TinyCubus: A Flexible and Adaptive Cross-Layer Framework for Sensor Networks

Pedro José Marrón, Daniel Minder, Andreas Lachenmann, and Kurt Rothermel

University of Stuttgart
Institute of Parallel and Distributed Systems (IPVS)
Universitätsstr. 38
D-70569 Stuttgart
{marron,minder,lachenmann,rothermel}@informatik.uni-stuttgart.de

1 Introduction

With the proliferation of sensor networks and sensor network applications during the last few years, the overall complexity of such systems is continuously increasing. Sensor networks are now heterogeneous in terms of their hardware characteristics and application requirements even within a single network. In addition, the requirements of current applications are expected to change over time. All of this makes developing, deploying, and optimizing sensor network applications an extremely difficult task. In the `TinyCubus` project we research a necessary infrastructure to support the complexity of such systems.

`TinyCubus` consists of a data management framework, a cross-layer framework, and a configuration engine. The *data management framework* allows the dynamic selection and adaptation of system and data management components. The *cross-layer framework* supports data sharing and other forms of interaction between components in order to achieve cross-layer optimizations. The *configuration engine* allows code to be distributed reliably and efficiently by taking into account the topology of sensors and their assigned functionality.

2 Overall Architecture

The overall architecture of `TinyCubus` mirrors the requirements imposed by the applications and the underlying hardware. As shown in figure 1, `TinyCubus` is implemented on top of TinyOS [3] using the nesC programming language [1], which allows for the definition of components that contain functionality and algorithms. We use TinyOS primarily as a hardware abstraction layer. For TinyOS, `TinyCubus` is the only application running in the system. All other applications register their requirements and components with `TinyCubus` and are executed by the framework.

2.1 Tiny Data Management Framework

The Tiny Data Management Framework provides a set of data management and system components, selected on the basis of the typically data-driven nature of

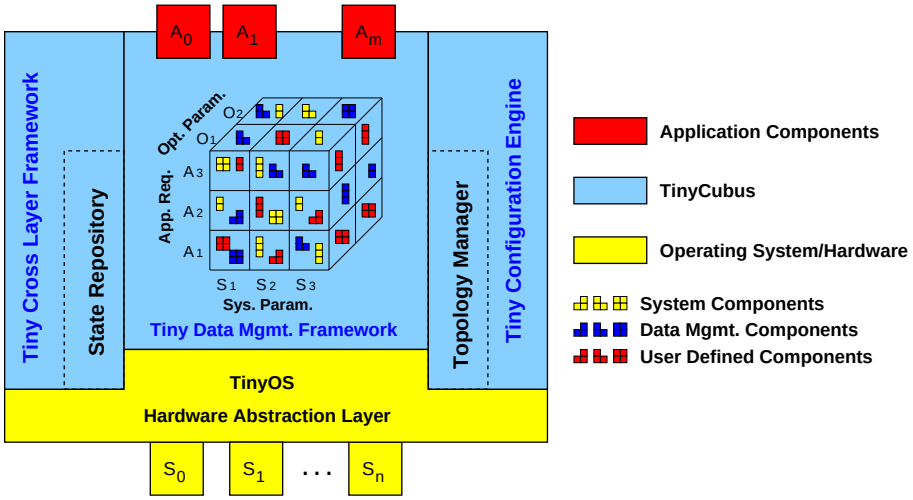

Fig. 1. Architectural components in `TinyCubus`

sensor network applications. For each type of standard data management component such as replication/caching, prefetching/hoarding, aggregation, as well as each type of system component, such as time synchronization and broadcast strategies, it is expected that several implementations of each component type exist. The Tiny Data Management Framework is then responsible for the selection of the appropriate implementation based on the information obtained from the system.

The cube of figure 1, called 'Cubus', combines *optimization parameters*, such as energy, communication latency, and bandwidth; *application requirements*, such as reliability; and *system parameters*, such as mobility. For each component type, algorithms are classified according to these three dimensions. For example, a tree based routing algorithm is energy-efficient, but cannot be used in highly mobile scenarios with high reliability requirements. The component implementing the algorithm is tagged with the combination of parameters and requirements for which the algorithm is most efficient. Eventually, for each combination a component will be available for each type of data management and system components.

The Tiny Data Management Framework selects the best suited set of components based on current system parameters, application requirements, and optimization parameters. This adaptation has to be performed throughout the lifetime of the system and is a crucial part of the optimization process.

2.2 Tiny Cross-Layer Framework

The Tiny Cross-Layer Framework provides a generic interface to support parameterization of components using cross-layer interactions. Strict layering (i.e.,

each layer only interacts with its immediately neighboring layers) is not practical for wireless sensor networks [2] because it might not be possible to apply certain desirable optimizations. For example, if some of the application components as well as the link layer component need information about the network neighborhood, this information can be gathered by one of the components in the system and provided to all others.

If layers or components interact with each other, there is the danger of loosing desirable architectural properties such as modularity. Therefore, in our architecture the cross-layer framework acts as a mediator between components. Cross-layer data is not directly accessed from other components but stored in a state repository.

Other examples of cross-layer interactions are callbacks to higher-level functions, such as the one provided by the application developer. TinyOS already provides some support with its separation of interfaces from implementing components. However, the TinyOS concept for callbacks is not sophisticated enough for our purposes, since the wiring of components is static. With `TinyCubus` components are selected dynamically and can be exchanged at runtime. Therefore, both the usage of a component and callbacks cannot be static; they have to be directed to the new component if the data management framework selects a different component or the configuration engine installs a replacement for it. `TinyCubus` extends the functionality provided by TinyOS to allow for the dereferencing and resolution of interfaces and components.

2.3 Tiny Configuration Engine

In some cases parameterization, as provided by the Tiny Cross-Layer Framework, is not enough. Installing new components, or swapping certain functions is necessary, for example, when new functionality such as a new processing or aggregation function for the sensed data is required by the application. The Tiny Configuration Engine addresses this problem by distributing and installing code in the network. Its goal is to support the configuration of both system and application components with the assistance of the topology manager.

The topology manager is responsible for the self-configuration of the network and the assignment of specific roles to each node. A role defines the function of a node based on properties such as hardware capabilities, network neighborhood, location etc. A generic specification language and a distributed and efficient role assignment algorithm is used to assign roles to the nodes.

Since in most cases the network is heterogeneous, the assignment of roles to nodes is extremely important: only those nodes that actually need a component have to receive and install it. This information can be used by the configuration engine, for example, to distribute code efficiently in the network.

3 Application Studies

Three specific sensor network applications play an important role in the research performed at the University of Stuttgart: Sustainable Bridges [5], Cartalk 2000

[4], and our in-house application called 'Sense-R-Us'. These applications are being studied as canonical examples for a wide range of applications that deal with static and mobile sensor nodes. Their analysis allows us to identify requirements and characteristics that apply to applications that fall in this category.

The goal of the Sustainable Bridges project is to provide cost-effective monitoring of bridges using static sensor nodes in order to detect structural defects as soon as they appear. A wide range of sensor data is needed to achieve this goal, e.g., temperature, relative humidity, swing level, vibrations, as well as noise detection and localization mechanisms to determine the position of cracks. Nodes are stationary and are placed manually at well-known positions. In order to perform the localization, nodes sample noise emitted by the bridge at a rate of 40 kHz and, by using triangulation methods, the position of the possible defect is determined. This process requires the clocks of adjacent sensors to be synchronized within 15-25 μs of each other. Finally, sensors are required to have a lifetime of at least 3 years so that batteries can be replaced during the regular bridge inspections.

In contrast, the goal of the Cartalk 2000 project is to develop a cooperative driver assistance system based on mobile ad-hoc inter-vehicle communication. The vehicle should warn the driver of road obstacles, slowing-down traffic, or bad road conditions a vehicle ahead has detected. It supports the driver on highway entry, lane merging, and in right of way scenarios when the situation can become unclear. Since sensors are integrated into cars, they are mobile with respect to each other. A wide range of highly dynamic sensor data, e.g., speed, position, and tire pressure, is gathered continuously. The processing of data must be performed in a timely manner and immediately sent to other drivers that might be interested in it. Thus, time-constrained communication is important for the system since data must be forwarded to the appropriate cars at the right time. In contrast to the Sustainable Bridges application, energy constraints are less severe in this application since sensor nodes are directly connected to the electric system of the car.

The 'Sensor-R-Us' application aims at building a Smart Environment in our department using motes. Base stations are installed in all rooms and serve as location beacons and gateways to the mobile nodes carried around by the employees. Using a PC based GUI, the position and status ('in a meeting', e.g.) of persons or sensor information like room temperature can be queried. The query processing capability of Sensor-R-Us is much higher than of the other two applications: In addition to simple selection, projection, and aggregation, queries can be stored in a node, and their execution can be triggered by a sensor or timer event. Results of a query can also be stored in a node and be used as 'virtual sensors'.

3.1 Comparison of the Applications

All applications need some common functionality: routing, query processing, and data aggregation. Concerning Query Processing and Data Aggregation, they use simple queries that may be triggered by sensor or timer events and simple

aggregation functions. Sense-R-Us needs the most advanced Query Processing and Data Aggregation component to provide local storage of query results and more advanced or user-defined aggregation functions.

Routing is needed in all applications, but differs the most. In Sustainable Bridges a standard energy-efficient routing takes care of clusters and roles, CarTalk uses two different geographic routing algorithms, and Sense-R-Us needs a routing scheme which must come to terms with hybrid network topologies.

Nevertheless we have identified three functional blocks that can be implemented as components and stored in the Cubus.

3.2 Dynamics of Parameters

By analyzing these three applications, we give reasons for the dimensions of the Cubus.

In the Sustainable Bridges application, small Telos motes with only 48kB program flash are used. Several complex functions for analyzing the sampled data are needed which do not fit into the flash at the same time. The good news is that their usage frequency varies highly. The TinyCubus monitors the usage (system parameter) and can replace scarcely used components with dummies to optimize for low program size.

The system environment of CarTalk changes rapidly: a vehicle can wait with hundreds of other vehicles on a car park or it can run on a lonely highway, meeting others only every 10 minutes. An adaptive routing and also probably a clustering algorithm is obviously needed. When driving, energy is not a very important problem, this changes if the vehicles does not move for some time. In contrast, in circulating traffic CarTalk has high latency requirements which do not apply while being on a parking lot. Thus, optimization criteria change.

In Sense-R-Us a mobile node of an employee might send information regularly to the base station in the employee's own room. While being in this room, the information can be sent directly. But if the application detects that the mote is outside the room of the employee for at least a few minutes, it tightens its security requirements. The Cubus will adapt to this changed application requirement and install a component that will sign each message to the base station to proof its authenticity. On the system parameter side, topology changes will occur which affects routing and possibly clustering.

4 Conclusion and Future Work

In this paper, we have described the architecture of `TinyCubus`, a flexible, adaptive cross-layer framework for sensor networks. Its specific requirements have been derived from the increasing complexity of the hardware capabilities of sensor networks, the variety and breadth found in typical applications, and the heterogeneity of the network itself. Therefore, we have designed our system to have the Tiny Data Management Framework, that provides adaptation capabilities, the Tiny Cross-Layer Framework, that provides a generic interface and a

repository for the exchange and management of cross-layer information, and the Tiny Configuration Engine, whose purpose is to manage the upload of code onto the appropriate sensor nodes. We have underpinned the design of the cubus with the analysis of three current applications regarding functionality and dynamic parameters.

The implementation of `TinyCubus` is still under way and, although the prototypes for the cross-layer framework and configuration engine are already partially functional, there is still some work to do.

References

1. D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proc. of PLDI'03*, pages 1–11, 2003.
2. A. J. Goldsmith and S. B. Wicker. Design challenges for energy-constrained ad hoc wireless networks. *IEEE Wireless Communications*, 9(4):8–27, 2002.
3. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proc. of the 9th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.
4. D. Reichardt, M. Miglietta, L. Moretti, P. Morsink, and W. Schulz. CarTALK 2000: Safe and comfortable driving based upon inter-vehicle-communication. In *Proc. of the Intelligent Vehicle Symp.*, volume 2, pages 545–550, 2002.
5. Sustainable bridges web site. http://www.sustainablebridges.net.