Using Logical Neighborhoods to Enable Scoping in Wireless Sensor Networks

Luca Mottola and Gian Pietro Picco (Advisor)
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
{mottola,picco}@elet.polimi.it

ABSTRACT

Wireless Sensor Networks (WSNs) are now enabling applications whose objective is not just to monitor the environment, but also to perform actions on it so as to implement complex control loops. Unlike early WSN projects where the application tasks were mainly relegated to the fringes of the network, e.g., to a powerful base station, in sensing and acting scenarios the application intelligence is brought in the network, and distributed among the nodes [1]. These applications are often composed of many collaborating sub-tasks, each involving only a subset of the nodes in the system. Therefore, the programmers must worry about how to identify these subsets and address them, before concentrating on the application goals. This results in additional programming effort and more complex code, affecting the reliability of the resulting application.

In this work, we propose a programming abstraction called *Logical Neighborhood*, whose goal is to raise the level of abstraction from the physical neighborhood of a node to a logical notion of proximity. The programmers can specify the nodes part of a logical neighborhood using a declarative language we devised, based on application-defined attributes of the nodes. To address the members of a logical neighborhood, our framework provides a general communication API, supported by a dedicated routing scheme. Here, we present the logical neighborhood abstraction, illustrate our dedicated routing solution briefly reporting on some performance results, and point at current and future investigations based on the logical neighborhood abstraction.

Categories and Subject Descriptors: C.2.2 [Network Protocols]: Routing protocols; D.2.11 [Software Architectures]: Languages

1. INTRODUCTION

Wireless sensor networks (WSNs) are increasingly employed in a variety of settings to gather data from the physical world. Habitat monitoring [15], one of the most popular applications, is paradigmatic in this respect. In that case, the system architecture features a *single* base station collecting data from a high number of *homogeneous* nodes. Conversely, researchers are now investigating the use of WSNs to implement decentralized control loops that rely on data *sensed* to decide on *actions* to be performed. In these settings, the applications often rely on *localized interactions* [6], therefore, each node may act as a base stations for a small portion of nearby devices. Moreover, *heterogeneous* nodes are deployed to provide various sensing and acting capabilities [1]. Applications range from localization facilities to control systems in tunnels or buildings, interactive museums and home automation [19].

New challenges are brought by these application scenarios. Indeed, while in early WSN deployments the application goals were mainly realized by a single task performed across the whole network (typically, sensing and reporting environmental data), sensing and acting applications are usually composed of many collaborating tasks, each affecting only a given part of the system. For instance, a system deployed to perform control and monitoring in a building needs to perform at least three main tasks, i.e., structural monitoring, in-door environment monitoring, and response to extreme events such as fire or earthquakes [5]. To realize the latter functionality, actuator nodes controlling water sprinklers need to monitor nearby temperature sensors and take the appropriate countermeasures where and when needed. In these settings, the developer must worry not only about the implementation of the application logic, but also about which subset of the system should be involved and how to reach it. As no dedicated programming constructs and mechanisms exist for the latter task, the result is additional programming effort, increased complexity and, in absence of well-established and reusable solutions, less reliable code.

Our research addressed these issues by introducing the notion of *logical neighborhood* [16, 17], illustrated in Section 2. This provides an abstraction that replaces the conventional notion of physical neighborhood—i.e., the set of nodes within the communication range of a given device—with a logical notion of proximity determined by applicative information. Logical neighborhoods are specified declaratively using the SPIDEY language we designed, conceived to be a simple extension to existing WSN programming languages (e.g., nesC [8] in the case of TinyOS [10]). The programmers can address the members of a logical neighborhood by using a simple message passing API, which replaces broadcast to the physical neighbors. This enables a form of logical broadcast where the receivers are the nodes satisfying the neighborhood specification, instead of the nodes within communication range.

The aforementioned communication API is supported by a novel routing mechanism, described in Section 3. This is expressly devised in support of logical neighborhoods, and takes into account node heterogeneity explicitly. Our performance evaluation shows that it efficiently supports logical neighborhoods, therefore demonstrating the feasibility of our approach. Our abstraction and API can also foster a fresh look at existing programming models by replacing the conventional physical broadcast with our logical notion of proximity. At the same time, logical neighborhoods can enable

MDS06, November 27-December 1, 2006 Melbourne, Australia

```
node template Sensor
  static Function
  static Type
  dynamic BatteryPower
  dynamic Reading

create node ts from Sensor
  Function as "sensor"
  Type as "temperature"
  Reading as getTempReading()
  BatteryPower as getBatteryPower()
```

Figure 1: SPIDEY: sample node definition and instantiation.

```
neighborhood template HighTempSensors(threshold)
  with Function = "sensor" and
       Type = "temperature" and
       Reading > threshold

create neighborhood htsn100
  from HighTempSensors(threshold: 100)
  max hops 2
  credits 30
```

Figure 2: SPIDEY: sample neighborhood definition and instantiation, `threshold` is a parameter bound at instantiation time.

novel programming abstractions based on the scoping mechanisms they provide. These opportunities are illustrated in Section 4.

The actual usefulness of the logical neighborhood abstraction will be ultimately dictated by its effective use in real-life WSN applications. To this end, we intend to evaluate the advantages brought by our abstraction by developing an extensive set of relevant applications on top of logical neighborhoods. For this task to be effective, we also need to support our routing mechanism with an analytical model, so as to give the developers the ability to fine-tune our framework depending on their needs. Section 5 discusses these future research goals.

Finally, in Section 6 we highlight how the SPIDEY language used to define logical neighborhood is more expressive than existing frameworks, and how our abstraction is inherently more flexible and general than existing proposals in the field.

2. ABSTRACTION

The logical neighborhood abstraction revolves around only two concepts: *nodes* and *neighborhoods*, both specified using the SPIDEY language [17]. Nodes represent the portion of a real node's state and characteristics made available to the definition of any logical neighborhood. The definition of such a (logical) node is encoded in a *node template*, which specifies a node's exported attributes. This template is then used to derive actual instances of (logical) nodes, by specifying the actual source of data. Figure 1 reports a fragment of SPIDEY code that defines a template for a generic sensor. The attributes in a node template can be **`static`** or **`dynamic`**. The former represent information assumed not to vary in time, e.g., the type of measurement a sensor node provides. Instead, dynamic attributes represent information that by definition changes with time, e.g., the current sensor reading. In Figure 1, the template is then instantiated by binding attributes to constant values or functions of the target language, obtaining one (logical) node.

A logical neighborhood can be defined based on arbitrary predicates on node templates. As already illustrated for nodes, a neighborhood is first defined in a template, which basically encodes the corresponding membership function, and then instantiated by specifying *where* and *how* the neighborhood is to be constructed and maintained. For instance, Figure 2 illustrates the definition of a neighborhood template involving temperature sensors whose reading is above a given threshold[1]. The template is then instantiated so that it evaluates the corresponding predicates only on nodes that are at most of 2 hops away from the node defining the neighborhood, and by spending a maximum of 30 "credits". Figure 3 shows a pictorial representation of the example.

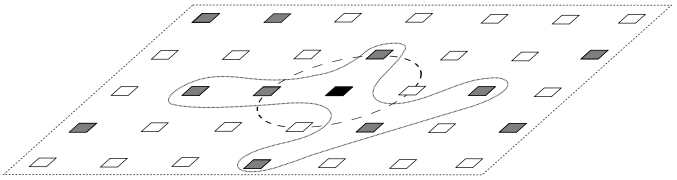


Figure 3: A pictorial representation of the example in Figure 2. The black node specifies the *logical* neighborhood, and its *physical* neighborhood is denoted by the dashed circle. The dark nodes satisfy the neighborhood template `HighTempSensors` when the threshold is set to 100^{o}C. However, the nodes included in the neighborhood instance `htsn100` are only those lying within 2 hops from the black node, as specified with the **`hops`** clause.

In particular, the **`credits`** construct is an application-defined measure of *communication cost*, explicitly defined by supplying at each node a *sending cost function* through a particular SPIDEY construct. This describes the cost a node incurs in sending a broadcast message to the physical neighbors, thus naturally taking into account the heterogeneity of the nodes in the system. For instance, one can define higher cost for battery-powered sensors and lower costs for resource-rich nodes. The "credits" attached to a logical neighborhood are then evaluated as the sum of the sending costs each node involved in routing messages for that neighborhood incurs in. Therefore, the construct exposes the trade-off between accuracy and resource consumption up to the application.

Notice how **`credits`** and **`hops`** represent different information. The former constrain the span of a neighborhood depending on the amount of resources the developers is willing to spend to reach the neighborhood members. Hence, neighborhood instantiated with a high number of credits have a broader coverage of the system, at the price of higher resource consumption. Conversely, the **`hops`** construct limits a logical neighborhood depending on the shape of the network topology, regardless of the resources consumed. Combining the two provides even greater flexibility.

More advanced features of the SPIDEY language allow, in a given neighborhood template, for expressions composed of the usual boolean operators **`and`**, **`or`** and **`not`**. Moreover, different neighborhood templates can be combined using usual set operations such as **`intersection`**, **`union`**, and **`minus`**, and can also be defined as a subset of another, already defined, neighborhood template.

Communication in a logical neighborhood is made available to the programmer by redefining the usual broadcast facility. In particular, we change the signature of the send operation to be

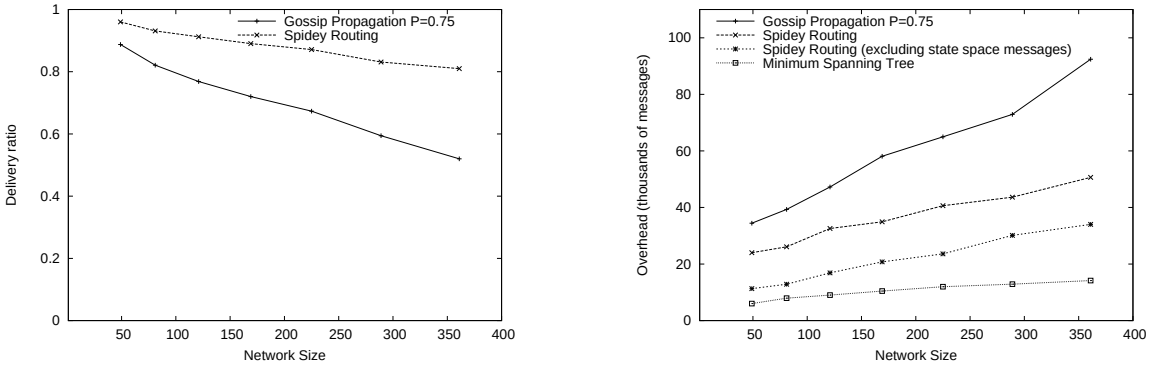
```
send(Message m, Neighborhood n)
```

thus making it dependent on the (logical) neighborhood to which the message is addressed. To implement communication in a logical neighborhood, we need a routing mechanism able to deliver messages to neighborhood members efficiently. In the next section we illustrate how we addressed this issue.

3. ROUTING

The logical neighborhood abstraction is essentially independent of the underlying routing layer. Nevertheless, its characteristics

[1]In case a neighborhood template defines a predicate over an attribute not defined in a node template, the whole neighborhood template evaluates to false.



(a) Message delivery against network size.

(b) Network overhead against network size.

Figure 4: Evaluation against gossip and ideal multicast along the minimum spanning tree.

cannot be easily accommodated by existing routing approaches in WSNs. Indeed, these usually focus on how to collect efficiently data from many sensors to a single node. In our approach the perspective is reversed: we must efficiently transmit an application message from a single node to those matching the neighborhood predicate. Moreover, logical neighborhoods are a scoping mechanism, and therefore can be used in conjunction with several mechanisms other than data collection, as we will discuss in Section 4. Finally, credit management is a distinctive feature of our approach that would anyway require appropriate integration. For these reasons, we designed a dedicated routing strategy supporting the abstraction. Due to space limitations we can only sketch its behavior here. A more detailed description is available in [16].

Our approach to routing is *structure-less* (i.e., it does not exploit overlays), is based on the notion of *local search*, and relies on two core mechanisms. The first mechanism builds a distributed *state space* by periodically propagating node profiles, i.e., the list of node attributes and their values. In doing this, each node stores the cost (in credits) to reach a device whose profile contains a specific ⟨attribute,value⟩ pair. This cost is evaluated in terms of the aforementioned node sending cost, by accumulating the corresponding amount of credits along the path to that device. Nevertheless, the propagation of node profiles is constrained so that each node has enough information to reach only the node associated to the "cheapest" path, determined by looking at the credits needed to reach it. Therefore, the spreading of node information can be limited to small portions of the system, thus scaling better.

The second mechanism enables messages to smartly "navigate" the state space. Messages addressed to a logical neighborhood contain the neighborhood template, and the corresponding credits and number of hops specified when instantiating the neighborhood. The credits are "spent" while navigating the state space. Each message is always sent along at least a *decreasing path*, i.e., a path where the cost needed to reach a ⟨attribute,value⟩ pair matching the neighborhood template is decreasing. Whenever such a path is found, a *credit reservation* mechanism, exploiting the state space information, reserves enough credits to reach the node at the end of the decreasing path. The remaining credits are used to route the message along *exploring* paths, i.e., directions where the cost to reach a matching node is non-decreasing. This is done to avoid local minima in the state space, and to find further decreasing directions towards different neighborhood members. In [16], we evaluated this routing scheme, proving how it performs well with respect to a pure gossip mechanism. An excerpt of our simulation results is presented next.

Routing Performance. We implemented our solution in TinyOS [10] and evaluated its performance using the TOSSIM [12] simulator. Our deployment scenario is a grid where each node can communicate with its four neighbors[2]. This choice simplifies the interpretation of the results by removing the bias induced by unstructured scenarios, and also models well some of the settings we target, e.g., indoor WSN deployments [20]. Each simulation run lasted 1000 s—a value above which we verified all the measures exhibit a variance less than 1%.

A single logical neighborhood was defined with a sender generating one message per second, and having 10% of the nodes in the system as neighborhood members. The node sending cost was constant and identical throughout the system. Credits were assigned by slightly overestimating the cost to reach a node in the system along the shortest path, and weighing this value by the probability of the node being a receiver. As already mentioned, the definition of a model supporting fine-tuning of credit assignment is our immediate research goal, and will be discussed in Section 5.

Figure 4 illustrates some of our simulation results, obtained by comparing our solution against a gossip scheme with the forwarding probability set to 0.75. Figure 4(a) shows that the percentage of delivered messages is consistently higher than in gossip, and is significantly less sensitive to an increase of the network size. As for network overhead, (i.e., the number of messages exchanged at the MAC layer), Figure 4(b) evidences how we generate almost half of the overhead of gossip, and yet deliver significantly more messages. In this respect, we provide additional insights by showing our protocol performance with and without the messages needed to build the state space, and by comparing against the ideal lower bound provided by the minimum spanning tree rooted at the sender (computed with a global knowledge of network topology). The gap between gossip and our solution is even more evident not counting the cost to build the state space. This highlights how efficient is the pure routing mechanism, once the routing information is in place. This is particularly significant, as the construction of the state space is a fixed cost that is paid once and for all. In other words, adding another sender—regardless of the neighborhood it addresses—does *not* increment the overhead due to state space generation.

4. BUILDING OVER LOGICAL NEIGHBORHOODS

Logical neighborhoods are independent of what is running on top. Our current research is exploring how to couple logical neigh-

[2]We used the TinyOS' `LossyBuilder` to generate topology files with transmission error probabilities taken from real testbeds.

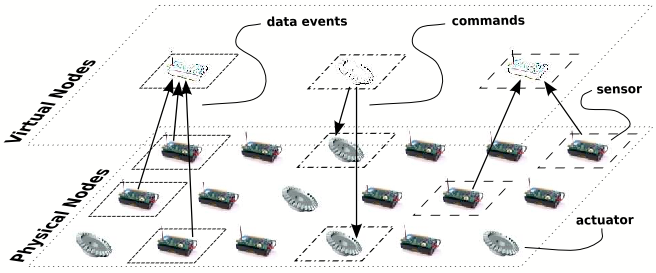


Figure 5: The flow of information between real devices and virtual sensors or actuators. Nodes belonging to the same logical neighborhood are identified with the same dashed lines.

```
create node vts from Sensor
  Function as "virtualSensor",
  Type as "temperature",
  Reading as average(roomTempSensors) every 30

float average(Neighborhood: nhood) {
  sum = 0; counter = 0;
  for(node in nhood) {sum += node.Reading; counter++;}
  return sum/counter;
}
```

Figure 6: Sample definition of a virtual sensor reporting the average of the temperature readings in a neighborhood. `roomTempSensor` is the identifier of the previously instantiated neighborhood. In `average()`, we use an abstract construct `in`, that our SPIDEY translator maps to constructs of the target language.

```
node template Actuator
  static attribute Function
  static attribute Type
  operation Activate(int regulation)
  operation Deactivate()

create node ws from Actuator
  Function as "actuator"
  Type as "waterSprinkler"
  Deactivate() as regulateSprinklerFlow(0)
  Activate(int regulation)
              as regulateSprinklerFlow(regulation)

neighborhood template RoomSprinklers()
  with Function = "actuator" and
       Type = "waterSprinkler" and
       provides(Activate(int regulation) and
       provides(Deactivate())
```

Figure 7: A neighborhood containing water sprinklers that can be activated or deactivated.

borhoods with higher-level programming models and applications. In this section, we report on three such efforts.

Virtual Nodes. In the sensing and acting scenarios we introduced in Section 1, logical neighborhoods provide the ability to define subset of nodes and interact with them. However, the data collection required to implement the sensing tasks, and the distribution of commands to actuator nodes to affect the environment are still on the programmers' shoulders. This issue can be faced by enabling *virtual nodes*, acting either as *virtual sensors* or *virtual actuators* [3]. The former enables data sensed at nodes part of a neighborhood to be processed according to an application-provided aggregation function, and then perceived as the readings of a single, virtual device. Conversely, the latter allow to operate a potentially sparse set of actuator nodes from a single entry point, thus simplifying the application code. These concepts are graphically illustrated in Figure 5.

To define virtual sensors, one simply instantiates a logical node binding one or more attributes to an aggregation function taking a neighborhood instance as parameter, as in Figure 6. When this happens, the SPIDEY translator generates a programming entity (e.g., a nesC component and corresponding interface in the case of TinyOS), that can be used to access the state of the virtual sensor. Conversely, Figure 7 illustrates how virtual actuators are enabled by defining logical nodes not only in terms of exported attributes, but also by specifying the operations they offer. A further **`provides`** construct is made available for the definition of logical neighborhoods. This can be used to predicate on exported operations, as exemplified in Figure 7. Similarly to virtual sensors, our SPIDEY translator generates a programming entity used to control the set of actuator nodes belonging to the neighborhood. Once created, virtual sensors and actuators can be addressed like any other node and hence be part of further logical neighborhoods. This allows one to recur the process arbitrarily, creating hierarchies of virtual nodes.

Virtual actuators are readily implemented on top of the routing facility we described in Section 3: only a suitable encoding of operations and their parameters (if any) is needed. Conversely, several optimizations may be possible to implement virtual sensors, based either on the fact that multiple virtual sensors are likely to insist on the same neighborhood, or depending on the mathematical characteristics of the aggregation function at hand. Currently, we are investigating dedicated routing mechanisms to implement virtual sensor efficiently. Early results appear in [3].

Context-aware Tuple Spaces. Tuple spaces are a programming abstraction offering communication and coordination decoupled in time and space [9]. As such, they are particularly amenable to dynamic environments characterized by changing topologies. In [4], a tuple space is used to share data sensed by WSN devices among mobile units. The scenario is that of an *immersive* WSN: a field of possibly heterogeneous devices is spread in a geographical region, and mobile users roam around communicating with these devices through 1-hop, physical broadcast. The tuple space does not span the WSN devices. Instead, these are seen as simple data sources, and used to infer the *context* surrounding the mobile user.

However, 1-hop communication is still a limitation. Logical neighborhoods can be used to give the users the ability to to define the shape of the "cloud" surrounding them, i.e., the set of devices their context is derived from, and do that regardless of the physical distance from the mobile unit. The concept is illustrated in Figure 8. For instance, a user might be willing to adapt the behavior of the application running on her mobile device depending on temperature readings within some geographical scope, whereas a different user might be willing to modify her application's user interface based on the actuators nearby. Our framework is able to provide this despite user mobility, as the routing scheme we described in

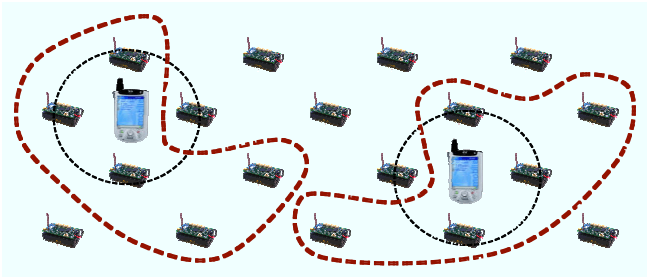


Figure 8: Tuple spaces-based data sharing using logical neighborhoods: the 1-hop physical broadcast determining the context of mobile users is replaced with arbitrarily shaped logical neighborhoods.

```
node template Sensor
  static Function
  static Type
  dynamic RoutingComponent
  dynamic RoutingComponentVersion

create node sn from GenericDevice
  Function as "sensor"
  Type as "temperature"
  RoutingComponent as getRoutingName();
  RoutingComponentVersion as getRoutingVersion();
```

Figure 9: SPIDEY: defining attributes to export information on the node software configuration.

```
neighborhood template ObsoleteRouting(name, version)
  with Function = "sensor" and
       Type = "temperature" and
       RoutingComponent = name and
       RoutingComponentVersion < version

create neighborhood updateDD
  from ObsoleteRouting("DirectedDiffusion",3.0)
```

Figure 10: SPIDEY: definition of a neighborhood to update the DirectedDiffusion routing component where the version is less than 3.0.

Section 3 works regardless of a possible moving sender. Indeed, if the node defining the neighborhood is the only moving device, it can just broadcast the message, and let the remaining (fixed) nodes route the message according to the state space information, which are instead stored on stationary nodes only.

Software Updates. Software update in WSNs is known to be a challenging task, and existing proposals (e.g., [13]) mostly address homogeneous scenarios where a software update must be directed to all nodes in the system. In heterogeneous scenarios, instead, the problem of software update becomes more difficult: the nodes run different software configurations based on their hardware characteristics and the tasks they are supposed to accomplish.

Logical neighborhoods can be exploited to address this issue. Simply, one could define node templates that export attributes describing the software components running on a node, as in Figure 9. This way, logical neighborhoods can be defined by predicating on the software configuration of the system. For instance, in Figure 10 we define a neighborhood including only the nodes in the system currently running a specific routing component whose version is less than a given value. At that point, directing a piece of code to all the nodes actually needing a particular software update is as simple as sending a (logical) broadcast message addressed to that neighborhood. Note how using logical neighborhoods for software updates naturally solves both the problem of addressing the subset of nodes needing the update itself, and the problem of distributing the code across multiple hops efficiently.

5. RESEARCH AGENDA

Our research agenda comprises a natural extension of the research activity described in Section 4, as well as further investigations in closely related directions. We here sketch the main topics we intend to investigate.

Augmenting the Language. A need we recognized while working with logical neighborhoods is the ability to condition the belonging of a node A to a given neighborhood based on the belonging of a different node B to the same neighborhood. For instance, this might be required to define neighborhoods with peculiar topological properties such as Yao graphs [14]. The current version of SPIDEY cannot capture this requirement, as the neighborhood template is evaluated independently on each node. Therefore, we need to enrich the SPIDEY language, so as to enable more complex node selection mechanism, and modify the underlying routing scheme accordingly. For this reason, we plan to evaluate the trade-offs involved at the network level in simulation studies, so as to have a quick way of evaluating different design choices.

Evaluating the Abstraction. We want to verify the claim that applications written on top of logical neighborhoods result in cleaner and more simple code, and thus yield more reliable systems. To evaluate this, we need to study the advantages brought to the programmer by our abstraction not only qualitatively, but also *quantitatively*. For instance, we need to assess the benefits provided by our abstraction in terms of lines of code, inter-component dependencies, and complexity of the resulting application. To this end, we intend to consider existing and publicly available applications, try to re-implement them on top of our abstraction, and compare the two versions. In doing this, we will need to define fair metrics to be applied on the original code and on the one we will develop.

Routing. Our credits reservation mechanism already provides some management of available resources through the use of credits. However, we intend to push further our current routing solution, exploring the possibility of having an adaptive mechanism that, based on a notion of reinforcement along a path, directs the credits available in a message towards zones of the system still unexplored. This may enable a more careful exploration of the state space, thus obtaining savings in network overhead. To this end, we plan to purse extensive simulations studies in various network scenarios.

Our routing scheme still lacks an extensive evaluation in real deployments scenarios. To this end, the users of our framework need supporting tools to customize its behavior, and explore the trade-off between system coverage and resource consumption. In particular, they need a way to evaluate the credits to be assigned to a logical neighborhood, based on the desired quality of service. An analytical model of our routing solution can be used for this purpose. In developing this, we intend to borrow from the large body of literature available in the field of network theory and random graphs [2].

6. RELATED WORK

The work closer to our is the neighborhood abstraction described in [22], where each node has access to a local cache of attributes provided by (physical) neighbors. Data collection is built into the constructs, therefore, communication flows only according to a many-to-one paradigm. The implementation considers only 1-hop neighbors and is based on broadcasting all attributes and filtering on the receiver side. Our framework is more flexible as it provides a general application-defined neighborhood abstraction, decoupled from the application functionality. Therefore, it can be used for purposes other than data collection, as described in Section 4.

The work on Abstract Regions [21], instead, proposes a model where ⟨*key,value*⟩ pairs are shared among nodes in a *region*. The span of a region is based mainly on physical characteristics of the network (e.g., physical or hop-count distance between sensors), and its definition requires a dedicated implementation. Therefore, each region is somehow separated from others, and regions cannot be combined. This results in a lower degree of orthogonality with respect to our approach. The concept of *tuning interface* provides access to a region's implementation, enabling the tweaking of low-level parameters (e.g., the number of retransmissions). Instead, our approach provides a higher-level, user-defined notion of cost that can be used to control resource consumption.

SpatialViews [18] is a programming language for mobile ad-hoc networks where *virtual networks* are defined depending on the node physical location and provided services. Computation is distributed

across nodes in a virtual network by migrating code from node to node. Common to our work is the notion of scoping virtual networks provides. However, SpatialViews targets devices much more capable than ours, and focuses on migrating computation instead of supporting an enhanced communication facility as we do.

Finally, in [7], the authors propose a language and algorithms supporting generic role assignment in WSNs. Their approach is, in a sense, dual to ours. Their approach *impose* certain characteristics (roles) on nodes in the system so that some specified requirements are met, while in our approach the notion of logical neighborhood *selects* nodes in the system based on their characteristics.

In our routing scheme, we were influenced by Directed Diffusion [11] in setting up routes towards potential neighborhood members. However, an important difference is that our profile advertisements do not propagate to the whole network, unlike interests in Directed Diffusion. Moreover, we do not assume data collection as the main functionality, and therefore we cannot rely on any knowledge about message content, required in Directed Diffusion for interpolation along failing paths. Finally, routing in Directed Diffusion is entirely determined by gradients, while we make the system more resilient to changes by allowing exploratory steps, whose use is nevertheless under the control of the programmer through the credit mechanism.

7. CONCLUSIONS

In this paper, we illustrated our current and future research on logical neighborhoods, a programming abstraction enabling a notion of scoping in WSNs. Our communication API, supported by an efficient routing scheme, extends communication in the physical neighborhood to a logical level, where the programmers specify the neighbors of a node using the SPIDEY language we devised.

We pointed out the topics of our current research, geared towards coupling applications, existing communication models, and novel programming abstractions with logical neighborhoods. Moreover, we intend to support the users of our abstraction with suitable tools to explore the trade-offs involved, and perform extensive evaluation of our solutions in simulation as well as in real deployments.

8. REFERENCES

[1] I. F. Akyildiz and I. H. Kasimoglu. Wireless sensor and actor networks: Research challenges. *Ad Hoc Networks Journal*, 2(4):351–367, 2004.

[2] B. Bollobas. *Random Graphs*. Academic Press, Harcourt Brace Jovanovich, 1985.

[3] P. Ciciriello, L. Mottola, and G. P. Picco. Building Virtual Sensors and Actuator over Logical Neighborhoods. In *Proc. of the 1^{st} ACM Int. Wkshp. on Middleware for Sensor Networks (MidSens - colocated with ACM/USENIX Middleware)*, 2006.

[4] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A.L. Murphy, and G.P. Picco. TINYLIME: Bridging Mobile and Sensor Networks through Middleware. In *Proc. of the 3^{rd} IEEE Int. Conf. on Pervasive Computing and Communications (PerCom 2005)*, pages 61–72, 2005.

[5] M. Dermibas. Wireless sensor networks for monitoring of large public buildings, 2005. Tech. Report, University of Buffalo. Available at `www.cse.buffalo.edu/tech-reports/2005-26.pdf`.

[6] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: scalable coordination in sensor networks. In *Proc. of the 5^{th} Int. Conf. on Mobile computing and networking (MobiCom)*, 1999.

[7] C. Frank and K. Römer. Algorithms for generic role assignment in wireless sensor networks. In *Proc. of the 3^{rd} ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2005.

[8] 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 Conf. on Programming Language Design and Implementation (PLDI'03)*, pages 1–11, 2003.

[9] D. Gelernter. Generative communication in Linda. *ACM Computing Surveys*, 7(1):80–112, 1985.

[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *ASPLOS-IX: Proc. of the 9^{nt} Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[11] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE Trans. Networking*, 11(1):2–16, 2003.

[12] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In *Proc. of the 5^{th} Symp. on Operating Systems Design and Implementation (OSDI)*, pages 131–146, 2002.

[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 1^{st} Symp. on Network Systems Design and Implementation (NSDI04)*, 2004.

[14] X. Y. Li, P. J. Wang, Y. Wang, and O. Frieder. Sparse power efficient topology for wireless networks. In *Proc. of the 35^{th} Annual Hawaii Int. Conf. on System Sciences*, 2002.

[15] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. of the 1^{st} ACM Int. Wkshp. on Wireless sensor networks and applications*, pages 88–97, 2002.

[16] L. Mottola and G. P. Picco. Logical Neighborhoods: A programming abstraction for wireless sensor networks. In *Proc. of the the 2^{st} Int. Conf. on Distributed Computing on Sensor Systems (DCOSS)*, 2006.

[17] L. Mottola and G. P. Picco. Programming wireless sensor networks with logical neighborhoods. In *Proc. of the the 1^{st} Int. Conf. on Integrated Internet Ad hoc and Sensor Networks (InterSense)*, 2006.

[18] Y. Ni, U. Kremer, A. Stere, and L. Iftode. Programming ad-hoc networks of mobile and resource-constrained devices. In *Proc. of the ACM SIGPLAN Conf. on Programming language design and implementation*, pages 249–260, 2005.

[19] E. Petriu, N. Georganas, D. Petriu, D. Makrakis, and V. Groza. Sensor-based information appliances. *IEEE Instrumentation and Measurement Mag.*, 3:31–35, 2000.

[20] R. Stoleru and J.A. Stankovic. Probability grid: A location estimation scheme for wireless sensor networks. In *Proc. of the 1^{st} Int. Conf. on Sensor and Ad-Hoc Communication and Networks (SECON)*, 2004.

[21] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *Proc. of 1^{st} Symp. on Networked Systems Design and Implementation (NSDI)*, 2004.

[22] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks. In *Proc. of the 2^{nd} Int. Conf. on Mobile systems, applications, and services (MobiSys)*, 2004.