Adaptation and Cross-Layer Issues in Sensor Networks

[#] Pedro Marrón, Andreas Lachenmann, Daniel Minder, Jörg Hähner, Kurt Rothermel, Christian Becker
IPVS, Universität Stuttgart
Universitätsstr. 38, D-70569 Stuttgart, Germany
{*marron,lachenmann,minder,haehner,rothermel*}*@informatik.uni-stuttgart.de*

Abstract

An intrinsic characteristic of current projects in the area of sensor networks is the heterogeneity of hardware and application requirements. In addition, the requirements of current applications are expected to change over time. This makes developing, deploying, and optimizing sensor network applications an extremely difficult task. In this paper, we present the architecture of `TinyCubus`*, a flexible and adaptive cross-layer framework for TinyOS-based sensor networks that aims at providing the necessary infrastructure to cope with the complexity of such systems.* `TinyCubus` *consists of three parts: a data management framework that selects and adapts both system and data management components, a cross-layer framework that enables optimizations through cross-layer interactions, and a configuration engine that installs components dynamically. We show the feasibility of our architecture by describing and evaluating a code distribution algorithm that optimizes its behavior by using application knowledge about the sensor topology.*

1. Introduction

In the last few years wireless sensor networks have been proposed as a way to unobtrusively gather real-world data. A sensor network consists of small networked devices equipped with sensors. Each node is able to sense the physical world and process data in the network and transmit it using multi-hop communication. Since most nodes are resource-constrained, energy consumption and, in general, efficient resource management plays an important role.

In order to acquire data, sensor networks use various kinds of hardware devices. Although many research groups use Berkeley Motes together with TinyOS [1], there is no standard platform for sensor nodes yet, which leads to a heterogeneity in hardware.

Likewise, applications are continuously evolving and are, therefore, highly heterogeneous. New applications continue to arise and although there are similarities, each of them has its own specific requirements.

The network itself, defined as a collection of devices, might also be heterogeneous: In more recent applications, a network often consists of different devices that are able to perform different tasks and is no longer considered a homogeneous environment. For example, some nodes are equipped with special kinds of sensors, whereas others may have more processing power for complex calculations or act as gateways to infrastructure-based networks. Furthermore, if application requirements change or another application is executed, the network has to adapt. However, developing adaptation for every application and optimizing the code over and over again are complex, error-prone tasks. In order to simplify application development, system software in the form of a flexible, adaptive framework that supports a large number of hardware platforms and applications is clearly needed.

In this paper we present the architecture of `TinyCubus`, which aims at providing the necessary infrastructure to deal with 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. Finally, the *configuration engine* allows code to be distributed reliably and efficiently by taking into account the topology of sensors and their assigned functionality.

The contribution of this paper is twofold. First, we describe the architecture of `TinyCubus`, a flexible, adaptive cross-layer framework for sensor networks. Secondly, we describe and evaluate a code distribution algorithm used by the configuration engine to disseminate components and code reliably and efficiently within the network, using the cross-layer data provided by the framework. The results of our evaluation show that our algorithm reduces the number of messages exchanged if the topology of the network is structured and known to the application.

The remainder of this paper is structured as follows. The next section presents the overall architecture of our framework and gives more detailed information about its three parts. Section 3 describes and evaluates the code distribution algorithm used by the configuration engine. Section 4 gives an overview of related work and section 5 concludes this paper and describes future directions.

2. Overall Architecture

The overall architecture of `TinyCubus` has been developed with the goal of creating a generic reconfigurable framework for sensor networks. As shown in figure 1, `TinyCubus`

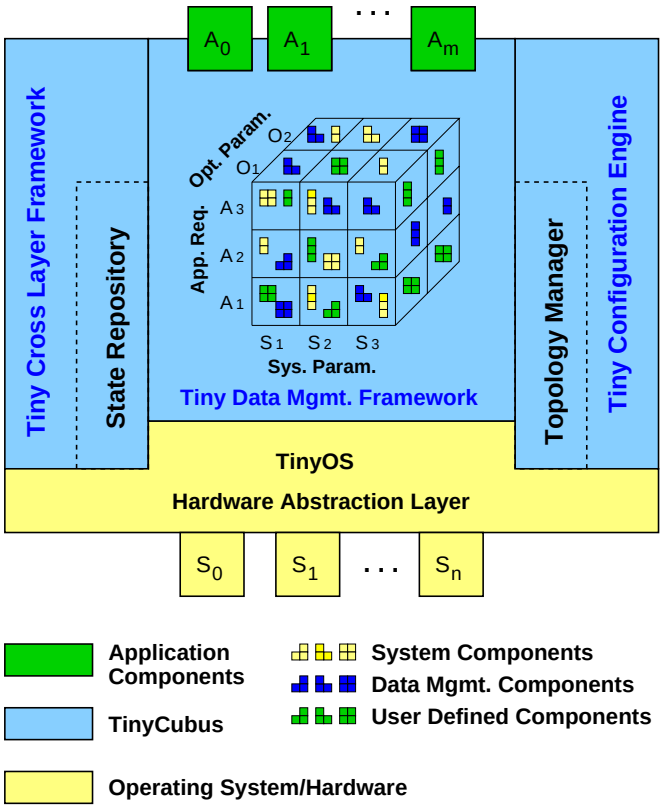

Fig. 1: Architectural components in `TinyCubus`

is implemented on top of TinyOS [1] using the nesC programming language [2], which allows for the definition of components in the sense of TinyOS. From the point of view of TinyOS, `TinyCubus` is the only application running in the system. All other applications register their requirements within `TinyCubus` and are executed by the framework.

`TinyCubus` itself consists of three parts: the Tiny Data Management Framework, the Tiny Cross-Layer Framework, and the Tiny Configuration Engine, which are described in the following sections.

A. Tiny Data Management Framework

The Tiny Data Management Framework provides a set of data management and system components. 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 current information contained in 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.

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. We are currently investigating different strategies that determine when it is necessary – and beneficial – to select a different component and where this component should be loaded from.

Furthermore, the parameters and requirements in the three dimensions of the Cubus (system parameters, application requirements, and optimization parameters) have to be carefully selected. Regarding the *system parameters*, we analyze which of them can be measured by a sensor node. In the simplest case these observations are purely local, such as the number of neighbors and their mobility, but in some cases it might be needed to disseminate certain information. By examining sensor network applications we determine the *application requirements*. In the broadest sense, they can be subsumed under the term 'quality of service'. Examples are consistency, accuracy, reliability, and real-time constraints. Finally, the *optimization parameters* describe how an algorithm distinguishes itself from other algorithms under the same system and application parameters. These can be latency, communication, and energy.

B. Tiny Cross-Layer Framework

The Tiny Cross-Layer Framework provides a generic interface to support the parameterization of components that use cross-layer interactions. As described in [3], strict layering is not practical for wireless sensor networks 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. Other examples for cross-layer interactions are callbacks to higher-level functions, such as the ones provided by the application developer. The Tiny Cross-Layer Framework provides support for both forms of interaction. To deal with callbacks and dynamically loaded code, `TinyCubus` extends the functionality provided by TinyOS to allow for the dereferencing and resolution of interfaces and components.

1) State Repository: 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 the state repository. Thus, if a component is replaced (e.g., to adapt to changed requirements), no component that uses the old component's cross-layer data is affected by the change, given that the new component also provides the same or compatible data. We

expect that most components available in the framework will be developed with cross-layer optimizations in mind.

For this, components must know what cross-layer data is available in the state repository. For this purpose we use a specification language which allows us to specify what cross-layer data a component needs and provides. With this specification components that make cross-layer data available can also determine if other ones use their data and if they have to gather the data at all.

2) Callbacks: Regarding callbacks to other components, TinyOS already provides some support with its separation of interfaces from the implementation of 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.

C. 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 and role assignment algorithms.

1) 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. Examples for roles are `SOURCE`, `AGGREGATOR`, and `SINK` for aggregation applications, `CLUSTERHEAD`, `GATEWAY`, and `SLAVE` for clustering applications as well as `VIBRATION` to describe the sensing capabilities of a node. In previous work [4] we describe a generic specification language and an algorithm for efficient role assignment that are briefly outlined in the remainder of this section.

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. As we show in Section 3, this information can be used by the configuration engine, for example, to distribute code efficiently in the network.

2) Role Specification and Role Assignment Algorithm: In order to assign roles to nodes in the network, the topology manager uses a generic specification language and a decentralized role assignment algorithm. In the specification language a role is defined by a rule. If a rule is satisfied, the algorithm assigns the role to the node. For example, the following rule assigns the role `CLUSTERHEAD` if there is no other node with this role in its 1-hop neighborhood:

```
CLUSTERHEAD :: {
  count(1-hop) {role == CLUSTERHEAD} == 0
}
```

Whenever possible the role assignment algorithm only uses local knowledge. However, if information about the network neighbors is required (e.g., the number of nodes in the neighborhood with a given role), the node has to retrieve this information from its neighbors while avoiding conflicting role assignments (see [4] for details).

3. Role-Based Code Distribution Algorithm

Let us now describe a code distribution algorithm that makes use of cross-layer information to optimize the number of messages sent to perform a code update. In many sensor network applications the topology of the roles in the network is known in advance and follows a regular structure. This is definitely the case if roles are defined with routing in mind, such as with clustering approaches. Of course, in the general case, roles can be based on other properties of the application or the system at hand. A good example is provided by the Sustainable Bridges application (Fig. 2), where nodes affixed to the edge are equipped with vibration sensors, whereas others are only required to provide temperature readings. The goal of the Sustainable Bridges application is to provide a cost-effective monitoring of bridges using static sensor nodes in order to detect structural defects. A wide range of sensor data such as temperature, humidity, vibration and noise detection and localization mechanisms are needed to achieve this goal.

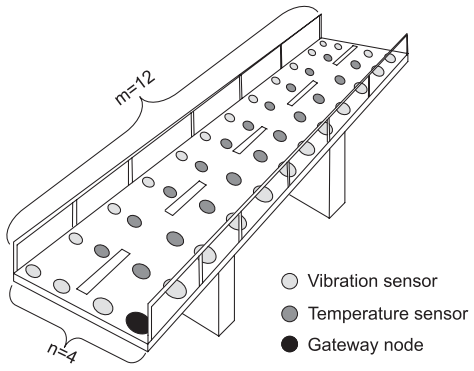

Fig. 2: Sensor topology for Sustainable Bridges

Having information about different roles, and assuming that, in most cases, a difference in role assignment is motivated by differences in functionality, our code distribution algorithm can leverage this knowledge to route code updates only through the set of nodes that really need it, that is, belong to a specific role. In other words, if the code for nodes with vibration sensors is updated, for example, because a new in-network vibration data processing algorithm is needed, this should not affect the temperature nodes available in the system. Of course, the code distribution algorithm has to make sure that all nodes receive the appropriate messages reliably so that, in the end, they all run the same version of the application.

Thinking of the severe energy constraints of sensor nodes in this particular application, and taking into account that the energy cost for data transmission is very high, a scheme that can reduce the number of messages sent unnecessarily to irrelevant nodes is definitely beneficial.

A. Detailed Description

Our code distribution algorithm uses the information about role assignments provided by the Tiny Cross-Layer Framework to efficiently disseminate code updates to specific roles. The algorithm starts at gateway nodes by broadcasting data to its k_r-hop neighborhood, where r is a role and k_r is a parameter that determines the number of hops the algorithm is able to tolerate over nodes with a different role from r. Then, only nodes with role r forward this data further to their own k_r-hop neighbors, thus flooding the nodes with role r while using only those nodes with other roles that are necessary to reach them. The algorithm can be parametrized by selecting k_r for each role. The topology of the network is, therefore, crucial. If, such as for the case depicted in Fig. 2, the network is at most 1-hop connected for a given role r, it is possible to reach all target nodes with maximum efficiency. However, in the general case, especially if topologies are random, other nodes with roles different from r need to be involved in the process of forwarding this information.

In addition, the distribution algorithm makes use of implicit acknowledgments. If a neighbor forwards a message sent by node n, n treats this message as an acknowledgment. If after a certain amount of time, the neighbor does not forward the message, n retransmits it. Following the modularization techniques advocated at the beginning of this paper, this reliability component of our algorithm can be replaced with any other scheme that ensures reliable transmissions.

Finally, in our algorithm, a node n waits a random time $t \in [0, \ldots, t_{max}]$ before retransmitting a message. This is just one possible way to avoid the broadcast storm problem, mentioned in [5] and, like the reliability component, can be replaced with any other scheme that avoids collisions. Of course, the choice of t_{max} is directly related with the delay observed in the evaluation of the algorithm.

In summary, our role-based dissemination algorithm has three settable parameters that, in our system, are maintained by the Tiny Cross-Layer Framework: r, the role of the target nodes; k_r, the network connectivity used for broadcasting data; and t_{max} that determines the maximum retransmission time.

Assumptions: In the implementation of our algorithm, we assume that roles have already been assigned and that there is no dynamic reassignment of roles while the code dissemination algorithm runs. This means that the connectivity k_r of the network for a given role r can be determined upfront. Furthermore, we assume that nodes are stationary, do not fail, and have already determined their neighborhood with respect to a given role r and network connectivity k. Finally, communication is assumed to be performed via bidirectional local broadcasts and that transmission failures, if they occur, are not permanent.

B. Evaluation

In order to show the feasibility of our approach, we have implemented the role-based code distribution algorithm for motes running TinyOS [1]. In our experiments, we compare the efficiency of our algorithm with a flooding approach that has been modified to provide reliability and collision avoidance. The results presented in this paper have been obtained using TOSSIM, the TinyOS simulator provided by UC Berkeley [6].

1) Experimental Setup: For our experiments, we have analyzed the following scenario: Sensor nodes are laid out in an evenly spaced 12×4 grid with the role assignment depicted in Fig. 2, which represents the topology of the Sustainable Bridges application [7]. There is only one gateway node, located in one of the corners, used to inject messages to the network. The distance between the nodes is 10 meters and their radio model is set to a lossless disc model with a communication range of 15 meters. Finally, packet losses occur only due to collisions and the maximum retransmission delay t_{max} has been set to 150 ms and 600 ms respectively.

In our scenarios, we assume the presence of two roles: `VIBRATION` and `TEMPERATURE`, that represent the two types of sensors found in the network. We evaluate the code distribution algorithm by sending (fictitious) code updates from the gateway node to all vibration sensors.

2) Performance Results: Fig. 3 shows the number of messages sent on average by each node in the Sustainable Bridges scenario. The graph compares the messages sent by both flooding and our role-based distribution algorithm for maximum retransmission delay $t_{max} = 150ms$ and $600ms$, respectively. Role assignments on the x-axis vary from the original configuration depicted in Fig. 2 to all nodes being assigned the `VIBRATION` role. The measurements shown are the average of 100 runs. In the graph, we can see that flooding with $t_{max} = 150ms$ requires about 5 messages per node, whereas with $t_{max} = 600ms$, it requires only a little over 2 on average. Since the flooding algorithm retransmits messages in the presence of collisions until all nodes are reached, the average number of messages sent is greater than 1 and varies with the length of t_{max}. In addition, the graph shows that the number of messages sent is independent of the ratio of vibration to temperature sensors, since flooding does not distinguish among them to distribute data.

In contrast, our role-based algorithm performs much better than flooding, especially when the ratio of vibration to temperature sensors is low, since only vibration sensors are required to forward messages[1]. As expected, the number of messages per node increases as the ratio of vibration to temperature sensors increases. In the extreme (when all nodes in the network are vibration nodes), our algorithm behaves just like flooding.

Fig. 4 depicts the average delays needed by both algorithms to reach all vibration nodes. Maximum delays (not shown in

[1]Recall that the network topology in this scenario exhibits 1-hop connectivity for the `VIBRATION` role.

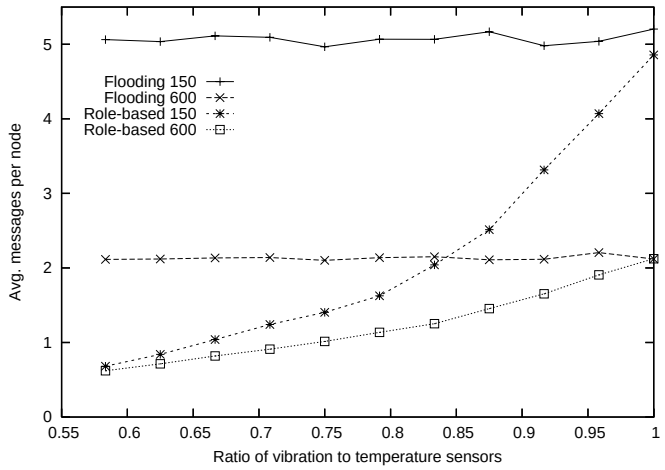

Fig. 3: Avg. number of sent messages per node

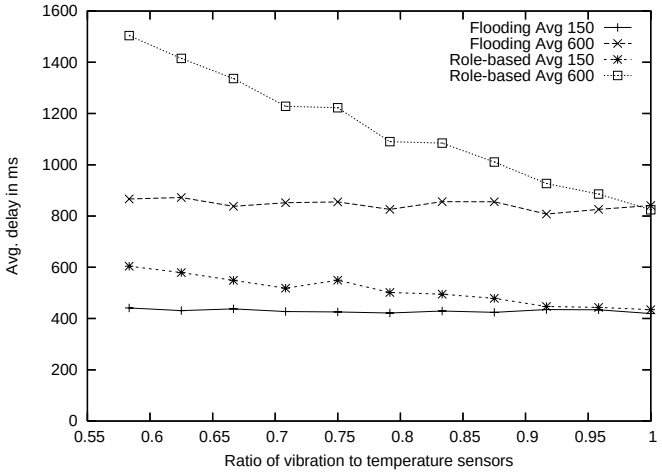

Fig. 4: Avg. delay for message delivery

the graph) are for our algorithm in the worst case as much as twice as long as the delay needed on average. In addition, average delays for flooding are at most 1.5 times better than our role-based algorithm. The reason is that flooding uses not only vibration nodes to forward the data (which allows for more parallelism), and the fact that in our network all vibration nodes are located in a square so that, if one vibration node chooses a long random delay to avoid collisions, data distribution as a whole is delayed. Nevertheless, by choosing for example $t_{max} = 150ms$, it is possible to keep the number of sent messages low (see Fig. 3), while achieving delays just slightly above those of flooding (compare Flooding Avg 150 and Role-based Avg 150 in Fig. 4).

C. Advantages of Role-Based Code Distribution

As we have seen in the evaluation section, the results provided by our role-based algorithm are very promising for structured scenarios. For these cases, we can use application knowledge about the topology of the network to improve on the number of messages sent while maintaining reliability.

In general, our algorithm can be used to distribute any kind of data whose destination varies based on information such as roles. Furthermore, if we assume that roles have already been assigned, our algorithm is more efficient than plain flooding.

Even in the case where nodes are mobile and their distribution changes, if we assume that the ratio of r to all other nodes is high enough, we could try to determine values of k_r that work well. Furthermore, each node might decide to perform this estimation and conclude that its own neighborhood is relatively static with respect to changes in the topology, and that certain values of k_r work even in the presence of mobility.

Since the algorithm is parametrized with respect to the properties of the network, we can select the appropriate version based on the desired goal. There is, therefore, a tradeoff between latency and the number of messages required by the algorithm that can be used by our framework to adapt to the requirements of the application or the network itself.

Finally, although the experiments presented in this paper only deal with two distinct roles, our results are clearly valid for any number of roles.

4. Related Work

`TinyCubus` and our role-based code distribution algorithm are related to a variety of other work. In this section, we provide a description of relevant projects that are in the process of creating frameworks similar (in part) to ours, related code distribution schemes, and finally, routing algorithms that, like ours, use cross-layer data to make forwarding decisions.

SensorWare [8] and Impala [9] aim at providing functionality to distribute new applications in sensor networks. For this purpose, they create abstractions between the operating system and the application, although both differ slightly from each other. SensorWare uses a scripting language that is not really well-suited for resource-limited platforms. It uses special commands of the language that allow the forwarding of the current program to other nodes, and tries to avoid unnecessary code transfers by transmitting the code only if the script is not already running on the neighboring nodes. SensorWare does not support adaptation and cross-layer interactions, as it is the case in our framework.

In Impala, new code is only transmitted on demand if there is a new version available on a neighboring node. Furthermore, if certain parameters change and an adaptation rule is satisfied, the system can switch to another protocol. However, this adaptation mechanism only supports simple adaptation rules. Although it uses cross-layer data, Impala does not have a generic, structured mechanism to share it and so, is not easily extensible.

The MobileMan project [10] is a system that aims at creating a cross-layer framework similar to ours. However, MobileMan is not targeted towards sensor networks and assumes environments typical of mobile ad-hoc networks, which are, in the general case, not so limited in terms of resources. In addition, MobileMan focuses on data sharing between layers of the network protocol stack and, therefore, does not include

the configuration and adaptation capabilities found in our framework.

EmStar [11] is a software environment for Linux-based sensor nodes that, like MobileMan, assumes the presence of higher-end nodes as part of the sensor network. EmStar also contains some standard components for routing, time synchronization, etc., but it is not able to provide the adaptation mechanisms available in our framework.

Regarding related work concerned with the implementation of code distribution, Ripple [12] is a code distribution algorithm implemented using EmStar. In order to reduce the number of messages, this algorithm uses a publish/subscribe scheme where a single node in the neighborhood sends code updates to its subscribers. Similar to our approach, it includes a mechanism to transmit code updates reliably, but it fails to consider cross-layer data (e.g., role information) and, therefore, data is always forwarded to all nodes.

Another example of code propagation for sensor networks is Trickle [13]. Trickle periodically broadcasts meta-data about the software version nodes are using, and focuses on detecting whether or not a code update is needed. On the other hand, our role-based algorithm is used to selectively send code updates to nodes that are supposed to receive it based on their role assignmnent. Of course, it would be possible to combine both algorithms to further optimize code updates in our system.

Finally, there are a number of routing algorithms [14], [15] that use cross-layer information to improve on their efficiency, although this is usually done on a protocol-specific basis. One example is the use of spatial information for routing, as has been done in the Cartalk 2000 project [16]. However, Cartalk does not provide a generic mechanism to allow for arbitrary cross-layer data sharing that can be used with other schemes.

5. 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 breadth of typical sensor 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, that manages the upload of code onto the appropriate sensor nodes.

Furthermore, we have provided the description of a novel role-based code distribution algorithm that uses cross-layer information, such as role assignments, in order to improve on the number of messages needed to distribute code to specific nodes. The results of our evaluation show that this algorithm performs several times better than plain flooding in scenarios where the topology and distribution of roles within the network is well-known.

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 work to do. We are in the process of integrating our framework with an additional application that provides the capabilities found in a smart environment and that will fully make use of the functionality provided by `TinyCubus`.

Finally, regarding the role-based code distribution algorithm, we plan on extending it to support highly mobile sensor nodes, like the ones found in the Cartalk 2000 project, and to include functionality found in related projects, like Trickle. In addition, we would like to analyze other types of topologies where nodes are randomly placed and not equally spaced and investigate how well our role-based algorithm works under such conditions.

References

[1] 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*, 2000, pp. 93–104.

[2] 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 the ACM SIGPLAN 2003 Conf. on Programming Language Design and Implementation*, 2003, pp. 1–11.

[3] A. J. Goldsmith and S. B. Wicker, "Design challenges for energy-constrained ad hoc wireless networks," *IEEE Wireless Communications*, vol. 9, no. 4, pp. 8–27, 2002.

[4] K. Römer, C. Frank, P. J. Marrón, and C. Becker, "Generic role assignment for wireless sensor networks," in *ACM SIGOPS European Workshop*, 2004, to appear.

[5] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," *Wireless Networks*, vol. 8, no. 2/3, pp. 153–167, 2002.

[6] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proc. of the 1st Intl. Conf. on Embedded Networked Sensor Systems*, 2003, pp. 126–137.

[7] Sustainable bridges web site. [Online]. Available: http://www.sustainablebridges.net

[8] A. Boulis, C.-C. Han, and M. B. Srivastava, "Design and implementation of a framework for efficient and programmable sensor networks," in *Proc. of the 1st Intl. Conf. on Mobile Systems, Applications, and Services (MobiSys 2003)*, 2003.

[9] 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, pp. 107–118.

[10] M. Conti, G. Maselli, G. Turi, and S. Giodano, "Cross-layering in mobile ad hoc network design," *IEEE Computer*, vol. 37, no. 2, pp. 48–51, 2004.

[11] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin, "EmStar: A software environment for developing and deploying wireless sensor networks," in *Proc. of USENIX 2004*, 2004, pp. 283–296.

[12] T. Stathopoulos, J. Heidemann, and D. Estrin, "A remote code update mechanism for wireless sensor networks," University of California, L.A., Tech. Rep. CENS-TR-30, November 2003.

[13] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proc. of the 1st USENIX/ACM Symp. on Networked Systems Design and Implementation*, 2004.

[14] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proc. of the Hawaii Intl. Conf. on System Sciences*, vol. 2, 2000, p. 10 ff.

[15] W. H. Yuen, H. no Lee, and T. D. Andersen, "A simple and effective cross layer networking system for mobile ad hoc networks," in *Proc. of the 13th IEEE Intl. Symp. on Personal, Indoor and Mobile Radio Communications*, vol. 4, 2002, pp. 1952–1956.

[16] J. Tian, L. Han, K. Rothermel, and C. Cseh, "Spatially aware packet routing for mobile ad hoc inter-vehicle radio networks," in *Proc. of the IEEE 6th Intl. Conf. on Intelligent Transportation Systems (ITSC)*, vol. 2, 2003, pp. 1546–1551.