Programming Wireless Sensor Networks with Logical Neighborhoods: A Road Tunnel Use Case

Luca Mottola[†] and Gian Pietro Picco[‡]
[†]Politecnico di Milano, Italy, mottola@elet.polimi.it, [‡]University of Trento, Italy, picco@dit.unitn.it

1. MOTIVATION AND SCENARIO

Wireless sensor networks (WSNs) involving *actuation* are increasingly envisioned in a range of fields [1]. Among these, there is considerable interest in leveraging off WSNs to improve safety in *road tunnels* [6]. Researchers are envisioning tunnels equipped with WSN nodes that gather physical readings such as temperature and light, monitor the structural integrity of the tunnel, and sense the presence of vehicles to detect a possible traffic congestion. Based on sensed data, the system operates a variety of devices, such as ventilation fans inside the tunnel, and traffic lights at the entrances. For instance, when a sensor detects the presence of a fire in a sector, the fans in the same sector are activated, and the traffic lights are turned red to prevent further vehicles from entering the tunnel.

To implement this class of systems, dedicated programming abstractions and communication protocols are needed. Indeed, the presence of *heterogeneous* nodes, coupled with a highly *decentralized* form of processing, make mainstream solutions (e.g., [3,4]) no longer applicable. These are usually designed with *homogeneous* nodes in mind, and focus on a *system-wide*, centralized task (e.g., data gathering at a single sink). This approach is impractical in systems involving actuation, as it may negatively impact on latency and resource consumption [1]. Instead, in our tunnel scenario the processing involves mostly *subsets* of nodes sharing similar characteristics, e.g., all the nodes controlling a fan in a specific tunnel sector. Therefore, the programmer must be provided with appropriate abstractions to "slice" the system based on the application requirements. We tackled the above problem with *Logical Neighborhoods* [8, 9], a programming abstraction that allows developers to redefine a node's neighborhood based on logical properties of the nodes, regardless of their physical position.

2. LOGICAL NEIGHBORHOODS

Logical neighborhoods are defined using a *declarative* programming language we designed, called SPIDEY. This is conceived as an extension of existing WSN programming frameworks. Programmers interact with the nodes in a logical neighborhood using an API that mimics the traditional broadcast-based, message-passing communication facility. Instead of the nodes within radio range, the message recipients are now the nodes matching a given neighborhood definition. Therefore, programmers still reason in terms of neighboring relations, but retain control over how these are established. A dedicated and yet efficient routing mechanism [8] enables communication in a logical neighborhood. Our current implementations target the Contiki [2] and TinyOS [5] operating systems.

The definition of logical neighborhoods is based on two concepts: *nodes* and *neighborhoods*. Nodes represent the portion of a real node's features made available to the definition of any logical neighborhood. Their definition is encoded in a *node template*, which specifies a node's exported attributes. This is used to derive instances of logical nodes, by specifying the actual source of data. Figure 1 reports a fragment of SPIDEY code to define a template for a generic actuator, and instantiate a logical node controlling a traffic light. To this end, the node attributes are bound to constant values or functions of the target language.

```
node template Actuator
  static Function
  static Type
  static Location
  dynamic BatteryPower
  operation Activate()
  operation Deactivate()

create node tl from Actuator
  Function as "actuator"
  Type as "traffic_light"
  Location as "entrance_east"
  BatteryPower as getBatteryPower()
  Activate() as turnLight(RED)
  Deactivate() as turnLight(GREEN)
```

Fig. 1: Node definition and instantiation.

```
neighborhood template TrafficLights(loc)
  with Function = "actuator" and
       Type = "traffic_light" and
       Location = loc

create neighborhood tl_east
  from TrafficLights(loc: "entrance_east")
  max hops 2 credits 30
```

Fig. 2: Neighborhood definition and instantiation.

A logical neighborhood is defined using predicates on node templates. Analogously to nodes, a neighborhood is first defined in a template which essentially encodes the corresponding membership function, and then instantiated by specifying *where* and *how* the template is to be evaluated. For instance, Figure 2 illustrates the definition of a neighborhood which includes the nodes controlling the traffic lights on a specific tunnel entrance. The template is instantiated so that it evaluates only on nodes that are at a maximum of 2 (physical) hops away from the node defining the neighborhood, and by spending a maximum of 30 "credits". The latter is an application-defined notion of communication costs, which exposes the trade-off between accuracy and resource consumption. The more credits are attached to a logical neighborhood, the higher is the *coverage* of the system as well as the *resources* spent to achieve that coverage. More details on the SPIDEY language are in [9].

A pictorial representation of the logical neighborhood concept is

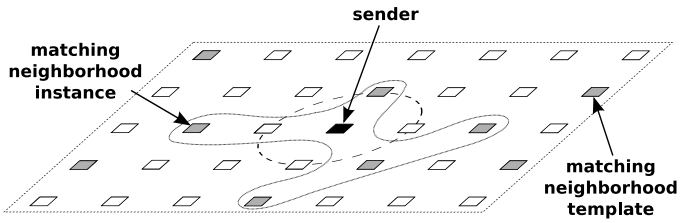

Fig. 3: A pictorial representation of a logical neighborhood.

provided in Figure 3. The black node is the one defining the logical neighborhood, and its physical neighborhood (i.e., nodes lying within its radio range) is denoted by the dashed circle. The grey nodes are those satisfying the predicate in a neighborhood *template*. However, the nodes included in the neighborhood *instance* are only those lying within 2 hops from the sending node, as specified through the **hops** clause during instantiation in Figure 2.

Sending messages to a logical neighborhood is accomplished with a modified version of the traditional broadcast communication primitive, as in `send(Neighborhood n,Message m)`. This is supported by a dedicated routing protocol, whose characteristics and performance are illustrated in [8].

3. DEMONSTRATION HIGHLIGHTS

To demonstrate a sample tunnel scenario, we use 20+ TMote Sky nodes [7] to model three tunnel sectors, as illustrated in Figure 4. We decrease the transmission power to create a *multi-hop* scenario in a limited space. As for actuation, we modified some of the nodes to control externally attached devices. Specifically, 12 V mini-fans and lights are used to model the fans inside the tunnel and the traffic lights at the entrances. For practical reasons, fire and presence sensors are "implemented" with light sensors, triggered using flashlights. Our setup is shown in Figure 5. Based on this setup, we showcase various use cases involving different logical neighborhood definitions, such as:

Use case 1: when presence sensors recognize a traffic jam on a lane, the fans are activated along the same lane from that location to the corresponding entrance, and the traffic light is turned red only on that lane. Figure 4 depicts the nodes involved in this case.

Use case 2: when light sensors read values above a safety threshold, the lights at the corresponding tunnel entrance are activated to avoid shadowing effects, and improve the visibility to drivers entering the tunnel.

Use case 3: when fire sensors detect the presence of fire in a sector, the fans in the same and adjacent sectors are activated, and the traffic lights are turned red on both ends of the tunnel.

Our demonstration also involves two laptops. One is used for illustration purposes, showing relevant code snippets and a high-level descriptions of the processing involved. Instead, the second laptop is moved inside the network to overhear packets in different positions. This lets the audience observe the current network topology, as well as understand how our routing protocol operates. Further, we plan to give flashlights to the public, to let them interact with our demo directly.

Acknowledgements. This work is partially supported by the European Union under the IST-004536 RUNES project.

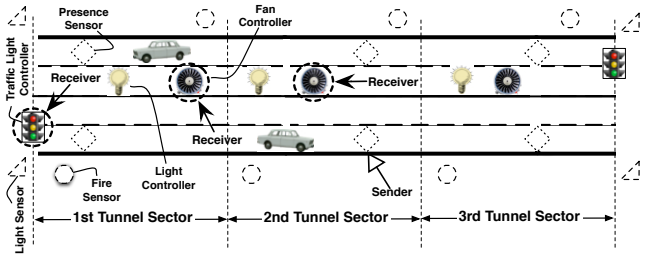

Fig. 4: Road tunnel scenario and nodes involved in use case 1.

Fig. 5: Overall setup and nodes controlling fans and lights.

4. REFERENCES

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

[2] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proc. of the* 1^{st} *IEEE Wkshp. on Embedded Networked Sensors*, 2004.

[3] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed Diffusion for wireless sensor networking. *IEEE/ACM Trans. Networking*, 11(1), 2003.

[4] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1), 2005.

[5] J. Hill et al. System architecture directions for networked sensors. In *ASPLOS-IX: Proc. of the* 9^{th} *Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2000.

[6] P. Costa et al. The RUNES middleware for networked embedded systems and its application in a disaster management scenario. In *Proc. of the* 5^{th} *Int. Conf. on Pervasive Communications (PERCOM)*, 2007.

[7] MoteIV Technology, `www.moteiv.com`.

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

[9] 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.