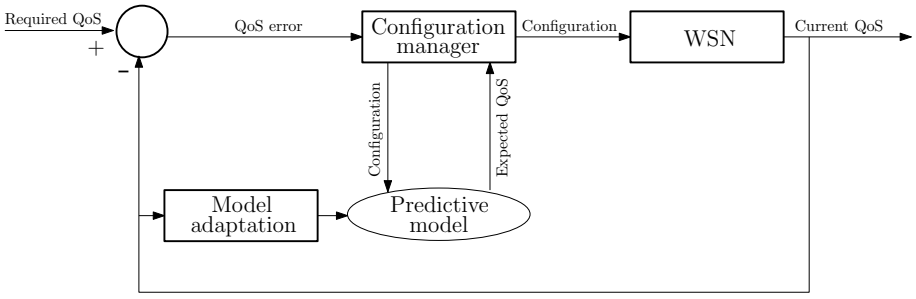


Fig. 1: Overview of feedback-control-based QoS provisioning approach

required that a sufficiently high number of sensor data packets are received at the sink, i.e., they exhibit a low packet-loss. The time between sensing data and receiving it at the sink allowing it to be analysed, i.e., the packet latency, is another aspect to be constrained. This ensures that vital signs can be determined and changes in health properties can quickly be observed by the monitoring system.

Configuring the WSN such that the values of network performance metrics are sufficient, often referred to as providing the required *Quality-of-Service* (QoS), is a non-trivial task. This configuration step involves setting all controllable protocol parameters of each individual sensor node, such as the radio transmission power, buffer sizes, maximum number of retransmissions and selected routing parent(s). These parameters often have a conflicting impact on the performance metrics, e.g., the transmission power of a node makes a trade-off between lifetime and packet-loss. Furthermore, due to the heterogeneity in the network, for example due to different hardware or different external interference, nodes can have a different influence on the metrics and trade-offs. Finally, optimizing the configuration of nodes should not be considered in isolation as multiple nodes influence the same network metric, such as the packet latency to the sink.

In practice, like in the health-monitoring scenario, we see that WSNs are operating in a dynamic context where events, such as external interference, mobility of nodes or fluctuating traffic load, continuously influence the network QoS. A straightforward solution would be to focus on the worst-case dynamics and determine the best configuration for that situation using existing analysis approaches [Ferentinos and Tsiligiridis 2007; Hoes et al. 2009]. But a single (worst-case) configuration is typically not able to make an efficient trade-off between the QoS metrics at all times, especially if the network is not experiencing its worst-case behaviour. Therefore, run-time reconfiguration is needed where parameters are adapted to continuously provide the required QoS.

This paper deals with run-time reconfiguration of a dynamic heterogeneous WSN to provide the required QoS, defined by multiple network performance metrics, given a deployed network with selected protocols and its controllable parameters. We introduce a distributed reconfiguration approach that lets every node control their parameters.

Fig. 1 shows a conceptual representation of our approach. Every node uses a feedback control strategy, where estimates of the current QoS of the network are used to adapt the node's controllable parameters. Each node decides to what extent it should contribute to reduce any observed deviation between the observed and required QoS or leave it to other nodes. Using an adaptive predictive model, of how different parameters influence QoS, the new configuration is selected. Based on observations of the current QoS and QoS before reconfiguration, the predictive model is updated to match

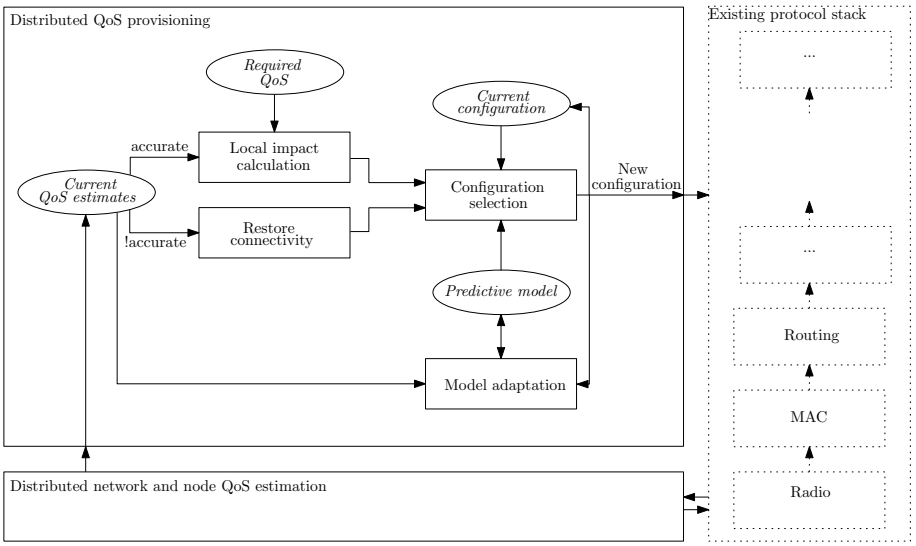


Fig. 2: Implementation of feedback control mechanism on a node

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. The reconfigurations are done in a decentralized fashion concurrently on every node in the WSN. Together, the adaptations ensure network QoS.

Fig. 2 shows how the conceptual representation is translated to a practical implementation running on all nodes. As a basis for the reconfiguration approach, an existing efficient distributed service, as described in [Steine et al. 2011a], is used to estimate the current network QoS. 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, the local change on the network QoS that should be established by the node is calculated using knowledge of the required QoS and effort of other nodes spent on providing this QoS. 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. A new configuration is selected using information about the current configuration and the expected impact of changing any of the controllable parameters. The latter is provided by predictive models, which are dynamically adapted at run-time to adjust to the dynamics in the network. The implementation of these individual procedures is explained in detail in this paper. Extensive simulations are used to evaluate the accuracy of our reconfiguration approach using the predictive models. They show that the adaptive predictive model is sufficiently accurate to support the reconfiguration approach, with limited overhead. Moreover, simulations of the response of the network to a single upset event, i.e., the step-response, give insight in convergence and stability for setting the parameters of the approach, such as the speed at which nodes reconfigure themselves. The applicability and performance of our approach in practice is analyzed with an actual monitoring deployment in an office building. 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 at a longer network lifetime.

Compared to current distributed run-time reconfiguration approaches for QoS provisioning in a WSN, our approach:
(1) considers multiple conflicting QoS metrics, while current approaches often consider only one, typically power, and thereby ignore important trade-offs.
(2) lets nodes reconfigure to control relevant network QoS metrics, e.g., latency to the sink, instead of local QoS metrics, such as latency to the parent(s). Focusing on local QoS ignores the fact that in a dynamic heterogeneous network, the local QoS required to achieve the network QoS, differs between nodes and over time. Our approach explicitly considers this heterogeneity and these dynamics in the network. The overhead of distributing the required network information to all nodes is limited due to the use of an existing efficient distributed service [Steine et al. 2011a] and compensated by the (potentially large) improved efficiency of QoS provisioning.

An earlier version of the reconfiguration approach as discussed in this paper was published in [Steine et al. 2012]. This paper incorporates and extends [Steine et al. 2012], with the following new contributions:
- A more elaborated discussion on how the distributed service as introduced in [Steine et al. 2011a] allows every node to estimate the network QoS information needed.
- Instead of optimizing node power consumption we optimize the network lifetime, since minimizing power consumption is not necessarily the same as maximizing the network lifetime. Optimizing lifetime is typically of more interest for battery powered devices which, for instance, have different battery capacities or are powered up at different moments in time.
- An improved derivation of local adaptation rate, which controls how much a node contributes to reducing any observed network QoS error. It results in a better balancing, between nodes, of the effort, and allows a more intuitive interpretation.
- Improved robustness by explicitly covering the case where nodes get disconnected from the main network so that they are no longer able to derive accurate network QoS estimates.
- Instead of the simplistic static linear models used in [Steine et al. 2012], new adaptive predictive models are used to determine how parameters should be adapted to achieve a desired change in QoS. A feedback control approach is integrated, within the already used feedback control strategy to control the configuration, to maintain accurate model parameters based on the current QoS and QoS before reconfiguration. These models more accurately reflect the changing impact of a reconfiguration in dynamic WSNs compared to the previously used static impact models, leading to better performance.
- Extensive simulations of the response of the reconfiguration approach to a single upset event, i.e., the step-response. This analysis provides insight in the behaviour of the distributed approach.
- New experiments with an actual deployment for monitoring persons in a building showing the feasibility of our approach and improved QoS provisioning over current (re-)configuration approaches.

The remainder of this paper is structured as follows. In Section 2 we discuss related work. Section 3 presents an example to show the need for a distributed reconfiguration approach and it examines the required functionality of such a mechanism. It furthermore introduces terminology used throughout this paper. In Section 4 we introduce our distributed reconfiguration approach. Section 5 states how nodes estimate the node and network QoS information that is locally needed. In Section 6 we explore the behaviour and performance of the approach with the use of extensive simulations. Section 7 discusses the results of the integration of the feedback controller for an actual monitoring scenario and shows results. Section 8 concludes.

2. 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 for the design-time derivation of a single (worst-case) configuration [Ferentinos and Tsiligiridis 2007; Hoes et al. 2009], but the need for (additional) run-time reconfiguration, or adaptation, to respond to dynamism in the network is recognized by many researchers. Our run-time reconfiguration approach distinguishes itself from related 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 [Kogekar et al. 2004], 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 [He et al. 2003] a feedback control approach is used to guarantee that each node maintains an average delay for packets transiting a node. In [Chipara et al. 2006] 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 [Ye et al. 2004; Sun et al. 2008]. 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 [Zimmerling et al. 2012] 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 [Park et al. 2013] 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 resutls 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 the distributed service of [Steine et al. 2011a], 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 [Melodia et al. 2006].

We adopt ideas from the area of control theory and (decentralized or distributed) model predictive control [Richards and How 2004]. 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. THE GOAL OF QOS PROVISIONING

In this section we formalize the goal of distributed run-time reconfiguration for QoS provisioning in a dynamic heterogeneous WSN. In Section 3.1 we go into details of QoS provisioning. We introduce the terminology used throughout this paper in Section 3.2. Section 3.3 illustrates the desired functionality of the reconfiguration approach to achieve the goal of QoS provisioning using an example.

3.1. QoS Provisioning

Looking from the WSN end-user perspective, the WSN should provide sufficient QoS to successfully perform the task specified by the end-user. To achieve this, we formulate requirements related to the value of one or more network QoS metrics, such as packet latency and packet-loss to a particular node or network lifetime. We want the values to be either satisfying a constraint, for example a packet latency to all sinks less than 1 second, or be optimized for a given metric aspect, for example a maximum network lifetime. We combine the metrics to optimize into a single cost-function, referred to as the QoS optimization objective.

The configuration of the network, i.e., parameter values of the controllable parameters for every node, should establish that constraints are met, while the QoS objective is optimized. We assume that adapting controllable parameters results in trade-offs between the constrained metrics and the optimization objective. This implies that a constrained metric, say latency to any of the sinks, should not just meet the constraint, but also be close to the constraint to allow freedom to optimize the objective, say maximize the lifetime, thereby avoiding QoS over-provisioning. In practice we will typically see that the QoS metric value of a path, such as the latency to the sink, does not exactly sum up to the constraint, due to the discrete nature of the parameters and resulting impact on the QoS metric. With our QoS provisioning strategy we therefore focus on providing a value for all constrained QoS metrics between a given $lowerbound$ and $upperbound$, where $lowerbound \leq upperbound \leq constraint$ assuming a lower metric value is better. For QoS metrics where a higher value is considered better, such as delivery ratio, a similar discussion holds.

With a reconfiguration approach steering the QoS metric close to the constraint, small deviations may directly violate the constraint. Infrequent and short violations of the constraint are often tolerated in practice, because in any practical environment we have to deal with uncontrollable factors, such as the behaviour of unreliable wireless communication and dynamism in the network. The $upperbound$ can be selected to be lower than the $constraint$ to provide a safety margin against constraint violations. Aiming at, for example, a lower latency reduces freedom to trade-off and optimize the lifetime. With a larger distance between the $lowerbound$ and $upperbound$, it is potentially easier for a reconfiguration approach to keep the latency within the bounds, but also results in less freedom to trade off and optimize the lifetime. We assume the re-

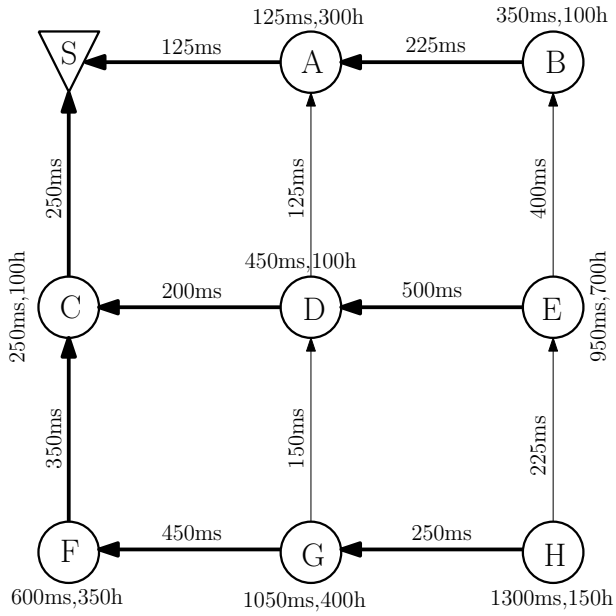


Fig. 3: Link and path latency for 9-node sensor network

quired *lowerbound* and *upperbound* to be given. Exploration of the trade-offs involved with selecting the *lowerbound* and *upperbound* for a particular deployment given the *constraint* are outside the scope of this paper.

In summary, the goal of QoS provisioning we want our reconfiguration approach to achieve is defined as follows:

DEFINITION 3.1. (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 objective is optimized.*

3.2. Terminology

Fig. 3 shows a simplified example of a WSN with 9 nodes. The directed edges between two nodes represent the links used to communicate packets towards sink S. Some nodes send packets to multiple neighbouring nodes, resulting in multiple *paths* to the sink.

DEFINITION 3.2. (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. 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 3.3. (CRITICAL PATH OF NODE n_i FOR A GIVEN QOS CONSTRAINT) *The critical path of node n_i for a given QoS constraint is the path with the critical value with respect to that constraint, e.g., highest latency to a sink.*

Every node has a critical path with respect to each of the constrained metrics. With a single constrained QoS metric, as in the example of Fig. 3, every node has a single critical path. The thicker solid lines in Fig. 3 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 and its latency is 450 milliseconds. 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 3.4. (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 node n_i to sink s for that constraint, where n_i is the end node of the critical path.*

There are three end-to-end critical paths (for the latency constraint) in Fig. 3 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 considered to be the *end-to-end critical path of node C*.

DEFINITION 3.5. (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.*

The end-to-end critical path of 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. When considering multiple constrained metrics, nodes have multiple end-to-end critical paths to all sinks. Of these paths more than one may require the reconfiguration approach to respond to get the value in between the specified lower and upper bound. Often, reconfiguration can only focus on one constraint at a time. The end-to-end critical path of node n_i defines the most important end-to-end critical path of node n_i to sink s considering all constrained QoS metrics. It is defined as follows.

DEFINITION 3.6. (END-TO-END CRITICAL PATH OF NODE n_i) *Given the end-to-end critical paths of node n_i for all constrained QoS metrics, the end-to-end critical path of node n_i is the most important end-to-end critical path according to a given selection mechanism. Example mechanisms may, for example, use priorities between QoS constraints, the relative error with the constraint or a combination of these two.*

With multiple QoS constraints, the end-to-end critical path of a node can change over time due to changes on the path itself and QoS constraints becoming critical. Iteratively reconfiguring to get the current end-to-end critical path within bounds will typically eventually result in achieving the goal of QoS provisioning, if such a configuration exists that meets all QoS constraints and optimizes the objective. Multiple nodes

can share the same end-to-end critical path. 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 3.7. (COLLABORATING NODES) *A set of collaborating nodes consists of all nodes which share the same end-to-end critical path.*

For every end-to-end critical path there is a unique set of collaborating nodes. 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 3.8. (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 3.9. (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. By definition, 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.

3.3. Motivating Example

Fig. 3 shows the QoS of the network after one or more dynamic events. For this example we aim at keeping 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. We consider the latency metric to be the most important (or only) QoS metric to focus reconfiguration on at this moment in time. The end-to-end critical path of every node is equal to the end-to-end latency-critical path.

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, or (3) a latency within range, but the lifetime of the nodes on the path is not optimal. In the example of Fig. 3 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.

(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) $e2elat_i < lowerbound$. 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, in contrast to most current run-time adaptation approaches that optimize only node QoS metrics. 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. 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.

4. DISTRIBUTED RECONFIGURATION FOR QOS PROVISIONING

We introduce a distributed reconfiguration approach where every node locally adapts its own parameters to ensure the provisioning of the required network QoS. Fig. 2 shows how the approach is translated into algorithms running locally on every node. 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. In this section we discuss the individual parts of our approach in detail. For this we assume that cer-

tain network QoS and node QoS estimates are available to the algorithm. The actual derivation of this information is discussed in Section 5.

In Section 4.1 we discuss the calculation of the local changes a node should make for reducing an observed network QoS error. This is done if accurate network QoS estimates are available. In the case that a node is not connected to the main network our approach initiates a reconfiguration to restore connectivity according to the approach discussed in Section 4.2. For the selection of a new configuration, adaptive predictive models are used 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. Section 4.3 discusses these models. Section 4.4 describes how to determine a new configuration using these models. The impact of a given parameter on one or more metrics may change over time due to changes in the network. Therefore, we adapt the model parameters at run-time. This is discussed in Section 4.5. We assume that at the current moment in time the end-to-end latency of all packets sent to the sink is the most important constrained metric to focus on for all nodes (see Definition 3.6). The QoS objective is based on a single metric, maximizing the network lifetime. This allows us to reuse the example of Fig. 3 during the explanation of our approach. For the ease of explanation, we furthermore assume that a lower QoS metric value is better, as is the case for the latency metric. For some metrics higher values are better, such as the packet delivery ratio. Our approach can be adapted to also deal with higher is better metrics, by inverting the comparisons on the appropriate places in the pseudo-code provided in the section and explicitly checking the kind of trend the model has.

4.1. Local Impact Calculation

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 the lifetimes. In Section 3.3, 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.

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

This function states that node i should impact the latency of the end-to-end critical path with a percentage, per_i, of the total amount, $amount_i$, we want the collaborating nodes to impact the latency of the end-to-end critical path. 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 6 and 7.

The values of $amount_i$ and per_i differ between the nodes in any of the three possible cases, as explained in Section 3.1, and are discussed in the remainder of this subsection.

With the latency of the end-to-end critical path outside the required latency range (cases (1) and (2) of Section 3.3), the task of the collaborating nodes is to resolve the latency error. The error observed by node i, $error_i$, is the difference between and the node's own local estimate of the end-to-end critical latency, $e2elat_i$, and the latency range as defined by the $lowerbound$ and $upperbound$.

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

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.

$$amount_i = -error_i$$

The percentage of the total amount of latency, $per_i, 0 \leq per_i \leq 1$, that should be taken care of by node i (and thereby also the percentage $(1-per_i)$ 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 percentage is based on several considerations. First of all, we want $\Sigma_{j \in N} per_j = 1$, 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 $e2elat_i > upperbound$ we furthermore want a higher percentage 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, $sumLife_i$, we can determine the percentage per_i by comparing a node's own estimated remaining lifetime, $lifetime_i$ with the sum. For the opposite case that $e2elat_i < lowerbound$ we want a higher percentage 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, $invSumLife_i$, and the inverse of the lifetime, $lifetime^{-1}$. For the two cases, per_i is defined by the following equations.

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

If we look at the path from node H to sink S in the example of Fig. 3, 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 percentages 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.

As soon as the end-to-end latency is within the target range, i.e., $lowerbound \leq targetLatency \leq upperbound$, situation (3) of the example of Section 3.3, 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 percentage, $per_i, -1 \leq per_i \leq 1$, as described below, by subtracting the percentage a node would take in reducing the latency from the percentage it would take in increasing the latency. Depending on the predominant factor, the percentage 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, the percentage is 0 for every node, as expected.

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

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.

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

Collaborating nodes D and E in the example of Fig. 3 are on an end-to-end critical path with a latency in the required range, but their lifetimes are unbalanced. The positive percentage 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 percentage $per_E = {}^{-6}/_{8}$, 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., $localrate_i = remainderPathLoss * rate_i$.

4.2. 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 5, 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.

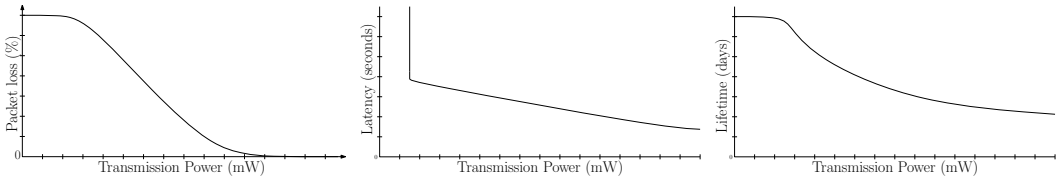
4.3. Predictive Model

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. Fig. 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*.

DEFINITION 4.1. (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 [Steine et al. 2012]. 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 of accuracy in dynamic networks. As impact of changing parameters strongly depends on changing



(a) (b) (c)

Fig. 4: Impact of transmission power on (a) packet-loss, (b) latency and (c) lifetime

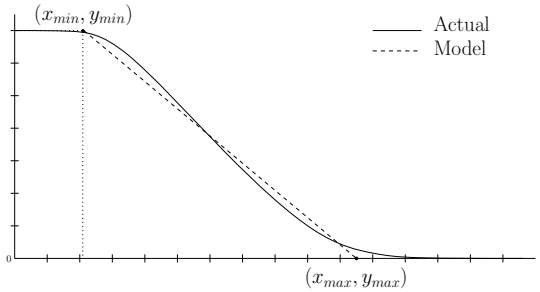


Fig. 5: Describing adaptation impact on QoS with four parameters

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 Fig. 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 assume a linear 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, $parval$.

$$qos(parval) = \begin{cases} y_{min} + ((parval - x_{min})/(x_{max} - x_{min})) * (y_{max} - y_{min}) & \text{if} \quad y_{min} \leq y_{max} \\ y_{min} - ((parval - x_{min})/(x_{max} - x_{min})) * (y_{min} - y_{max}) & \text{if} \quad y_{min} > y_{max} \end{cases} \tag{1}$$

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 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

ALGORITHM 1: Configuration selection of node n

```
1  ΔLat = |localrate_i|;
2  while ΔLat > 0 do
3      for all controllable parameters i do
4          (value_i, outside_i, Δlat_i, Δlife_i) := getBestValueForParameter(i, ΔLat);
5          (best, value_best, Δlat_best) := checkWhetherBestParameter(i, value_i, outside_i, Δlat_i, Δlife_i);
6      end
7      adapted := adaptParameter(best, value_best, ΔLat, Δlat_best);
8      if (adapted) then ΔLat := ΔLat − |Δlat_best|;
9      else break;
10 end
```

predicted not to be possible with (only) adapting the current parameter.

$$par(qosval) = \begin{cases} x_{min} + ((qosval - y_{min})/(y_{max} - y_{min})) * (x_{max} - x_{min}) & \text{if} \quad y_{min} \leq y_{max} \\ x_{min} + ((y_{min} - qosval)/(y_{min} - y_{max})) * (x_{max} - x_{min}) & \text{if} \quad y_{min} > y_{max} \end{cases} \tag{2}$$

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 4.5.

4.4. Configuration Selection

The global overview of the procedure to select the configuration is shown in Algorithm 1. The change in the latency metric to achieve is the (absolute) calculated 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 (line 2), the best possible parameter is selected from all controllable parameters (line 3) to achieve (part of) this change. The function getBestValueForParameter (line 4, later explained using the pseudo-code of Algorithm 2) determines to which value ($value_i$) to adapt parameter i given the required change in latency. It furthermore determines the characteristics of this adaptation, i.e., whether the new value is within the suitable range (given by Boolean value $outside_i$) and the predicted impact on latency, Δlat_i, and lifetime, $\Delta life_i$. This is needed by checkWhetherBestParameter (line 5, Algorithm 3) which iteratively checks whether the current 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 7, Algorithm 4) is used to determine if the best parameter should be adapted in this round. If so, the remaining change is updated (line 8). Else, the reconfiguration of the node is stopped (line 9).

Selecting the best parameter value. The pseudo-code for the selection of the best value of parameter i given a required change in latency, ΔLat, is shown in Algorithm 2. We discuss the code using the example of Fig. 6, having a decreasing impact trend, $y_{min} > y_{max}$ and $\Delta Lat < 0$. Given the current parameter value, the predicted latency value is provided by getCurrentLatency (using Equation 1) (line 1). Similarly, the exact desired parameter value to achieve ΔLat is provided by getExactParValue using Equation 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 Fig. 6). Note that the selected value for adaptation can potentially be outside the suit-

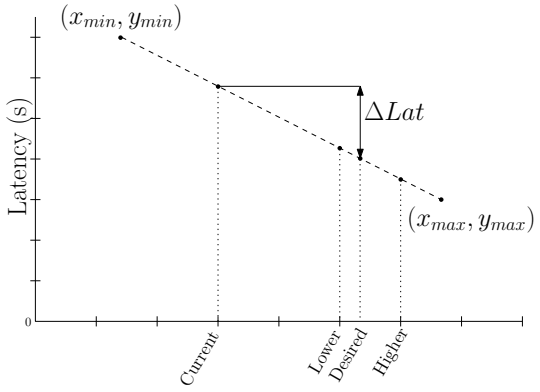


Fig. 6: Selecting a new parameter value from predictive model

ALGORITHM 2: getBestValueForParameter(i, ΔLat)

```
1  latency := getCurrentLatency(curValue_i);
2  desired := getExactParValue(latency + ΔLat);
3  lower, higher := getDiscreteParValues(desired);
4
5  value_i := −1, lowerPossible := false, higherPossible := false;
6  if ((y_min < y_max ∧ ΔLat > 0) ∨ (y_min > y_max ∧ ΔLat < 0)) then
7      if (lower ≠ −1 ∧ lower ≠ curValue_i) then lowerPossible := true;
8      if (higher ≠ −1) then higherPossible := true;
9      if (lowerPossible) then value_i := lower;
10     else if (higherPossible = true) then value_i := higher;
11 else
12     if (higher ≠ −1 ∧ higher ≠ curValue_i) then higherPossible := true;
13     if (lower ≠ −1) then lowerPossible := true;
14     if (higherPossible) then value_i := higher;
15     else if (lowerPossible = true) then value_i := lower;
16
17 if value_i ≠ −1 then
18     outside_i := x_min ≤ value_i ≤ x_max;
19     Δlat_i := getLatencyImpactFromModel(value_i, curValue_i);
20     Δlife_i := getLifetimeImpactFromModel(value_i, curValue_i);
21 end
22 return (value_i, outside_i, Δlat_i, Δlife_i)
```

able 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 4.5. 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, -1, and the best adaption option to

ALGORITHM 3: checkWhetherBestParameter($i, value_i, outside_i, \Delta lat_i, \Delta life_i$)

```
1  betterOption := false;
2  if (best = −1) then betterOption := true;
3  else if (bestOutside ∧ ¬outside_i) then betterOption := true;
4  else if (¬(¬bestOutside ∧ outside_i)) then
5      adaptRatio := Δlat_i/Δlife_i;
6      if (Δlat_i < 0 ∧ adaptRatio > bestRatio) then betterOption := true;
7      if (Δlat_i > 0 ∧ adaptRatio < bestRatio) then betterOption := true;
8
9  if (betterOption) then
10     best = i, value_best := value_i, Δlat_best = Δlat_i, bestOutside := outside_i, bestRatio := adaptRatio;
11 return (best, value_best, Δlat_best)
```

be unknown (equal to -1). 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 5). For our example, the desired value is higher than the current parameter value. The lower value is a possible adaptation option if it exists ($\neq -1$) and is not equal to the current parameter value (line 7). The higher value is possible if it exists (line 8). 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 9). 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 10). The above discussion also holds for the symmetric case that $y_{min} < y_{max}$ and $\Delta Lat > 0$ (added to line 6). 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 11-15).

If an adaptation option is available (line 17), its characteristics are determined. It is checked if the value is within the suitable range (line 18). Furthermore, the predicted change in latency (line 19) and lifetime (line 20) is determined using the predictive models. 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 -1 (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 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 4.5. 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 adaptation option. The pseudo-code to iteratively check whether the current option is the best so far is shown in Algorithm 3. The current

ALGORITHM 4: adaptParameter($best, value_{best}, \Delta Lat, \Delta lat_{best}$)

```
1  adapted := false;
2  if (0 ≤ |Δlat_best| ≤ |ΔLat|) then
3      Adapt(best, value_best);
4      adapted := true;
5  else
6      prob := ΔLat/Δlat_best;
7      if (randomFloat(0, 1) < prob) then
8          Adapt(best, value_best);
9          adapted := true;
10 return adapted
```

best parameter value is assumed to be stored in $best$ and is -1 if non existing. Initially there 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, 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 (line 5). 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 6). For a positive rate, we want the highest possible increase in lifetime, i.e., the lowest impact ratio (line 7). If the newly suggested adaptation option is found to be better, this is stored (lines 9 and 10) for future comparison.

(Probabilistic) Adaptation of best parameter. Algorithm 4 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 2,3 and 4). In the case that predicted impact is more than required, a probabilistic approach is used. The probability of adaptation is proportional to the overshooting of the required change (line 6). 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 7, 8 and 9). In practice, we see that the required change in latency in a single round is typically low and and a probabilistic adaptation is directly 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 4.2, 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.

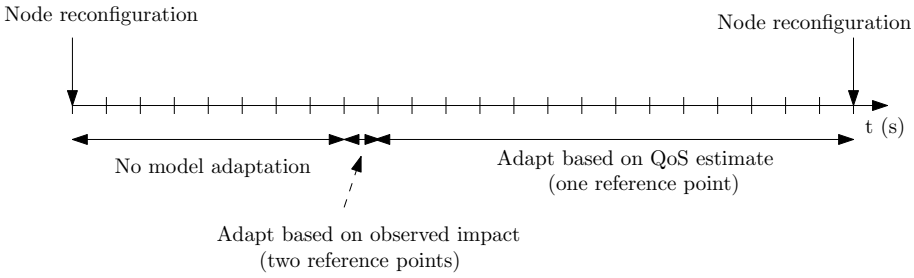


Fig. 7: Model maintenance strategy over time for a given node

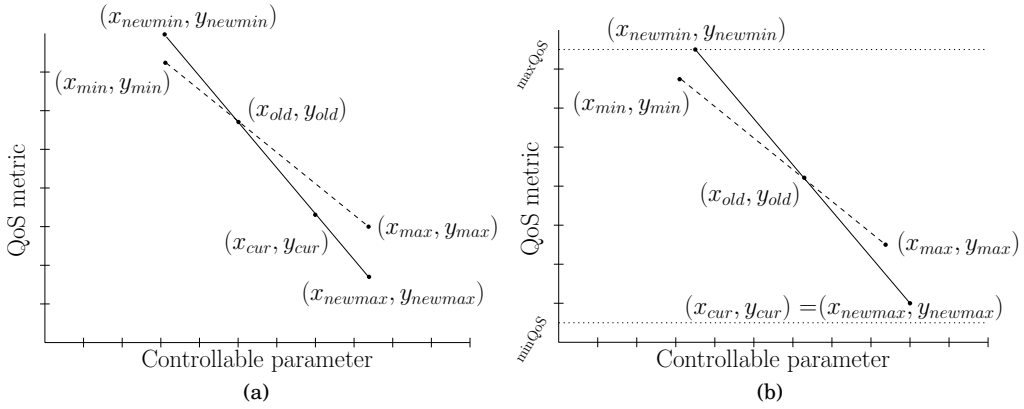


Fig. 8: Model before (dashed line) and after (solid line) adaptation. (a) Base case. (b) Current value outside suitable range and considering the minimum and maximum achievable QoS

4.5. Model Maintenance

We assume we have an initial predictive model for every parameter-metric pair at the start of WSN operation. Simulations, similar to those done to get the impact curves of Fig. 4, can be used to identify an initial model. The model is periodically adapted at run-time. As the model is impacted by reconfiguration, we update the model every round. Our strategy for the run-time model maintenance is visualized in Fig. 7. After a reconfiguration of the node, the predictive model is not adapted until the impact of reconfiguration is reflected in the local estimated network QoS. This takes time, due to the propagation of network QoS estimation and potential averaging of QoS values over time. 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 value of the parameter 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.

ALGORITHM 5: 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 ∨ x_old > x_max then
3      adaptForCurrentQoS();
4  else if x_old < x_cur ∧ y_old − y_cur < minDif ∧ x_cur < x_max then
5      x_newmax := x_cur;
6  else if x_old > x_cur ∧ y_cur − y_old < minDif ∧ 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);
```

Adapting the model based on observed impact of reconfiguration. Fig. 8a 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 5 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 6). 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 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 Fig. 8b, 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 Fig. 8b, for x_{newmax} and y_{newmax}. Finally, the

ALGORITHM 6: 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 ≤ x_cur ≤ 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) ∗ ((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) ∗ ((x_max − x_min)/(y_max − y_min));
12         y_newmax := minQoS;
13 AdaptModel(x_newmin, y_newmin, x_newmax, y_newmax);
```

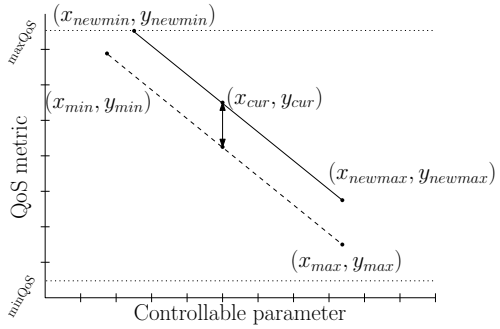


Fig. 9: Adapting model parameters with current parameter value

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 percentage 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.

Adapting the model based on current QoS. Fig. 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 6 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 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 Fig. 9. Finally, the model is adapted to the newly calculated parameter values (line 13).

5. QUALITY-OF-SERVICE ESTIMATION

Both network and node QoS metric information are needed by the nodes to support the distributed reconfiguration approach. A node needs 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. This network QoS information is determined and disseminated with an existing generic distributed service [Steine et al. 2011a]. This service is based on a distributed, iterative computation of a fixed point of a converging recursive equation which characterizes the desired network information. Section 5.1 summarizes the relevant aspects of the service and shows the equations characterizing our required network metrics. Specific details about, for example, the exact format of the packets to send and algorithms to locally run on the nodes, can be found in [Steine et al. 2011a]. The local estimation of node QoS required to compute network QoS is discussed in Section 5.2. For the ease of explanation, we focus on the single trade-off as used in the previous section, i.e., between the maximum latency to the sink and network lifetime. Our service is generic in the sense that it allows every network QoS to be estimated that can be described in the form of a converging recursive function. The estimation of latency to the sink can for example easily be replaced or extended with the estimation of packet-loss to get a reconfiguration approach (also) focusing on the trade-off between end-to-end packet-loss and network lifetime.

5.1. Network QoS Estimation

The estimation of the required network QoS information involves several steps. 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. Fig. 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 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

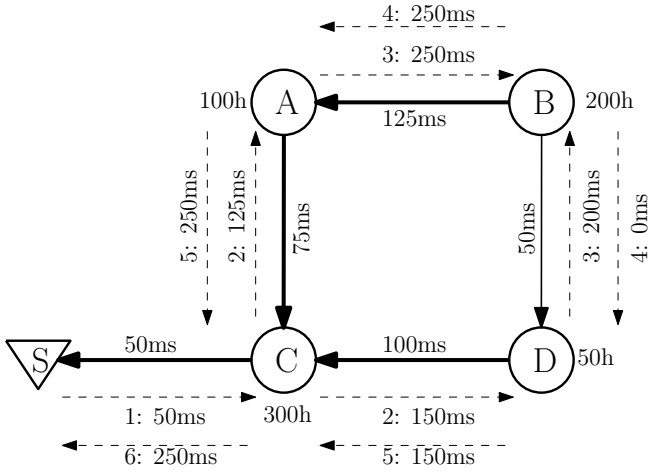


Fig. 10: Distribution of the latency to sink S and latency of the end-to-end critical path

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 Fig. 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

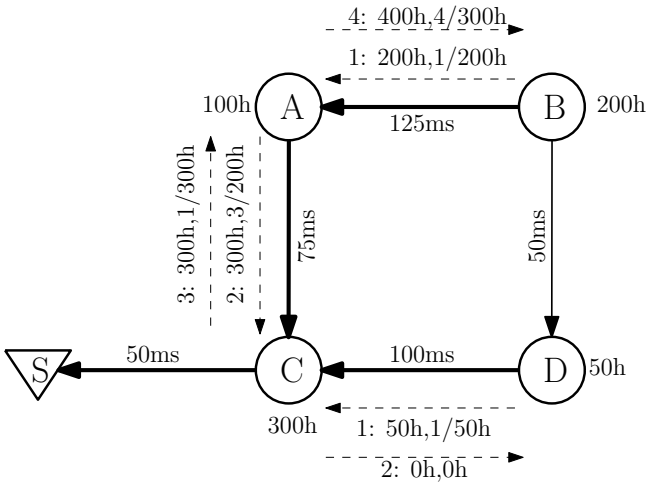


Fig. 11: Distribution of the sum and inverse sum of lifetimes of collaborating nodes

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 determines their value based on the end-to-end latency of their critical child, c_i.

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

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

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

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. Fig. 11 shows the flow of information needed to locally estimate the sum and inverse sum. 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 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).

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

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

The example suggests the network is in a static situation, with bidirectional links. Presence of any asymmetric, and potentially unidirectional, communication links [Kotz et al. 2003; Zhao and Govindan 2003], 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.

5.2. Node QoS Estimation

Node i should know its own link latency to all parents x, $ll_{i,x}$ to derive maximum latency information. 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 is non-trivial, because of the unpredictable changes in power consumption due to 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 gives 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 at 1500mAh

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 [Prayati et al. 2010] and the most important part of the power controlled by reconfiguration. In TinyOS [TinyOS website 2013] 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. [TelosB Datasheet, Crossbow Inc.]) an estimate of the power consumption can be made. This approach is used for all experiments in this paper. 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.

5.3. Generalizations

The collection of network QoS information is based on a generic service. This service is generic in the sense that it allows every network QoS to be estimated that can be described in the form of a converging recursive function. In the discussion above, the estimation of latency to the sink can for example easily be replaced or extended with the estimation of packet-loss to get a reconfiguration approach (also) focusing on the trade-off between end-to-end packet-loss and network lifetime.

6. CONTROLLER PERFORMANCE ANALYSIS

We start this section by discussing the approach used for performance analysis in more detail in Section 6.1. In Section 6.2 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 6.3. 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 6.4 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 6.5 we summarize the main conclusions. In this section, we use extensive simulations to evaluate the performance. A real deployment is used in the next section.

6.1. Introduction

During the distributed 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++ [OMNeT++ website 2013], a discrete-event simulation environment, with the use of MiXiM [MiXiM website 2013], a modeling framework for wireless networks.

6.2. Parameters

An important parameter of our reconfiguration approach is the speed of every node i, k_i, of the feedback control strategy (see Section 4.1). 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 enough 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 trade-offs involved with the parameters of the service, such as the update interval and the number of hops to forward information, are discussed in [Steine et al. 2011a]. The parameters of the service determine the speed at which information for local estimates becomes available to the nodes [Steine et al. 2011a]. As the availability of network QoS information forms the basis of our reconfiguration approach, the speed of propagation is important as we show in this section.

6.3. 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 [Tay et al. 1998]. 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 to 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 [TelosB Datasheet, Crossbow Inc.]. 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 [Polastre et al. 2004] 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 applica-

tion packet to the sink every 5 seconds. We assume the nodes to have the same initial battery-capacity of 2x1500mAh 1.5V. 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. The parameters that we allow to be controlled are the radio transmission power (any of the values specified in [TelosB Datasheet, Crossbow Inc.]), 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.

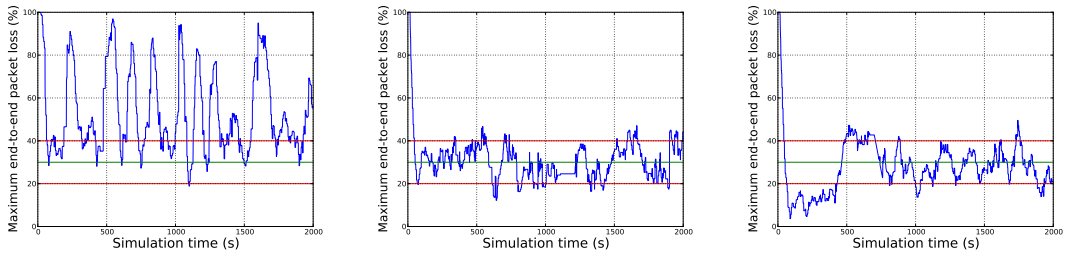
Impact of controller speed. Fig. 12 shows the step responses for the first 2000 seconds for three experiments with different speeds, i.e., values of k_i, for all nodes in the network.

Fig. 12a 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-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 [Steine et al. 2012]. 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.

Fig. 12b 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.

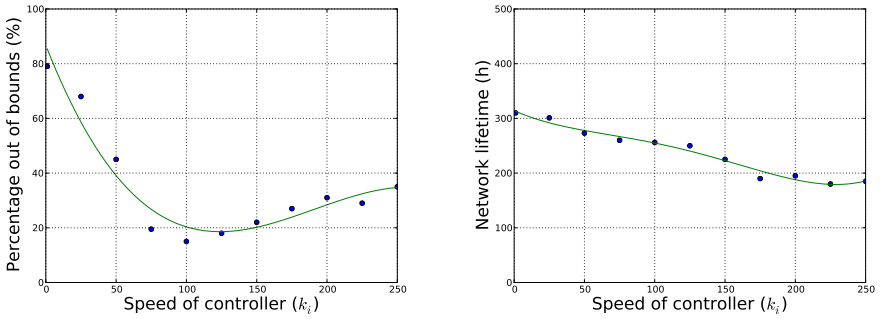
Fig. 12c shows the step-response for $k_i = 250$. The controller converges to an accurate stable situation, but requires a long time to converge. Every time dynamics occur that cause the packet-loss to be outside the required range, a long time is needed to reduce the resulting packet-loss error. Therefore, like a controller which is too fast, a controller which is too slow is not preferred.

Fig. 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.



(a) (b) (c)

Fig. 12: Packet-loss with controller speed that is (a) fast, (b) sufficient and (c) slow



(a) (b)

Fig. 13: Impact of controller speed on packet-loss behaviour and expected lifetime

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.

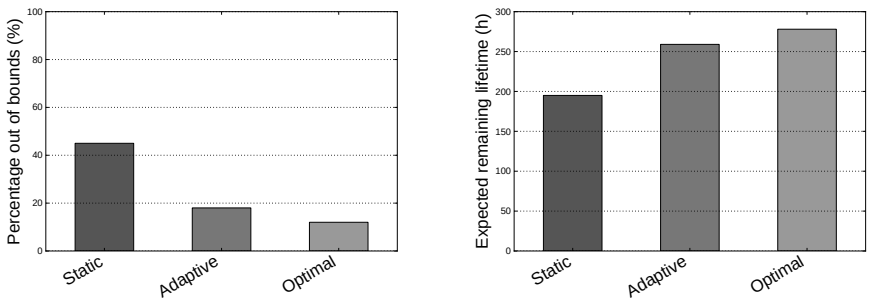
There is a range of speeds with approximately the same low percentage and high lifetime, in this case between around 75 and 125. For a deployment, we need to select the speed such that we get the desired behaviour as in Fig. 12b, i.e., to select k_i in the range providing the best possible provisioning 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 Fig. 12a, 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 that 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 sub-optimal 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.

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 the remaining inaccuracy of the dynamic model.



(a) (b)

Fig. 14: Impact of predictive models on packet-loss behaviour and expected lifetime

Fig. 14 shows the average percentage of time the end-to-end latency is outside the required range and the resulting lifetime using the three predictive models, for the same set of simulations.

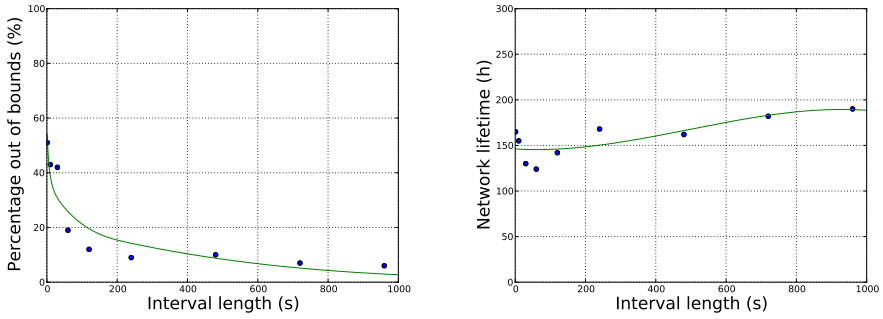
The static model results in a fairly large percentage of time in which the packet-loss is out of bounds, i.e., either higher or lower than the predefined packet loss range. With the dynamic approach, the percentage reduces and lifetime increases compared to the static approach as bounds on both the parameter 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.

6.4. 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 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. Fig. 15 shows both the average time the observed maximum packet-loss to the sink is outside the target range and the expected net-



(a) (b)

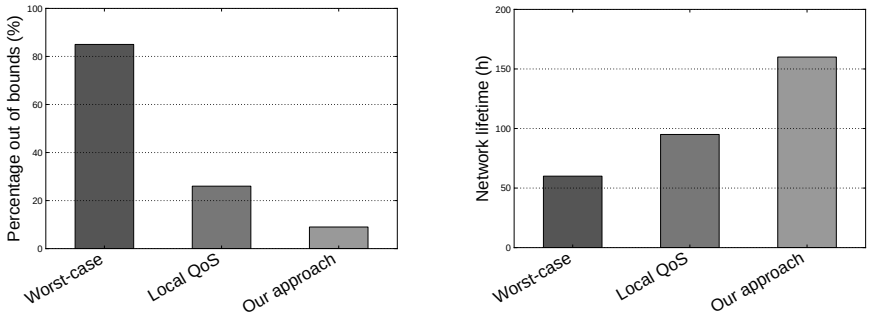
Fig. 15: Impact of amount of dynamics on packet-loss behaviour and expected lifetime

work 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 differences between the actual and estimated QoS. 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 of 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



(a) (b)

Fig. 16: Reconfiguration approach comparison

of every link to be as low as 5%. The experiments confirm that this level is necessary and sufficient.

Fig. 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; typically 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.

6.5. Conclusions from Simulations

From the simulations in this section we 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.

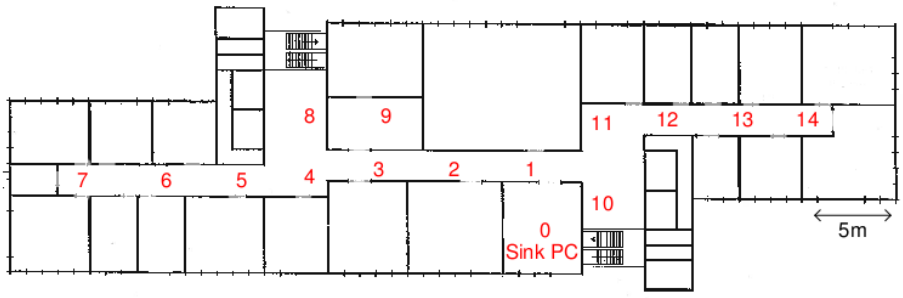


Fig. 17: Deployment set-up static nodes

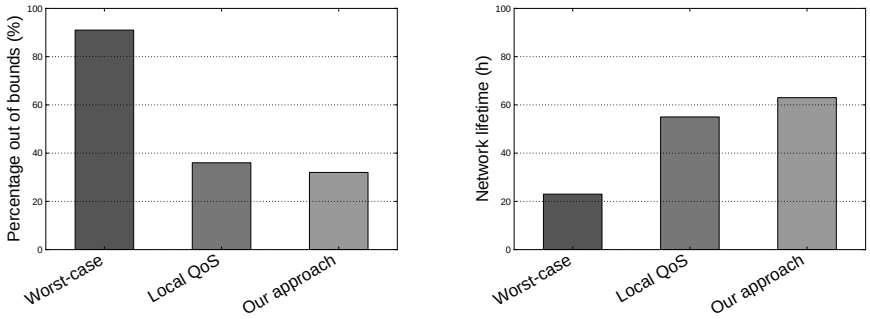
The introduced adaptive predictive model is more accurate than the previously introduced static approach allowing a better QoS provisioning at the expense of maintaining four model parameters for every parameter-metric pair.

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.

7. EXPERIMENTAL ANALYSIS

We implemented our reconfiguration approach, together with the distributed network QoS estimation service, in TinyOS [TinyOS website 2013] for a WSN deployed in our office building. It consists of 15 static TelosB [TelosB Datasheet, Crossbow Inc.] nodes deployed as shown in Fig. 17. Furthermore, 5 persons, which occasionally move between different locations in the office building, have a BSN [BSN website, ICL London] 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 40KB RAM and 7KB 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 [Moss and Levis 2008] 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 [Blagojevic et al. 2011] 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



(a) (b)

Fig. 18: Comparison of configuration approaches for (a) percentage out of bounds and (b) network lifetime

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, 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 cope 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.

Fig. 18a 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.

Fig. 18b 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 1.5V and a smaller full 750mAh 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 links, 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. Allowing a more fine-grained set of values would allow better adaptation. Furthermore, 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 of mobility 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 allow overhead. If it is important to quickly respond to infrequent relatively fast dynamics, with a large impact on network QoS, a reactive approach such as ours is inherently less efficient. For that case, one might resort to a proactive reconfiguration strategy, such as discussed in [Steine et al. 2011b]. With such an approach, observable network dynamics trigger reconfiguration before and independent of 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.

8. CONCLUSIONS

Wireless Sensor Networks (WSNs) have to deal with dynamics, such as node mobility and changing interference, which constantly affect the Quality-of-Service (QoS) they provide. To ensure that the QoS, expressed by multiple metrics, is maintained at runtime, adaptation of controllable protocol parameters is needed. In this paper, we introduced a 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, an existing distributed service is used which efficiently estimates network QoS information in dynamic heterogeneous WSN. Nodes reconfigure by adapting controllable parameters. 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 explore the parameters and characteristics of the approach and show with analysis of the step-response that our distributed control approach is stable, if the speed of the controller is set in accordance with the deployment characteristics. Experiments with an actual deployment show 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.

REFERENCES

BLAGOJEVIC, M., NABI, M., GEILEN, M., BASTEN, T., HENDRIKS, T., AND STEINE, M. 2011. A Probabilistic Acknowledgment Mechanism for Wireless Sensor Networks. In *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*. 63–72.

BSN WEBSITE, ICL LONDON. http://vip.doc.ic.ac.uk/bsn/m621.html. Last visited: April 2013.

CHIPARA, O., HE, Z., XING, G., CHEN, Q., WANG, X., LU, C., STANKOVIC, J., AND ABDELZAHER, T. 2006. Real-time power-aware routing in sensor networks. In *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on*. 83–92.

FERENTINOS, K. P. AND TSILIGIRIDIS, T. A. 2007. Adaptive design optimization of wireless sensor networks using genetic algorithms. *Computer Networks 51,* 4, 1031 – 1051.

HE, T., STANKOVIC, J., LU, C., AND ABDELZAHER, T. 2003. SPEED: a stateless protocol for real-time communication in sensor networks. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*. 46 – 55.

HOES, R., BASTEN, T., THAM, C.-K., GEILEN, M., AND CORPORAAL, H. 2009. Quality-of-Service Trade-off Analysis for Wireless Sensor Networks. *Performance Evaluation 66,* 3-5, 191–208.

KOGEKAR, S., NEEMA, S., EAMES, B., KOUTSOUKOS, X., LEDECZI, A., AND MAROTI, M. 2004. Constraint-guided dynamic reconfiguration in sensor networks. 379–387.

KOTZ, D., NEWPORT, C., AND ELLIOTT, C. 2003. The mistaken axioms of wireless network research. Tech. Rep. TR2003-467, Dartmouth College. July.

MELODIA, T., VURAN, M. C., AND POMPILI, D. 2006. 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. 78–92.

MILENKOVIĆ, A., OTTO, C., AND JOVANOV, E. 2006. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Computer Communications 29,* 13-14, 2521–2533.

MIXIM WEBSITE 2013. mixim.sourceforge.net. Last visited: April 2013.

MOSS, D. AND LEVIS, P. 2008. BoX-MACs: Exploiting Physical and Link Layer Boundaries in Low-Power Networking. Tech. rep., Stanford Information Networks Group Technical Report.

OMNET++ WEBSITE 2013. www.omnetpp.org. Last visited: April 2013.

PARK, P., DI MARCO, P., FISCHIONE, C., AND JOHANSSON, K. 2013. 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.

POLASTRE, J., HILL, J., AND CULLER, D. 2004. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*. SenSys '04. 95–107.

PRAYATI, A., ANTONOPOULOS, C., STOYANOVA, T., KOULAMAS, C., AND PAPADOPOULOS, G. 2010. A modeling approach on the TelosB WSN platform power consumption. *Journal of Systems and Software 83*, 1355–1363.

RICHARDS, A. AND HOW, J. 2004. A decentralized algorithm for robust constrained model predictive control. In *American Control Conference, 2004. Proceedings of the 2004*. Vol. 5. 4261 –4266.

STANKOVIC, J. A., CAO, Q., DOAN, T., FANG, L., HE, Z., KIRAN, R., LIN, S., SON, S., STOLERU, R., AND WOOD, A. 2005. Wireless sensor networks for in-home healthcare: potential and challenges. *Proceedings of High Confidence Medical Device Software and Systems Workshop*.

STEINE, M., GEILEN, M., AND BASTEN, T. 2011a. 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. 25–32.

STEINE, M., GEILEN, M., AND BASTEN, T. 2012. 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*. 739–742.

STEINE, M., VIET NGO, C., SERNA OLIVER, R., GEILEN, M., BASTEN, T., FOHLER, G., AND DECOTIGNIE, J.-D. 2011b. 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. 31–40.

SUN, Y., GUREWITZ, O., AND JOHNSON, D. B. 2008. 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. 1–14.

TAY, T., MAREELS, I., AND MOORE, J. 1998. *High Performance Control*. Springer.

TELOSB DATASHEET, CROSSBOW INC. www.xbow.com. Last visited: April 2013.

TINYOS WEBSITE 2013. www.tinyos.net. Last visited: April 2013.

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

YE, W., HEIDEMANN, J., AND ESTRIN, D. 2004. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *Networking, IEEE/ACM Transactions on 12,* 3, 493–506.

ZHAO, J. AND GOVINDAN, R. 2003. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*. SenSys '03. 1–13.

ZIMMERLING, M., FERRARI, F., MOTTOLA, L., VOIGT, T., AND THIELE, L. 2012. pTunes: runtime parameter adaptation for low-power MAC protocols. In *Proceedings of the 11th International Conference on Information Processing in Sensor Networks*. IPSN '12. 173–184.