Structuring Sensor Networks with Scopes

Daniel Jacobi*, Pablo E. Guerrero, Ilia Petrov, and Alejandro Buchmann

Dept. of Computer Science, Technische Universität Darmstadt
D-64289 Darmstadt, Germany
{jacobi, guerrero, petrov, buchmann}@dvs.tu-darmstadt.de

Abstract. Historically, wireless sensor network architectures assume that all nodes participate in a single global task. However, it is logical to think that sensor networks will be exploited by multiple, concurrent applications. In this demonstration we present the concept of Scopes, a powerful programming abstraction that allows to dynamically partition the set of nodes into groups according to certain criteria and to address a group as a whole to perform operations on them.

1 Introduction

Wireless Sensor Networks (WSNs) consist of hundreds of low-power nodes that form dynamic ad-hoc multi-hop networks. A common pattern observed across various WSN deployments is the need to partition the set of nodes into groups. This can be necessary to manage the network, delimit ownership boundaries, establish nodes' visibility, or execute specific sensing operations, among other reasons. Once these groups have been established, furthermore, it is natural to require traffic to and from the group members. In this paper we demonstrate an approach called Scopes [1] that efficiently tackles these two challenges.

A *scope* is defined as a group of nodes that match a given *membership condition*. As an example, we can *declaratively* express a scope as:

```
CREATE SCOPE scopeA ( COMPANY = 'XYZ' AND
          EXISTS SENSOR 'TEMPERATURE' AND TEMPERATURE < 20C );
```

In this case, `COMPANY` refers to a property stored on the sensor node, while `EXISTS SENSOR` refers to a capability a sensor node may have, i.e., whether it has a temperature sensor. These declarations are flattened into a pre-order byte array, thus achieving a *compact representation*. Once deployed, a scope can be used to exchange messages to and from the whole partition of nodes.

The scope membership condition is a logical expression that must be satisfied by a node to belong to the scope. The condition contains node capabilities and properties. A capability indicates the presence of a sensor or actuator. A property refers to a data value stored at a node, which in turn may be static or dynamic. A static property refers to an explicit data item such as `COMPANY = 'XYZ'`, while a dynamic property is calculated based on the present state of the node, for

* Supported by the DFG Graduiertenkolleg 1362, *Cooperative, Adaptive and Responsive Monitoring in Mixed Mode Environments.*

example `TEMPERATURE < 20C`. Dynamic properties are powerful but expensive, since every change results in a re-evaluation of the membership.

Scopes can be nested, forming a hierarchy. Each scope has a superscope; none-nested scopes have an artificial superscope called the *world*, which includes all available nodes. Conceptually, nesting scopes allows for clearer definitions and better organization. Technically, they reduce the communication overhead thus improving the energy efficiency.

In order to create a scope, a node is chosen as *scope root*. Although any node can inject a scope, in practice we observe two cases. First, a scope can be injected from a gateway (sink) node, in which case the scope has typically a global and permanent character. Second, inner nodes can inject a scope, which in most cases refer to the node's local neighborhood for tasks such as data aggregation. Once a scope is created, it is automatically maintained by the Scopes infrastructure. Scopes continue to exist even as nodes fail, leave or (re)join the network. Eventually, a scope can be removed from the network.

2 System Design

The present implementation of Scopes is built on top of the SOS operating system [2]. SOS offers performance and memory footprint similar to those of TinyOS. SOS, however, provides the capability of loading and unloading modules at run-time, a feature TinyOS does not support without special additions. The Scopes framework splits the system into three layers (see Figure 1). At the top is the *application layer*, where all application modules using Scopes reside.

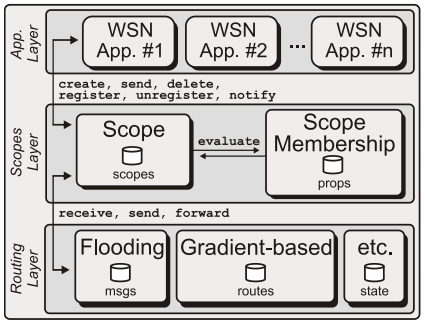

Fig. 1. Scopes Architecture.

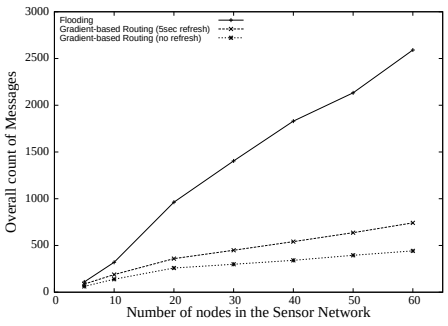

Fig. 2. Total no. msgs. Exchanged.

The *scopes layer* performs all the management and provides API's for the lower and upper layer. The Scope module maintains scope information and interfaces to application and routing modules. The interface on the application side provides methods to create, delete and send data via scopes and to register for notifications about the dynamic scope membership state changes. For routing modules, an interface provides a primitive to send data and also to signal scope

membership updates. With each scope, a *refresh* interval can be specified. The Scope Membership module evaluates on every node the membership condition. If a dynamic property changes, a scope reevaluation is triggered locally.

Right on top of SOS, the *routing layer* is placed. Scopes handles routing algorithms independently from the scope definition. We have implemented two standard routing algorithms, namely *flooding* and an extended *gradient-based* routing based on Directed Diffusion [3]. Our version adds the possibility of having bi-directional messaging. Figure 2 shows preliminary performance results.

3 Overview of the Demonstration

In the present demonstration we consider a harbor scenario, where several logistics companies must manage containers with goods of different types. Every container is equipped with nodes connected to sensors monitoring varying conditions, thus forming an inter-container ad-hoc network.

Technical setup. We use Tmote Sky nodes equipped with light, temperature and humidity sensors. Each node is pre-loaded with the SOS operating system and the Scopes framework. In order to simulate the harbor scenario, two static properties are set on every node: the company name and the container location.

Scenario 1. Creation of several scopes and nested scopes with different membership conditions, resorting to a container's company or position properties. This is to show the declarative character of scopes as well as the convenience of defining and working with them at the application level.

Scenario 2. Shows root-to-scope traffic by sending a command to a scope members, which triggers a data processing task. This occurs when a ship reaches its destination, where statistics for the entire trip are calculated locally.

Scenario 3. Shows scope-to-root traffic by sending back irregularities detected in Scenario 2. The resulting statistics and detected deviations are immediately reported to the harbor authorities and taken into account while unloading and placing containers on the dock freight slots.

Scenario 4. Injection of scopes from arbitrary nodes in the network. Provided a sharp temperature increase is detected by a container, a scope is created over neighboring containers located within a range of 10 meters, to which a message is sent to evaluate and report irregularities.

References

1. Steffan, J., Fiege, L., Cilia, M., Buchmann, A.P.: Scoping in Wireless Sensor Networks. In: 2nd MPAC Workshop, ACM Press (October 2004) 167–171
2. Han, C.C., Rengaswamy, R.K., Shea, R., Kohler, E., Srivastava, M.B.: A Dynamic Operating System for Sensor Networks. In: 3rd Intl. Conf. on Mobile Systems, Applications, and Services, New York, NY, USA (June 2005) 163–176
3. Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., Silva, F.: Directed Diffusion for Wireless Sensor Networking. IEEE Transactions on Networking **11**(1) (February 2003) 2–16