TinyAdapt: An Adaptation Framework for Sensor Networks

Daniel Minder, Marcus Handte, Pedro José Marrón

***Abstract*—Although algorithms for wireless sensor networks are usually optimised for a specific set of user preferences under certain network conditions, if the conditions or preferences change during run-time, the intrinsic parameters of the algorithms have to be changed accordingly or the algorithms have to be replaced. However, choosing a suitable set of algorithms and parameters can be a tedious and difficult task to do manually. In this paper, we present TinyAdapt, a novel adaptation framework for sensor networks, that performs this selection autonomously guided by previous results from simulations, testbeds or real deployments, measured network conditions and high-level user preferences. We show that its use in sensor networks improves their performance significantly with minimal overhead and that it increases their flexibility under changing conditions.**

I. Introduction

Nowadays, sensor networks are used in different application domains to monitor a broad range of real-world phenomena, e.g. in logistics, health care, biological studies or smart offices [1]. Despite the broad spectrum of domains, applications share a common set of basic algorithms, e.g. routing, clustering and time synchronisation, that are needed for their basic operation [2]. However, each particular algorithm is better suited for a specific type of environment such as mobile settings and it is optimised for a particular set of user preferences, e.g. latency. When developing an application, it is a difficult task to select the optimal set of algorithms and parameters since the selection requires a high degree of a priori knowledge about the application characteristics, the network conditions and the user preferences on optimisation. Moreover, in many cases these factors can change dynamically over time.

The manual adjustment of algorithms and parameters is usually not feasible. The sensor networks are often intended to operate unattended after their final deployment, the interrelation between high-level user preferences and low-level algorithm parameters might not be obvious, and, finally, understanding the whole parameter space makes it impossible for adaptation to be handled manually.

In this paper, we propose TinyAdapt, an adaptation framework for wireless sensor networks. TinyAdapt performs autonomous adaptation of algorithms by means of selection and parameterisation. The adaptation decisions are based on results obtained from previous simulations, testbed or real deployment runs which enables TinyAdapt to take full advantage of the accuracy provided by them. At run-time, the adaptation is initiated by changes to user preferences and measured network conditions. We show that TinyAdapt can indeed significantly increase the performance of the application under changing conditions.

The authors are with University of Duisburg-Essen, Germany (e-mail: {daniel.minder, marcus.handte, pjmarron}@uni-due.de).

II. Related Work

Since TinyAdapt uses a component abstraction to build adaptive applications component systems are shown first, then we review different adaptation approaches.

SNACK [3] provides a simplified programming language and a service library for sensor network applications. A programmer writes an application by parameterising and combining standard components. In the PCOM [4] middleware a component abstraction and explicit specification of dependencies is used. When starting an application PCOM recursively creates a component tree by trying to satisfy the dependencies of components, which can be located on multiple devices.

Besides single adaptive algorithms, some generic adaptation frameworks exist. In order to cope with the changing condition of its environment Impala [5] regularly gathers application parameters (e.g., number of neighbouring animals, amount of sensor data) and system parameters (e.g. battery level, geographic position) and, using a Finite State Machine, decides whether or not a different protocol should be used. In the reconfiguration architecture based on GRATISplus, DESERT and GRATIS [6] a user can model several possible implementations of the same application. During run-time, critical QoS parameters are gathered in the network and evaluated at the base station. Reconfiguration commands are sent to the nodes specifying which components have to be stopped, rewired and started. The system presented in [7] automates network protocol stack design and its run-time optimisation. The network is configured with several parameters and its performance attributes are measured. These attributes are combined to a single value using a utility function. Both parameters and utility value are fed into a knowledge base and new parameters are selected by simulated annealing.

III. System Description

The adaptation framework TinyAdapt is part of the TinyCubus framework [8] for TinyOS. Its three main components can be recognised in Figure 1: The cross-layer framework TinyXXL [9] that allows to share data between different layers thus enabling cross-layer optimisations, the Installation module as part of the configuration engine FlexCup [10] that distributes components in the network and provides installation support for them, and the Adaptation Engine and Monitoring components as part of the adaptation framework TinyAdapt that manages all available algorithms and/or parameterisations and selects the appropriate one. The figure also shows the data

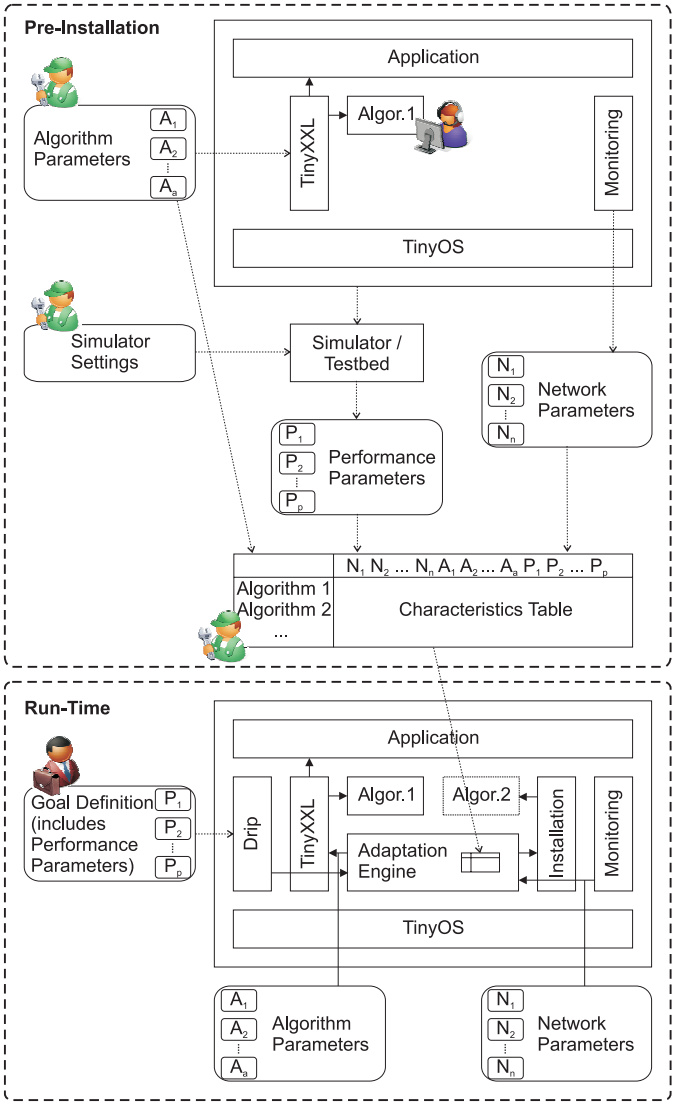


Fig. 1: System Overview

flow between components (solid lines) and the interactions between different users, framework, program, and simulator or testbed controlled by TinyAdapt (dashed lines).

In general, the TinyAdapt framework consists of a pre-installation part and a run-time system. Before installation different algorithms and different parameters for algorithms are evaluated. The results are stored in the Characteristics Table, which is used during run-time in the adaptation engine to select the best algorithm or best parameterisation. The definition of best is given by the user in form of a so called Goal Definition.

A. Pre-installation

During the pre-installation phase, parameterisable algorithms have to be prepared to use TinyXXL and the different algorithms and parameterisations have to be evaluated. Finally, modifications to the Characteristics Table are possible.

1) TinyXXL: TinyAdapt uses the cross-layer framework TinyXXL to parameterise an algorithm or the application. This minimises code dependencies between the algorithm modules and the TinyAdapt framework modules. As [9] points out, the necessary changes to use shared cross-layer instead of private data are minimal and straight forward. If immediate reaction is necessary when a parameter changes the user needs to provide a 'changed' event for this parameter. For example, if a data rate parameter is changed a timer usually needs to be stopped and restarted with the new rate.

2) Algorithm Evaluation: During this step the algorithms are evaluated using a simulator, a testbed or a real deployment. The more the simulation or the testbed resembles the final deployment of the application the better the adaptation engine will perform during run-time.

The engineer guides TinyAdapt by providing the application, the (parameterisable) algorithms, a list of possible values for the algorithm parameters, different simulator settings, and a list of network and performance parameters.

The algorithm parameters A are those that were included in TinyXXL in the preparation step. Since they are specific for each (type of) algorithm TinyAdapt does not make any assumptions. For each parameter, particular values are indicated that should be tested. TinyAdapt will create a new application for each parameter combination, deploy and test it.

When simulation is used several simulator settings have to be provided: for example the number of nodes, their locations, and, if needed, a mobility model or sensor sample data. These simulator settings affect certain network parameters N. Since from a node's perspective the change rate of the neighbours can be quite different than an average node speed from the simulator settings the network parameters are measured during run-time and not derived from the simulator settings. The network parameters currently implemented are Number of Neighbours and Mobility. If static scenarios are evaluated the user can also decide to switch off the Monitoring component.

The algorithms are evaluated with respect to several metrics, which we call performance parameters P. Some parameters are general for all algorithms, e.g. power consumption, others are specific for one class of algorithms, e.g. delivery ratio for routing algorithms. Currently, the parameters Power Consumption, Delivery Ration and Latency are implemented.

B. Characteristics Table

All three parameters sets A, N and P are collected for each algorithm in the Characteristics Table. After the algorithm evaluation part this table largely describes the behaviour of an algorithm with specific parameters under various conditions with respect to specific performance metrics.

In a post-processing step the table is examined and irrelevant parameter values are filtered out. A value is considered irrelevant if it does not influence the performance parameters significantly and instead a neighbouring value can be used. A curtailing algorithm suggests such values and presents them to the application engineer together with the error introduced by removing this value. He finally decides if he wants to have a more compact characteristics table by unifying more parameter values while accepting a possibly higher error and, therefore, a less accurate adaptation result or if he wants to have a larger table supporting more accurate adaptation.

C. Run-time System

1) Architecture: Several components have been added in the adaptable program compared to the test program. The Adaptation Engine is the core since it communicates with the other components and controls the adaptation process. The Characteristics Table is included here as well. As in the test program TinyXXL is connected with the parameterisable algorithms and application, but now new algorithm parameter values A can be set by the Adaptation Engine. The Monitoring component is also still present but it reports the network parameters N to the Adaptation Engine. The Drip component is responsible for disseminating a Goal Definition in the network and passing it to the Adaptation Engine. Finally, the Adaptation Engine can instruct the Installation component to replace the currently running algorithm by a different one.

2) Adaptation process: When a new Goal Definition arrives or network parameters change the adaptation process is triggered. A Goal Definition consists of constraint, relaxation and optimisation definitions. Constraints define requirements that the user of a sensor network has and that the application has to fulfil. They can be expressed as an inequality using the elements of P. For example, the user could specify a maximum power consumption of 0.4W and a delivery ratio of at least 80%.

When applying these constraints and the current values of the network parameters to the Characteristics Table three cases can happen: Exactly one matching algorithm or parameterisation is found, more than one or none at all. The first case is the simplest since we can directly select this solution. TinyAdapt also provides a way for resolving both other cases: If no algorithm is found, the user can provide a list of constraints which can be relaxed, e.g. the delivery ratio in steps of 5% until 50% are reached. After each relaxation step the (new) constraints are re-evaluated until at least one matching algorithm or parameterisation is found.

If eventually more than one algorithm or parameterisation fits (with or without relaxation), the user can specify an optimisation parameter and a direction. For example, the power consumption could be minimised.

The decision of the adaptation engine is communicated to other nodes to get a common agreement on how to adapt. Since it is not needed for the considered scenario we do not present it in this paper due to space restrictions.

The selected algorithm is installed or switched to if necessary. If the Characteristics Table for this algorithm contains parameters A, the algorithm has to be parameterised with them. For this purpose TinyAdapt sets the new value using TinyXXL. TinyXXL, in turn, notifies the parameterisable algorithms that one of its parameters has changed. The algorithm is then responsible for reading the new value from TinyXXL and acting accordingly.

IV. Evaluation

To test the TinyAdapt approach we consider security monitoring in a harbour by means of sensor nodes that are attached to the containers. The general objective of the application is to monitor the containers for a longer time without further intervention. In case suspicious conditions (e.g., vibrations or noise) are detected the harbour logistics centre can decide to select a higher data rate with a more robust transmission scheme. Of course, this will consume more energy than the long-term observation scheme and is, therefore, not suitable for normal operation.

We have built our application on TinyOS 2.1.0 using the Collection Tree Protocol (CTP). We changed the original TinyOS implementation of CTP so that the maximum beacon interval and the maximum number of retries when sending a data packet from node to node are stored in TinyXXL. Additionally, we enabled CC2420's Low Power Listening (LPL) and wired it to our LPL control module. TinyAdapt is integrated as module and interwoven by TinyXXL with the main application, CTP and LPL modules to share the algorithm parameters as explained in Section III-A1.

A. Pre-installation phase

The basic simulations are used to create the Characteristics Table as described in Section III-A2. All simulations are done using Avrora 1.7.110. MicaZ nodes are emulated, radio links are simulated using the lossy model. In general, communication is possible between the horizontal and vertical neighbours of a node. For the simulations, we abstract from the piling of the containers and place 25 MicaZ nodes in a 5x5 grid. The base station is located in one corner of this grid. Each node regularly sends measurements to the base station.

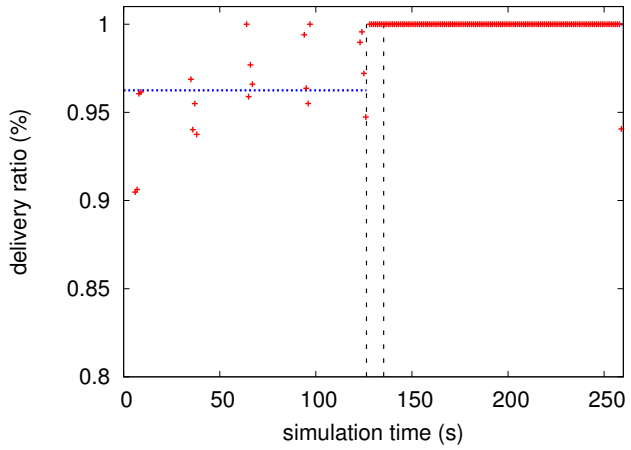
Four different algorithm parameters (data interval of application, maximum number of CTP retransmissions, maximum CTP beacon interval, low power listening interval) are tested with several different values. As performance parameters, the standard parameters as described in Section III-A2 are evaluated. Details on the parameters and on the curtailing step are omitted due to space restrictions.

B. Run-time results

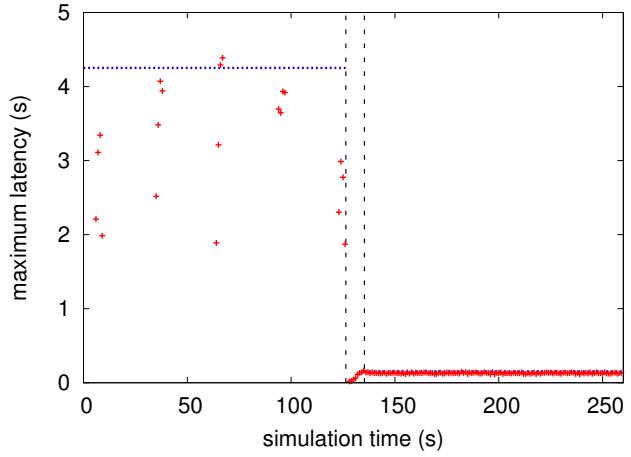
When building the application the start Goal Definition is evaluated and the parameters are set as default. In our example of a long-term observation application the Goal Definition requires a minimal delivery ratio of 90% and advises to optimise energy, which leads to a data packet interval of 30s and a LPL interval of 250ms. In the simulation, nodes are booted randomly at the beginning and after 5s setup time each node starts sending data packets regularly.

After 120 seconds, the user decides to switch from the long-term observation mode to a fine-granular observation. Therefore, a new packet interval of 2s, a maximum latency of 1s and delivery ratio as new optimisation goal is distributed to the network via the base station. TinyAdapt reacts to the changed Goal Definition, selects a different parameterisation (data interval 2s, LPL off) and configures the algorithms accordingly. The simulation stops after another 130s.

In the presented application, the Characteristics Table needs 448 bytes of RAM, which can be further reduced since we did not remove constant parameters. The new Goal Definition that is distributed in the network is only 13 bytes long. The adaptation process takes 1.32ms only.



(a) Delivery ratio



(b) Latency

Fig. 2: Performance parameters over time

The results of 25 simulation runs are shown in Figure 2. The two vertical lines always indicate the time span between the adaptation of the first node and the last node of all simulations and the horizontal lines specify the performance value for the currently active parameter combination obtained from the basic simulations. Figure 2a shows the delivery ratio of all packets sent in a 1s interval. The actual 1-second ratios are distributed around the prediction line, but the mean value of 95.9% is only slightly worse than the prediction of 96.3% from the basic simulations. After the nodes have adapted, all packets are received by the base station.

In Figure 2b the maximum latency of all packets sent in a 1s interval is shown. As can be seen from the graph the prediction values well establish an upper bound for latency for all transmissions; only twice the bound is exceeded minimally.

The basic simulations predict a needed energy of 52.22J for the first 130s. Since the adaptation starts approximately 4s earlier and since the adaptive application also includes Drip the needed energy is a little higher (57.88J). For the next 130s, 215.07J are required which is only slightly more than estimated 213.71J according to the basic simulations. When LPL is off, there is almost no difference if the radio is receiving or sending. Therefore, the additional Drip component has only a minor influence on the overall energy.

V. Conclusion and Future Work

We presented TinyAdapt that is able to adapt various parameters of lower-layer algorithms so that the application works optimally under changed conditions and/or changed user preferences. Therefore, different algorithms and their parameterisations are evaluated in a pre-installation phase. During run-time, this information is used to perform autonomous adaptation in an efficient and goal-oriented way.

In future work we will optimise the monitoring framework to better assess the characteristics of non-static environments. We will also examine situations where different parts of a network exhibit very different characteristics and, therefore, should be parameterised in a different way.

Acknowledgements

This work has been partially supported by CONET, the Cooperating Objects Network of Excellence, funded by the European Commission under FP7 with contract number FP7-2007-2-224053.

References

[1] K. Römer and F. Mattern, “The design space of wireless sensor networks,” *IEEE Wireless Communications*, vol. 11, 2004.

[2] P. Marron, D. Minder, and The Embedded WiSeNts Consortium, “Embedded wisents research roadmap,” IST/FP6 (IST-004400), 2006.

[3] B. Greenstein, E. Kohler, and D. Estrin, “A sensor network application construction kit (SNACK),” in *Proceedings of ACM SenSys ’04*, 2004.

[4] C. Becker, M. Handte, G. Schiele, and K. Rothermel, “PCOM - A Component System for Pervasive Computing,” in *Proc. of the 2nd IEEE Conference on Pervasive Computing and Communications (PERCOM’04)*, 2004.

[5] T. Liu and M. Martonosi, “Impala: A middleware system for managing autonomic, parallel sensor systems,” in *Proc. of the 9th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, 2003.

[6] S. Kogekar, S. Neema, B. Eames, X. Koutsoukos, A. Ledeczi, and M. Maroti, “Constraint-guided dynamic reconfiguration in sensor networks,” in *Proceedings of the third international symposium on Information processing in sensor networks (IPSN’04)*. New York, NY, USA: ACM Press, 2004.

[7] E. Meshkova, A. Achtzehn, J. Riihijärvi, and P. Mähönen, “Towards a user-centric network optimization engine,” in *Poster Proc. of 6th IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON 2009)*, 2009.

[8] P. J. Marrón, D. Minder, A. Lachenmann, and K. Rothermel, “TinyCubus: An adaptive cross-layer framework for sensor networks,” *it - Information Technology*, vol. 47, no. 2, 2005.

[9] A. Lachenmann, P. J. Marrón, D. Minder, M. Gauger, O. Saukh, and K. Rothermel, “TinyXXL: Language and runtime support for cross-layer interactions,” in *Proc. of the 3rd IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON 2006)*, 2006.

[10] P. J. Marrón, M. Gauger, A. Lachenmann, D. Minder, O. Saukh, and K. Rothermel, “Flexcup: A flexible and efficient code update mechanism for sensor networks,” in *Proc. of the 3rd European Workshop on Wireless Sensor Networks (EWSN 2006)*, 2006.