PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

On Neighbors, Groups and Application Invariants in Mobile Wireless Sensor Networks

Ştefan Gună

Advisor:

Prof. Gian Pietro Picco

Università degli Studi di Trento

November 2011

To my family
Familiei mele

This thesis, prepared under the supervision of **Prof. Gian Pietro Picco**, was reviewed by:

Prof. Christine Julien	University of Texas at Austin, USA
Prof. Kay Römer	University of Lübeck, Germany
Prof. Kamin Whitehouse	University of Virginia, USA

Abstract

The miniaturization and energy-efficient operation of wireless sensor networks (WSNs) provides unprecedented opportunities for monitoring mobile entities. The motivation for this thesis is drawn from real-world applications including monitoring wildlife, assisted living, and logistics. Nevertheless, mobility unveils a series of problems that do not arise in fixed scenarios. Through applications, we distill three of those, as follows.

Neighbor discovery*, or knowing the identity of surrounding nodes, is the precondition for any communication between nodes. As compared to other existing solutions, we provide a framework that approaches the problem from the perspectives of latency (the time required to detect an amount of contacts), lifetime (the time nodes are expected to last) and probability (the fraction of contacts guaranteed to be detected within a given latency). By formalizing neighbor discovery as an optimization problem, we obtain a significant improvement w.r.t. the state-of-art. We offer a solver providing the optimal configuration and an implementation for popular WSN devices.*

Group membership*, or knowing the identity of the transitively connected nodes, can be either the direct answer to a requirement (e.g., caring for people that are not self-sufficient), or a building-block for higher-level abstractions. Earlier works on the same problem target either less constrained devices such as PDAs or laptops or, when targeting WSN devices, provide only post-deployment information on the group. Instead, we provide three protocols that cover the solution space. All our protocols empower each node with a run-time global view of the group composition.*

Finally, we focus on the behavior of the processes monitored by WSNs. We present a system that validates whether global invariants *describing the safe behavior of a monitored system are satisfied. Although similar problems have been tackled before, the invariants we target are more complex and our system evaluates them in the network, at run-time. We focus on invariants that are expressed as first-order logic formulas over the state of multiple nodes. The requirement for monitoring invariants arises in both fixed and mobile environments; we design and implement an efficient solution for each. Noteworthy is that the solution targeting mobility bestows each node with an eventually consistent view on the satisfaction of the monitored invariants; in this context, the group membership algorithms play the role of global failure detectors.*

Keywords
[Wireless Sensor Networks; Network Protocols; Global Invariants; Distributed Algorithms]

Acknowledgments

I owe a great debt of gratitude to many people that have mentored me, collaborated with me, supported me, and shared their friendship throughout all the time I took to develop this work. In front of all, I would like to thank my advisor, Gian Pietro Picco, who has shown me what thorough research is and constantly pushed me to go beyond my limits. Perhaps the two most important lessons that I learned from him is the importance of argumentation and standing for my ideas.

I am also grateful to Christine Julien, Kay Römer, and Kamin Whitehouse who took the time to review my work and serve on my defense committee.

I would also like to thank everybody that had a direct contribution to this thesis: Amy L. Murphy (for a joint work that resulted in Chapter 4), Marco Cattani (for the results in Chapter 5), Luca Mottola (for the system in Chapter 6), Gianalberto Chini and especially Davide Molteni who provided tremendous help in the collection of valuable data required to support my work. I am double thankful to Amy, who also provided priceless advices towards the ending of my PhD studies. I would like to extend my gratitude to Daniel Petriceanu who provided insightful feedback on the analytical model in Chapter 4.

Outside of the thesis, I have enjoyed the work with Matteo Ceriotti and Daniele Fachin on the TeenyLime middleware, on which they built upon in their efforts in the tower [17] and tunnel [16]. I also owe a lot to Giuliano Mega for the stimulating discussions in front of the coffee machines that helped me keep a broader perspective on computer science.

Most importantly, some of these people have become my friends; I cannot be thankful enough for the quality time spent outside work with Adam, Amy, Chin8, Davide, Giuliano, G.P., Lele, Marco, Matteo, Meritxell, Pierre, Ramona, and Silvia. I also owe a lot to my friends back home, especially Alexandra and Ana, who kept me going on a straight line.

Although they have contributed only indirectly to this work, I owe the deepest gratitude to my parents and to Bogdan, to which I dedicate this thesis and who have always pushed me to pursue my dreams. Their encouragements, advices and support have played a key role in the completion of my studies.

Saving the best for last, I wish to thank Anamaria. Her kindness and endurance had helped me understand the true goals of my life.

Thank you all!

Ştefan

Mulțumiri

Sunt profund îndatorat tuturor celor de la care am învățat, cu care am colaborat, m-au ajutat și mi-au fost prieteni în timpul în care am realizat această lucrare. Înaintea oricui altcuiva, vreau să-i mulțumesc îndrumătorului meu, Gian Pietro Picco, cel care m-a învățat ce înseamnă cercetarea riguroasă și care m-a îndemnat în mod constant să-mi depășesc limitele. Probabil că cele mai importante două lecții pe care le-am învățat de la el sunt nevoia de a argumenta și de a susține ideile.

Sunt îndatorat și celor care mi-au revizuit teza și au făcut parte din comitetul meu de evaluare, Christine Julien, Kay Römer și Kamin Whitehouse.

Deasemenea, doresc să le mulțumesc și celor care au avut o contribuție directă la aceasta teză: Amy L. Murphy (pentru rezultatele din Capitolul 4), Marco Cattani (pentru rezultatele din Capitolul 5), Luca Mottola (pentru sistemul din Capitolul 6), Gianalberto Chini și în mod special Davide Molteni care mi-au oferit un imens sprijin în colectarea datelor necesare validării muncii mele. Sunt dublu îndatorat față de Amy, care mi-a sfaturi de neprețuit înspre finalul studiilor mele. Aș vrea să-i mulțumesc și lui Daniel Petriceanu, cel care mi-a oferit sprijin în dezvoltarea modelului din Capitolul 4.

Pe lângă teză, m-am bucurat să lucrez împreună cu Matteo Ceriotti și Daniele Fachin la middleware-ul TeenyLIME, pe care l-au folosit în eforturile lor din turn [17] și tunel [16]. Îi datorez mulțumiri și lui Giuliano Mega, în special pentru discuțiile din fața automatelor de cafea care m-au ajutat să îmi mențin o perspectivă largă asupra științei calculatoarelor.

Probabil că cel mai important lucru este faptul că o parte din aceste persoane mi-au devenit prieteni; nu le pot mulțumi suficient pentru momentele petrecute împreună în afara muncii lui Adam, Amy, Chin8, Davide, Giuliano, G.P., Lele, Marco, Matteo, Meritxell, Pierre, Ramonei și Silviei. Sunt îndatorat și prietenilor de acasa, în special Alexandrei și Anei, care m-au ajutat să merg pe o linie dreaptă.

Deși au contribuit numai în mod indirect la această teză, le datorez cea mai mare recunoștiință părinților mei și lui Bogdan, cei cărora le-am dedicat aceasta teză și cei care m-au îndemnat dintotdeauna să-mi urmez visele. Încurajările, sfaturile și ajutorul lor au avut un rol principal în terminarea studiilor mele.

Am păstrat ce este mai bun pentru sfârșit. Îi mulțumesc Anamariei pentru bunătatea și răbdarea pe care mi le-a arătat și care m-au ajutat să înțeleg care sunt obiectivele adevărate din viață.

Vă mulțumesc!

Ștefan

Contents

List of Tables

List of Figures

Introduction

Chapter 1

Introduction

Wireless Sensor Networks (WSNs) are a large collection of miniaturized untethered computing devices with scarce resources, self-organized and collaborating towards a common goal. These networks have been successfully deployed by scientists in a variety of environments and scenarios, e.g., historic buildings [17], road tunnels [16], vineyards [10], forests [114], volcanoes [123]. Although all previous success stories exploit an infrastructure with fixed nodes, researches have not neglected node mobility. In fact, recent applications such as medical care [23], shepherding [121] and monitoring of wildlife [33] or moving assets [46] denote the interest of the scientific community in transposing WSNs from fixed to mobile environments.

1.1 Goals and Motivating Scenarios

In the context of the fixed-to-mobile paradigm shift, this thesis addresses issues of WSNs mobility from the perspective of three fundamental problems: *1)* knowing all nearby nodes, *2)* knowing all nodes reachable through multi-hop communication, and *3)* knowing whether nodes behave according to a set of given specifications. Indeed, we maintain that these are key issues in the design of mobile WSNs. To understand why, we briefly focus on one of our research projects, ACube (`acube.fbk.eu`), where these problems come to the forefront of application requirements.

Funded by the local authorities of Trento, ACube studies how various technologies (WSN, video cameras, microphones, RFID tags) can be used as a means towards an assisted living platform. The resulting system will be deployed in several day-care centers for Alzheimer's impaired patients. A serious concern of the staff in these centers is the monitoring of:

1. patients in the proximity of other patients and social workers (to monitor the patients' social behavior) as well as of hazards such as stairwells and other dangerous areas (to monitor the patients' safety). Mainly due to privacy issues, we cannot employ video surveillance to solve this issue. Neither can we use IR sensors, as these do not convey the identity of a person. Instead, in our project patients and staff members carry WSN nodes, which we also use to tag hazards. Consequently, detection of proximity to people and/or hazards can be achieved when radios are

operated such that packets are received only at close range, which is not required to be highly accurate. Essentially, we use the radio as a proximity sensor.

Addressing this requirement requires solving the first problem tackled by this thesis, that is, discovering all nearby nodes.

2. patients that leave travel groups during trips outside the care centers. Usually, the trips occur in places where infrastructure is lacking. Therefore, an untethered technology such as WSN is useful to detect wandering patients in a fully autonomous and distributed fashion. However, as monitored patients must not feel they are under constant surveillance, they are allowed to exit the direct supervision range of the person attending the group. Consequently, the area to be covered can potentially be large and thus monitoring of groups must be performed across a multi-hop network.

   This requirement maps to the second goal set forth for this thesis, i.e., that of knowing all reachable nodes.

3. patients for which the surrounding environment is not appropriate, such as agitated patients not calmed by music. This provides motivation for a system that monitors application-level invariants, essentially predicates that describe the correct behavior of a monitored process. Arguably, monitoring invariants of a single node is simple. We are rather interested in *global invariants* , that is, those that are expressed over the combined state of several nodes. For instance, in ACube, such an invariant is "when on-body accelerometers report high activity, the music must be on". Violations of the previous invariant are detected through the observation of both the state of the accelerometer-enabled nodes and that of nodes monitoring music players.

   Notice that this requirement is the essence of our third goal: understanding whether nodes conform to a set of given specifications.

The previous research questions are not necessarily specific to our project, rather they are general problems lying at the core of a wider range of application scenarios. For example, we are collaborating with biologists that analyze the social behavior of roe deer. Their study relies on animal-to-fixed node contact traces (to infer location), on animal-to-animal contact traces (to study social contact) and on how the composition of herds changes over time (to study group behavior). Albeit detection occurs at greater distances, the first two problems emerge again: the discovery of all nearby nodes and that of all reachable nodes. Similarly, the need to monitor global invariants arises in other scenarios such as logistics. For instance, in the management of cold chains, it is essential to strictly control temperature in an uninterrupted series of storage and distribution activities. Faults in the chain can be identified by checking whether the refrigeration fans of a smart container are activated every time the temperature increases above a threshold. Unlike neighbor discovery and group membership, demands for global invariants also appear in applications where nodes are fixed, e.g., in business processes like the management of green buildings. We further expand on these scenarios in Chapter 3.

In the remainder of this chapter, we concisely illustrate the technical contribution of this thesis.

1.2 Contribution

The solutions to each of the aforementioned problems result in three building blocks that cover all three layers of a mobile WSN software stack, as depicted by Figure 1.1:

1. At the MAC layer, we design a *neighbor discovery protocol* that identifies all nodes in radio range. This building block provides person-to-hazard proximity detection and animal-to-animal contact traces in ACube, respectively the wildlife social study project.

2. At the routing layer, we design several algorithms that monitor *group membership*, i.e., that identify a node's transitively connected component. These protocols will be used in ACube to detect when patients leave their group and also to monitor herds in our wildlife project.

3. Finally, our contribution at the highest level, the application layer, is a system that monitors whether an observed process executes according to specifications. These are provided in a declarative language as a set of *global invariants*, or predicates expressed over the state of several nodes, and monitored by an efficient run-time. The system can be used either in a mobile (e.g., ACube, logistics) or in a combined fixed environment (e.g., business processes).

We briefly present each individual contribution next.

RUTh: Neighbor Discovery Made to Measure. Neighbor discovery, or knowing the identity of nodes in communication range, is a fundamental problem in mobile WSNs. Discovery *latency* and system *lifetime* are two key aspects along which this problem can be formulated. We address both in Chapter 4, showing how an optimal maximum lifetime can be achieved for latency-constrained scenarios and, dually, how latency can be traded off to optimally meet strong lifetime requirements. Moreover, by offering a choice between *deterministic* and *probabilistic* guarantees on discovery latency, we give developers another knob for optimizing their lifetime- or latency-driven application.

Although lifetime, latency and probability have been previously identified in the literature as dimensions of the neighbor discovery problem, we are the first to combine all three in a single, unified framework. Our approach relies on an analytical model, used at the core of an optimizer. The latter takes discovery constraints as input from application developers and outputs configuration parameters for our discovery protocol, RUTh ("R U There?"), implemented at the MAC layer of TinyOS. Our analysis validated by real-world experiments shows that RUTh outperforms other state-of-art approaches: for instance, given the same energy budget, it yields up to a 50% reduction of the discovery latency.

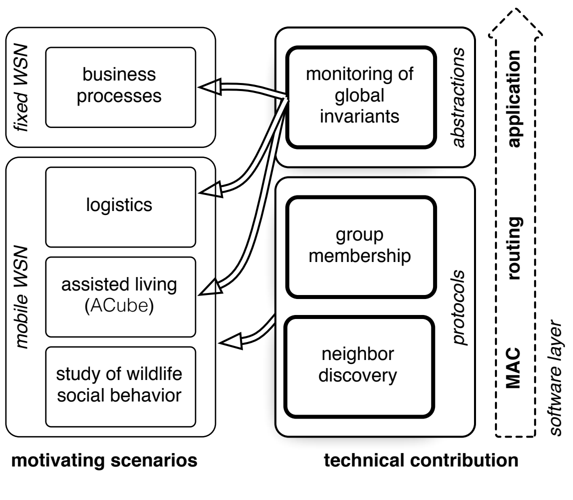


Figure 1.1: Overview of this thesis.

The Group Membership Problem. In Chapter 5, we design and compare protocols that enable nodes to have constant knowledge of which other nodes are reachable through multi-hop communication. An answer to this problem enables human-, animal- or asset-borne wireless devices to detect the joining or leaving of group members in infrastructure-less scenarios.

In contrast with Chapter 4, where we focus on knowing the neighbors within radio range, here we address the discovery of nodes beyond the physical range. We do so by analyzing three points that together cover the solution space. At one extreme, group membership information is *proactively* and collectively maintained by each node in the group. At the other extreme, dissemination of membership updates is triggered *reactively* by relying on a lower-level neighbor discovery protocol. In the middle lies a solution borrowing ideas from the two extremes. Group membership is maintained mainly as hard-state, however, nodes periodically broadcast state summaries to detect possible inconsistencies.

Different protocols exhibit different performance based on the degree of mobility. In this respect, we simulate artificial scenarios that allow us to assess the impact of mobility and group departures. We validate these through simulation of human GPS traces.

Depending on the application, the group monitoring protocols can either be used as stand-alone solutions, or can be used in conjunction with higher-level abstractions, such as the system introduced next.

Distributed Monitoring of Global Invariants. In Chapter 6, we present DICE ("Distributed Invariant CheckEr"), a system which monitors application-level global invariants in a WSN. A DICE invariant is a predicate that can be expressed as a boolean expression or as a linear inequality quantified over the state of multiple nodes. Invariants are specified in a declarative language. For example, referring to the cold chain scenario demanding the activation of refrigeration fans when the temperature of packages increases over a threshold, in DICE we specify the invariant denoting the correct behavior as:

```
invariant {forall m: type@m = FOOD_PACKAGE and temp@m > T}
          -> exists n: type@n = FAN and isActive@n}
```

where `temp@m` denotes the temperature of a food package `m`, and `T` is a constant.

An efficient run-time evaluates whether the monitored invariants are violated or complied with. The problem can be solved trivially by gathering all data from all nodes at a designated node where it is processed. However, this is inefficient in terms of communication overhead and detection latency. Therefore, we explore two alternate designs:

- a flat, decentralized solution, suitable for mobile, dynamic networks, such as those providing the core motivation for this thesis. Significant in this approach are the group membership algorithms in Chapter 5: these play the role of failure detectors, deciding whether a mobile node is truly departed or still contributing with state to the monitored invariant.

- a hierarchical, centralized solution exploiting in-network aggregation of the monitored state, suitable for scenarios where nodes are static. In this solution, only one node, i.e., the sink, is guaranteed to have complete knowledge on the network state and to properly detect invariant violations.

We implement both solutions and compare their performances using large-scale simulations and a real-world testbed. Results indicate that, in both cases, invariant violations are detected in a timely and energy-efficient manner. For instance, in a 15×15 grid network, violations are detected in less than a second and with only a few packets sent by each node.

Thesis Outline

Although we are aware that neighbor discovery, group membership and monitoring of global invariants are only some of the challenges related to mobile WSNs, we argue that these are among the most important ones, as seen in our application examples. Part I sets the background of this thesis. Specifically, in Chapter 2 we take a look at mobility in WSNs and in Chapter 3 we detail our application scenarios. Part II describes our contribution. We discuss the previous challenges and address each with an implementation at a different software layer. In Chapter 4 we present our neighbor discovery protocol which executes at the layer of MAC protocols. Our protocol uses an analytical model to optimally meet a given latency or lifetime target. The focus of Chapter 5 are protocols monitoring group membership; these run at the routing layer and their objective is to identify a node's connected component. Chapter 6 covers the application layer with a system that identifies whether the global state of the network complies to a set of given invariants. Finally, Chapter 7 concludes the thesis with an outlook on future research directions emerging from this work.

Part I

Background

Chapter 2

Mobile Wireless Sensor Networks

WSNs are a large collection of battery-operated small devices that perform the following steps: *i)* take sample measurements of the environment, *ii)* perform a minimal amount of processing, e.g., aggregation or compression of the sampled measurements, *iii)* transfer these through multi-hop communication to one or more collection points, and *iv)* possibly, in the case of sensor and actuator networks, operate a device which closes the control loop. The general assumption is that the network is transitively connected and static. As a consequence, WSN architectures, protocols, programming abstractions and their supporting run-time has been designed to support this view.

The work in this thesis deviates from this traditional view on WSNs in the sense that it addresses networks with mobile nodes. Similar paths have already been taken by other researchers. In this chapter we discuss the general motivation (Section 2.1) of mobile WSNs, we analyze the extent at which mobility appears in WSNs (Section 2.2), and we present the challenges of mobile WSNs (Section 2.3). We defer the analysis of the state-of-art topics specifically related to our contribution to the corresponding chapters.

2.1 Why Mobile Wireless Sensor Networks?

We distinguish in the relevant literature two main motivations for WSNs mobility:

- *Application requirements intrinsically demand mobile nodes.*

  One of the early works in the field [80] rightfully argues that WSN represent a non-intrusive, long-term and cost-efficient monitoring solution. Without the technology, some phenomena would otherwise be impossible to observe. The same type of monitoring is required for moving entities, may it be assets, animals or humans. Therefore, in this case, mobility of nodes becomes a functional requirement of applications. We review such applications in Section 2.1.1.

- *Mobility improves the performance of static deployments.*

  There are cases when, even if applications do not directly demand mobile sensors, they are exploited to address technical aspects. For instance, a number of papers suggest to relocate nodes in order to prevent premature energy depletion or to reduce radio interference. We review such papers in Section 2.1.2.

2.1.1 Mobile WSN Applications

WSNs offer unprecedented opportunities in the monitoring of mobile entities. In this section, we survey some of the most representative systems using mobile sensors applied on animals, humans and moving assets.

Animal study. One of most prominent application of mobile WSNs is in the field of animal study, e.g., badgers [33], turtles [109], zebras [56, 129], and our wildlife project described in Chapter 3. The most representative of these is in fact one of the early works in the field, that is, the ZebraNet project [56, 129], whose goals were to understand how, why, and when zebras undertake long-term migrations.

Shepherding. Applications of WSNs related to animals are not limited to "sense" only. Sensor and actuator networks have been actively employed in the shepherding of domesticated animals. In [121], the authors describe a system where an actuator, i.e., an animal-borne electric shock generator, prevents clashes between bulls in on-farm breeding paddocks. A similar system [11] was used to confine cattle within perimeters defined by virtual boundaries.

Asset sensing. Tracking of goods and freight is important in logistics, but also in day-to-day activities. For instance, in [81], the authors design a platform for monitoring freights. The hardware incorporates shock, sound, breach and environmental sensors and features a novel RF wake-up circuitry for greater energy efficiency. With a focus on indoor scenarios, systems such as [126] allow users to search the location of their everyday-use objects through queries issued in web browsers. In other cases, i.e., [66], the mobility of a sensor can be used in the exploration and charting of unknown structures, i.e, water pipes.

Human-centric and participatory sensing. Ongoing research suggests the utilization of human-borne sensors in assisted living systems like [125] and our project described in Chapter 3, health care [23], understanding of social behavior [67], in sports [35], or as a means to increase responsiveness in emergencies [52, 73, 98, 106]. In this context, human-borne motes provide identification [73], position tracking [62], motion control and posture detection [42, 72, 128] and vital sign monitoring [23]. WSNs may be retrofitted into hospitals [23], applied ad hoc at the place of a disaster [106] or can perform tracking and monitoring of individuals in remote places [35, 52].

An important amount of research has been undertaken on the sensing capabilities of cellular phones and other mobile "gadgets" that have lately became pervasive in our society. From this aspect, researchers are concerned mainly with *i)* participatory sensing, or on how resources donated by volunteers can be used to monitor a large scale phenomenon such as earthquakes [36], noise map [99], traffic conditions [88], or transit tracking [113], and *ii)* inferring information about the carrier's context or activities using sensors available on popular phones, e.g., cameras, microphones, accelerometers, gyroscopes [60, 74, 87]. Although most of these works target significantly more powerful platforms, some of the contributions of this thesis, e.g., neighbor discovery, can be transposed to this environment and serve as building blocks towards more complex functionality.

Environment monitoring. Mobile sensor networks may be used in an ad hoc deployment to monitor unpredictable phenomena which can be otherwise impossible to observe.

An interesting approach is the usage of WSNs on unmanned aerial vehicles, useful to understand atmospheric events, plumes and pollution [3].

Environmental monitoring using mobile entities have also been studied by other computer science areas, such as mobile and vehicular ad-hoc networks. For instance, researchers [53] envisaged that sensor on board vehicles can be queried to provide environmental and geo-imaging data through a SQL. In general, such networks employ devices that are more powerful and energy hungry, and use dedicated protocols (some surveyed in [105]), albeit some of the challenges are similar to the one faced in mobile WSNs, discussed in Section 2.3.

2.1.2 Improving the Performance of Static Networks

Mobility is also employed to address non-functional requirements and improve the performance of static networks. We discuss these possibilities next.

Sensing coverage. Two problems of WSNs are finding a node arrangement which maximizes the number of detected events or, dually, which minimizes the number of undetected events. As observed in [38, 43], these are actually two facets of the same problem, i.e., sensing coverage. When nodes are mobile, sensors can be relocated to increase the coverage in remote regions of the network on a per need basis. Consequently, a dense deployment is likely to become unnecessary, reducing thus the number of nodes and costs. Possible solutions to the coverage problem are described in [5, 43].

Connectivity and reliability. Similar to sensing coverage is the problem of network connectivity. Due to various causes, e.g., radio interference, node failures or node repositioning, the wireless network may become partitioned.

To mitigate this phenomenon, a greater number of nodes can be deployed to increase redundancy and connectivity to remote areas. However, this solution may be cost ineffective and potentially lead to other problems. For instance, an increased node density raises the risk of channel contention and collisions. The alternatives provided by mobile nodes are:

- mobile nodes can be repositioned to reconnect the severed nodes. This problem is similar to the one of sensing coverage, and the same solutions may be applied to both, e.g., [43].
- mobile nodes can visit nodes and transfer data using one-hop transmissions. This solution works best for collecting data in batches. Note that the same principle works both ways, that is, mobile nodes can disseminate data to fixed nodes, e.g., they may re-program the network with a newer software version. Mobile nodes collecting data are usually referred to in the literature as “data mules” [54, 58, 112].

In general, however, any adopted relocation scheme must ensure that node movement does not break connectivity.

Energy efficiency. Consider the typical WSN topology for sense-only applications: it resembles a tree, where each node relays towards the root data on behalf of itself or a number of other nodes. In the absence of any aggregation scheme, the closer to the root a node is, the more data it must relay. This phenomenon, called the “funneling effect” in [119], translates to a peak in the rate of energy consumption on nodes close to the root; in result, the operational lifetime of the network shortens considerably. As suggested in [119], this undesirable result can be diminished by a secondary fixed network with long(er) range data links that bypass the overloaded areas of the WSN.

Alternatively, one or more mobile sinks can be relocated to balance the energy consumption and increase the lifetime of the network as a whole. This is different from the previous case, where the goal was to increase sensing coverage and connectivity in certain areas of the network. Interesting analytical frameworks are provided in [4, 40, 75, 120], in which sink relocation is formulated as an optimization problem whose goal is to maximize the network lifetime.

Throughput. In addition to lifetime, it is important to asses the performance of networks. One metric that can be used is the network's throughput, an indication of how much information can be transmitted over the unit of time.

Interestingly, it has been shown that, at least theoretically, mobility increases the overall throughput of a network at the expense of the delivery delay. Intuitively, the mixture of node mobility, low power transmissions and multi hop communication reduces interference and allows data from more nodes to travel through the network at the same time. The maximum throughput increases from $O\left({}^{1}/\sqrt{n}\right)$, where n is the number of nodes, for static networks [47] up to $O(1)$ for mobile networks [45].

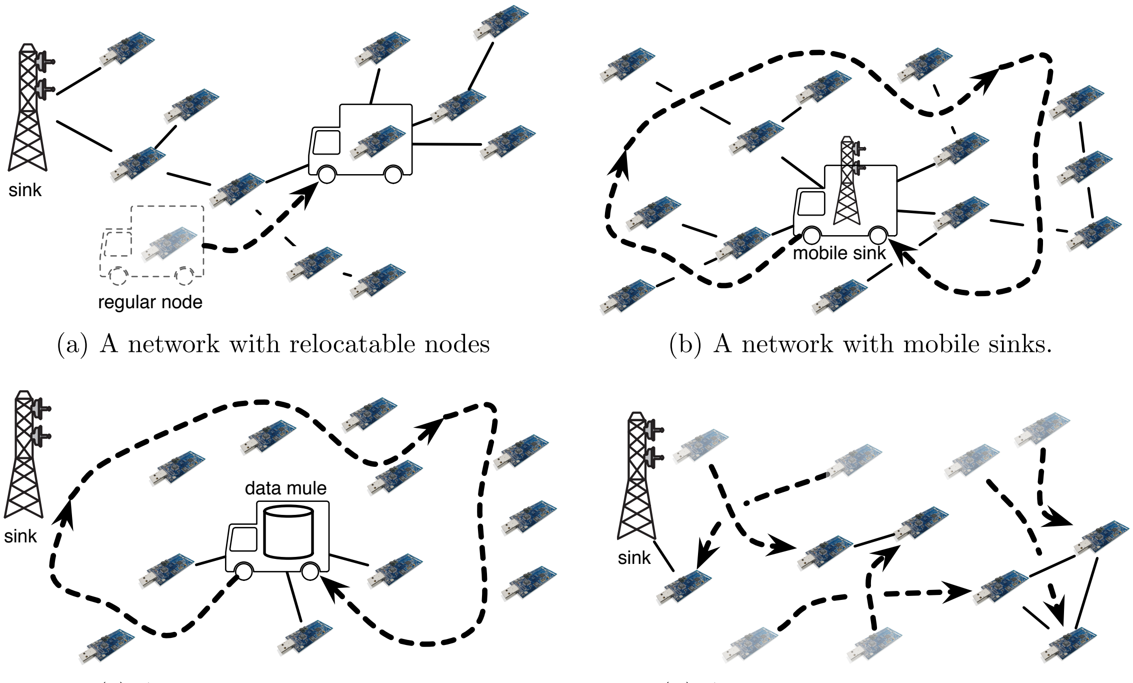
Security. In ad hoc networks in general and WSN in particular, establishing a secure route between nodes requires the existence of a key which is used by security protocols [59]. The key may be preset [19], or established at runtime [51]. The former may not offer sufficient flexibility and security, since keys are hardcoded in the memory of motes before deployment. The latter raises a circular dependency: in order to distribute a key in a multi-hop network, secure paths must first be established; however, secure paths cannot be established without first distributing a key. The procedures [51] to break this circular dependency are complex and unsustainable by the constrained hardware in WSNs.

Noteworthy is that, from this aspect, mobility indeed helps. Authors of [117] suggest that a simple solution to the problem of key distribution exists when nodes are mobile: once a node enters the direct communication range of another node, the two nodes can directly establish a security association, i.e., they exchange directly the minimal amount of data required to establish the non-repudiation, integrity and confidentiality of messages exchanged afterwards. The security association can be maintained across multi-hop paths once the node moves, and therefore the dependency between routing and security is limited.

2.2 Networks with Mobile Nodes

The aforementioned applications and techniques are supported by networks in which some or all networks nodes are mobile. In this section, we analyze the impact of mobility on network topology. The first step we take in this direction is to distinguish between the elements of a mobile network. We build on a survey [28] that describes networks as consisting of:

- *Regular nodes* that produce data and possibly relay data on behalf of other nodes.



(a) A network with relocatable nodes

(b) A network with mobile sinks.

(c) A network with data mules.

(d) A network with general mobility.

Figure 2.1: Examples of network topologies with mobile nodes.

- *Sinks* that consume information produced by regular nodes, i.e., they are the destination of the data. Networks usually have a single sink, but sometimes solutions using several sinks can be employed for load balancing and reliability [16].

- *Support nodes* that do not produce, neither consume data. Rather, they leverage on mobility to play the role of an intermediary data collector on behalf of sinks. Networks may have several or no support nodes at all.

As depicted in Figure 2.1 and also presented in [28], mobility in WSN can appear for any of the previous three types of nodes. We review these next.

Relocatable nodes. In Section 2.1.2 we discussed how mobility can increase the sensing coverage, connectivity and reliability of a network. Figure 2.1(a) depicts a network where one of the regular nodes is relocated to reconnect a part of the network which, due to an earlier failure, has been previously disconnected from the sink. It should be noted that this relocated node does not carry any data, but rather it is used to change the network topology. Similarly, it may be used to increasing the sensing coverage. A survey on the topic of connectivity and coverage can be found in [43].

Mobile sinks. When the deployment is sparse and it is impossible to form a collection tree spanning all nodes, leverage can be taken from a mobile sink collecting data. One such scenario is described in [6], in which sensors are deployed throughout the city and data is gathered using one-hop communication while mobile collectors passing in the vicinity of sensors.

In more complex cases, a routing structure must be maintained while the sink is moving. For instance, a deployment like in Figure 2.1(b) featuring mobile sinks can be used to load balance the energy consumption and increase the overall network lifetime. Here, differently from the previous case, regular nodes are fixed and form a collection tree rooted at sink (or the nearest one, if several sinks are being used), e.g., like in [4, 40, 120].

Data mules. An approach for the case when both the sink(s) and sensing nodes are static and disconnected requires the existence of one or more special support nodes that play the role of data mules. Like in Figure 2.1(c), data is collected from sensors by a mule using one-hop communication and delivered to the sink once the two are in communication range. In this scenario, the mule solely plays the role of a data carrier. Therefore, the mule must have a large storage capacity. Such a topology is described in [54].

General mobility. When the monitored entities are mobile, e.g., [129] and our wildlife application (Chapter 3), the network experiences the most general form of mobility, as depicted in Figure 2.1(d). In this case, node mobility cannot be described using a structure. Moreover, a node can be an originator of messages, but also a data mule that relays messages on behalf of other nodes. In these networks, also called *delay tolerant networks*, a contemporaneous end-to-end path between the message originators and source is not guaranteed; routing of packets occurs opportunistically when two nodes encounter one another, as in [130]. This general form of mobility can result from application requirements (e.g., monitoring of zebras [129]), but can also be exploited to observe phenomena too large to be covered by a fixed network (e.g., as suggested by [3]).

This characterization of mobile WSNs is the first step in understanding the interactions between mobile nodes and their corresponding challenges. We describe these next.

2.3 Challenges of Mobile Wireless Sensor Networks

In Section 2.1.1 we reviewed a wide range of applications entailing mobile nodes. To develop such applications, developers face a number of challenges that emerge at different layers:

- From the networking aspect, developers are concerned with developing energy efficient protocols that meet the requirements of the applications.
- The programming platform has a direct impact on the difficulty of the software development process. Expressive and efficient languages, abstractions or middleware platforms make the difference in complex applications.
- The application logic is a difficult problem in itself. Consider, for instance, the shepherding application in Section 2.1.1; a poor decision may result in useless electric shocks applied to the wrong bull.

Hereafter, we focus on the networking (Section 2.3.1) and programming aspects (Section 2.3.2) of mobile WSNs. In this section, we purposely neglect application logic as it varies greatly with the application scenarios. Moreover, we have already address this subject in Section 2.1.1.

2.3.1 Networking With Mobile Nodes

Recent experiments [23] demonstrate that unmodified state-of-art collection protocol, e.g., CTP [44] are unsuitable when mobility appears. Moreover, a study [7] shows that if the paths of two motes intersect, they have a time window during which they can communicate[1] that ranges from 61 s down to 2 s for a relative velocity between motes of 1 m/s, respectively 5.55 m/s (20 km/h). Communicating in a mobile environment is thus a difficult task.

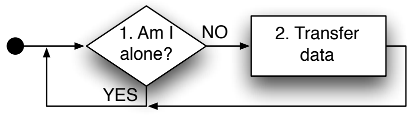


Figure 2.2: Networking flow in mobile WSNs.

From the point of view of communication, we characterize the process using the flow in Figure 2.2:

1. *Contact detection.* Communication between two nodes is possible only when they are within radio range; at any other point in time, communication attempts are not only futile, but also waste precious energy.

2. The actual *data transfer* with or without routing of packets. As we have just seen, the time window for communication may be very short. Therefore, when two nodes are in contact, communication efficiency should be at its utmost.

3. *Contact detection revisited.* Several throughput-related optimizations are possible when two nodes are in contact, e.g., the radio duty-cycling [96] can be disabled. However, once contact is lost, these optimizations are wasteful and must be disabled to increase lifetime.

We review each of the these states next.

Contact detection. Neighbor discovery, or contact detection, is the problem of identifying the nodes in radio range. This information is the precondition for any communication between two nodes. Protocols for contact detection can equally identify the departure of neighbors and, consequently, can be used to tear-down any data transfer. Protocols for neighbor discovery fall into two main categories, as follows.

In the first class fall *asynchronous* protocols that make no assumption on the time when the contact occurs. Two nodes discover each other by exploiting schemes in which sleep intervals are interleaved with beacon broadcasts or samples of channel activity. Generally, the longer the sleep interval is (w.r.t. the durations corresponding to broadcasts and channel samples), the longer both node lifetime and discovery latency are. Our neighbor discovery protocol falls into this category; we discuss similar protocols, e.g., [32, 57, 85, 116], in Chapter 4.

The second class of protocols, the *scheduled rendezvous* protocols, exploit information on the mobility and patterns of the interaction among nodes. By making assumptions on when nodes are likely to be in contact, these protocols afford longer sleep periods and wake up the radio only at pre-defined times. The wake up schedule can be based on:

[1]The experiments in [7] involved two nodes such that a mobile node passes as close as 15 m to a static node. The time window during which nodes can communicate is defined as the duration of the interval when the achieved packet delivery ratio is greater than 75%.

- already existing models of the interaction patterns, like in [129],
- already existing knowledge on the route followed by mobile nodes, e.g., a bus carrying a data mule and passing through the vicinity of sensors [18], or
- knowledge gained at run-time from previous encounters [34].

Irrespective of the adopted method, scheduled rendezvous has the advantage of a decreased energy consumption and, implicitly, an increased lifetime. However, such protocols must rely on time synchronization that, considering the disconnected nature of the network, can only come from an external source such as a GPS receiver. Additionally, these protocols are unsuitable for the cases when the interaction patterns are random and when the application demands the detection of contacts that fall outside a pattern.

Data transfer. As described earlier in this section, nodes may have a limited time window during which data transfer can occur. Therefore, during this time, throughput must be maximized. Data transfer can be as simple as an offload to a neighboring mule [54] or a fixed routing structure [23]. Another interesting case appears when the sink is mobile and the routing structure must cope with the mobility of the sink. The general approach in this case is to exploit one ore more fixed nodes in the neighborhood of the sink and build a hierarchical routing scheme. For instance, in [61, 127], the authors describe protocols where mobile sinks select a "proxy" node, that is, a single node that connects a sink to the remainder of the network.

Complex cases entail routing, possibly in an arbitrary fashion across mobile nodes similarly to the case of mobile ad hoc networks. To further increase difficulty, paths can be transiently disconnected, as in the case of delay tolerant networks. However, the subject of end-to-end routing in mobile networks falls outside of the contributions of this thesis; we refer the interested reader to relevant surveys on the topic [105, 130].

2.3.2 Application Development

A decade has passed since the first release of TinyOS [49], the de facto operating system for WSNs. During this time, a wide landscape of programming abstractions and middleware platforms for WSNs [91] have emerged; a proper abstraction reduces the programming hurdle without sacrifices in efficiency. Interestingly, none of the higher level programming platforms explicitly addresses mobile networks. In practice, applications are developed using low level primitives in an ad hoc manner. No reference software architecture for mobile WSNs has emerged so far, even in the case when middleware platforms have been employed in mobile applications. Consider for instance Impala [71], the middleware layer used in the ZebraNet project; although it offers an interesting approach to code modularity, reprogramming, and concurrency, there is no support genuinely dedicated to node mobility.

In the literature describing high-level abstractions for WSNs, there are however several indirect references to node mobility. Interestingly, most relate to the problem of node identity, which we address in this thesis at various levels. For instance, in TeenyLIME [26], the middleware transparently manages a list of one-hop neighbors. Abstract Regions [122], Pleiades [64], Hood [124] all build high level operators such as enumeration, data sharing, annotation on top of one- or multi-hop neighborhoods. From the perspective of our contribution, RUTh can be in principle employed as a underlying layer to offer these abstractions a view on the one-hop neighborhood and, similarly, our group membership protocols to provide a view of the neighborhood at the multi-hop level.

Differently from the state-of-art, in Chapter 6 we present a run-time solution of our application invariant monitoring system which is designed from start for mobile nodes. Unlike the aforementioned abstractions, DICE is not a general purpose programming solution. Rather, it allows a simple development of applications where the goal is to monitor processes that can be described using logical expressions. DICE makes use of our contributions at the lower levels, for instance, it employs group membership as a global failure detector. However, group monitoring is only a building-block for the run-time and does not appear in the programming abstraction, which to which node mobility is transparent.

2.4 Discussion and Outlook

Mobility of wireless sensor networks has emerged not only as a direct necessity of applications, but also as an elegant solution addressing non-functional requirements, e.g., to increase sensing coverage. In this chapter, we reviewed these two aspects, and we also surveyed the literature to identify the degree in which mobility appears in network topologies. We discuss challenge occurring during the development of mobile applications from the perspective of networking and programming. Our contribution directly enables several applications such as monitoring of wildlife, but can also be envisaged to provide services for higher level abstractions.

Chapter 3

Application Scenarios

The research in this thesis is driven by several real-world applications scenarios addressed during the Ph.D. studies. In these applications, node mobility appears at various extents: some applications are based on purely mobile networks, other applications are a mixture of mobile and fixed nodes, while some solutions we describe are applicable to static scenarios. Moreover, at the time when this thesis was written, the applications described in this chapter had different levels of maturity:

- We have already designed, implemented and tested a solution for the ACube assisted living scenario. We present this application in Section 3.1, and we describe the role of our protocols in ACube and data gathered from a test deployment in Chapter 4.
- We have been recently provided with the hardware enabling the wildlife monitoring project described in Section 3.2. We designed and implemented an initial solution to this problem, which we are currently testing in the field.
- In our group, there is currently ongoing work on applying WSNs as supporting means for business process. We are presently assessing opportunities of applying DICE, the system described in Section 6, in scenarios similar to the those described in Section 3.3.

Next, we discuss each application in more details.

3.1 Assisted Living: The ACube Project[1]

There are no doubts that aging is a widespread phenomenon. In fact, by 2050 there will be more elderly than children and the population's average age will be 45 everywhere but in Africa [63]. The frequency of degenerative diseases associated to old age will increase accordingly: one person in 85 will be suffering from the Alzheimer's disease in 2050 [9]. In this somber reality, new technologies are sought to achieve greater efficiency and lower costs in social care.

One application of the research in this thesis is ACube, a project funded by the Autonomous Province of Trento. The aim of this project is to build technologies for an

[1]An early description of the ACube application appeared in [21].

assisted living platform that targets people suffering from cognitive impairments, mainly patients suffering from the Alzheimer's disease.

3.1.1 Scenario and Requirements

The small-size, accuracy, and ease in deployment make WSNs more suitable to be retrofitted in care centers w.r.t. traditional monitoring means, such as video cameras, RFIDs and microphones. In addition, WSNs are less intrusive and can be used to monitor patients in locations where privacy is of concern. In ACube, a number of fixed WSN nodes will be deployed throughout several day-care facilities, while patients and care-givers will carry mobile nodes that

- raise alerts when patients are in the proximity of hazards,
- report social contacts among patients, and between patients and care-providers, and
- detect basic posture changes, i.e., falls or long immobility periods.

Thus, in ACube, the detection and reporting latencies of such events are important as both increase the responsiveness of social workers. Moreover, considering that in ACube mobile nodes can be collected and recharged on a daily base, we can trade off lifetime to decrease event detection latency.

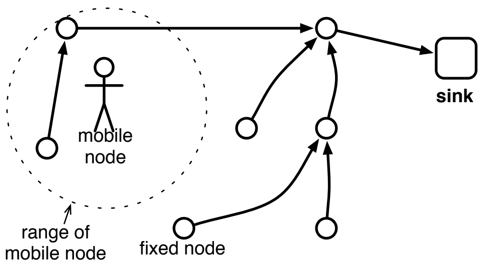


Figure 3.1: ACube network topology.

3.1.2 Solution Overview

As illustrated in Figure 3.1, our network consists of a mixture of mobile and fixed nodes:

- Mobile nodes detect events, i.e., proximity, contacts, falls. They are a scaled down version of the TMote SKY [97] featuring an on-board accelerometer and a rechargeable battery. Figure 3.2 shows the picture of a mobile node, including its casing.
- Fixed nodes play the role of hazard markers and form the infrastructure necessary to relay events on behalf of mobile nodes on a multi-hop path to the sink; the latter is a central node connected to a PC which dispatches events to operators. In ACube, the role of fixed nodes is played by regular TMotes.

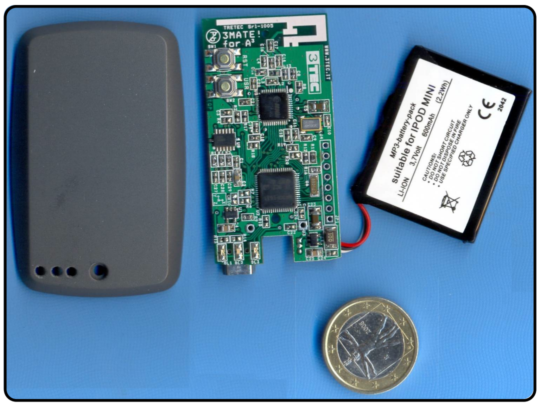


Figure 3.2: The mobile node in ACube.

The software architecture of our solution is built around the TeenyLime middleware [26]. TeenyLime empowers WSN programmers with a higher level of abstraction

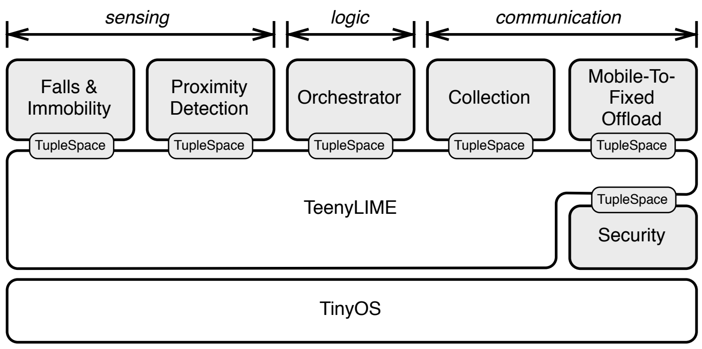


Figure 3.3: ACube software architecture.

that replaces the OS-level communication constructs with a shared memory space spanning neighboring (1-hop) nodes. The shared memory is provided in the form of a tuple space, i.e., a collection of sequences of typed fields, and is represented by a memory block on each node shared with other nodes within communication range. Along with the higher-level tuple space abstraction, TeenyLIME provides interfaces that allow a low-level configuration of the underlying hardware. TeenyLIME is built on top of TinyOS [49] and applications are developed in the component model inherited from nesC.

The ACube components interact exclusively through the tuple space abstraction, as indicated by the architecture illustrated in Figure 3.3. This allows a high decoupling of components, providing the flexibility to alter or insert only those components required to cope with changes in the deployment or in the project's requirements. For instance, the software configuration of a fixed node does not include functionality specific to fall and immobility detection, which is available instead on mobile nodes. We overview these components next:

Proximity Detection This component is included on both mobile and fixed nodes and its duty is to notify when mobile carriers approach fixed nodes or other carriers. The operation of this component is based on the neighbor discovery protocol described in Chapter 4.

Falls & Immobility This component is included only in the configuration of mobile nodes. Fall and immobility detection is achieved through an algorithm that uses accelerometers to detect the posture and peaks in the carrier's movement. The fall detection functionality is out of the scope of this thesis; it is described in [21].

Orchestrator Included on all nodes, the Orchestrator coordinates the activities of other components so that they do not interfere in their operation. It achieves this by multiplexing hardware sensors, controlling the radio duty cycle and scheduling transmissions according to the radio duty cycle. For instance, this component manages the radio chip in order to address the communication demands of the proximity detection and collection components; in this case, the Orchestrator decides when the radio should be on and when transmissions should occur.

Mobile-To-Fixed Offload Mobility of patients and care providers prevents us from building a collection tree that include mobile nodes. Consequently, these offload data to neighboring fixed nodes that further relay the information further. This process is handled by a specialized component, running on both fixed and mobile nodes. We present the behavior of this component in Chapter 4.

Collection This component is included only on fixed nodes and its purpose is to forward events to the sink along a collection tree formed by all fixed nodes. To build the tree and ensure forwarding, we adapt protocols previously developed in our group to address other deployments [16, 17], as described in Chapter 4.

Security Differently from other components, the Security component lays between TeenyLime and TinyOS. Its purpose is to provide an encryption layer that prevents any unauthorized messages to pass through the radio stack. In this respect, we use the software solution available at [29].

RUTh, our neighbor discovery protocol, lies at the core of the software architecture of the WSN in ACube. Its nature, however, poses peculiar challenges to any protocol running on top of it. We discuss these aspects at large and evaluate them in Chapter 4.

3.2 Study of Wildlife Social Behavior[2]

The understanding of social interactions is a stepping stone towards a better understanding of evolution, of endangered species, of the mechanisms behind epidemics, of the equilibrium in fragile ecosystems, and of the abilities to adapt to the surroundings. Study of wildlife has been traditionally performed by biologists using direct observation, although the effectiveness of this method is severely hampered by the human intrusion in the environment and impossibility of a full observation coverage in space and time.

Consequently, considerable research efforts have been invested into energy efficient technologies (e.g., GPS [12], RFID [33]) that allow an untethered, non-intrusive, and long term monitoring of wildlife. Along these lines, WSNs are suitable to the wildlife monitoring problem because of their small form factor, energy efficiency, and, most important, of their ability to embed application logic into the environment (e.g., [121]).

3.2.1 Scenario and Requirements

We are working with biologists that are analyzing the social behavior of roe deer. They are interested in:

- information on the contacts between animals, and
- information on where deer spend time.

In our wildlife monitoring system, we tag each monitored deer with a node as described in the next section. Our solution must take into account that deer are solitary animals.

[2]An early description of the wildlife monitoring application appeared in [89].

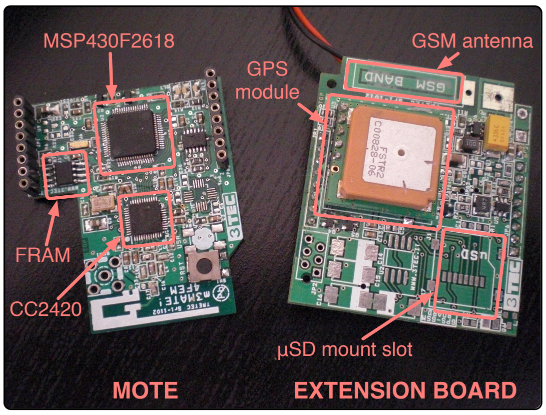


Figure 3.4: Nodes used in the wildlife project.

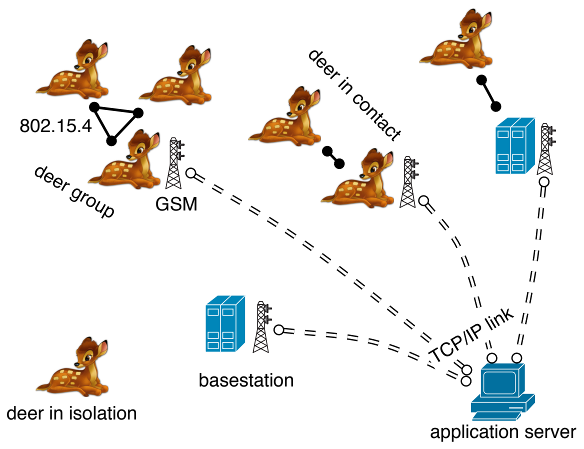


Figure 3.5: Wildlife network topology.

Consequently, the system must be designed to operate at maximum efficiency when nodes are disconnected w.r.t. other nodes. Moreover, deer are difficult to capture [95], thus, differently from ACube, a greater emphasis must be made on lifetime, rather than latency.

3.2.2 Solution Overview

In this project, we use motes purposely designed to meet the previous requirements. Our motes are derived from the TMote SKY [97] and feature a Texas Instruments MSP430F2618, a MCU from the MSP430 family from a more recent generation w.r.t. the one empowering standard TMotes. Compared to the TMote, the new MCU is faster and has larger code storage[3]; it is necessary to handle a larger amount of peripherals, as follows. We use the standard Chipcon CC2420 transceiver for its networking capabilities, but we also employ it as a "contact sensor". To increase lifetime, collected data is persisted on a FRAM chip, a fast, non-volatile memory, operable at lower voltages w.r.t. to standard Flash memory[4]. An extension board provides a combination of the following peripherals: a GPS module, a GSM modem, and a µSD card for additional storage.

As per Figure 3.5, we distinguish the following categories of entities in this project:

- All monitored deer bear a mote that features at least a GPS device. These use the radio transceiver to run the neighbor discovery protocol described in Chapter 4. All detected neighbors and localization information are persisted in the FRAM.

- We employ several fixed basestations, each equipped with a GSM modem and SD storage, to collect data from the deer-borne motes. We deploy these in the proximity of feeding stations that biologists expect to be repeatedly visited by deer.

[3]The new MCU that we use runs at 16 MHz, has 116 KB of Flash memory and consumes 365 µA and 0.5 µA when active, respectively when in standby. Comparatively, the MSP430F1611 on standard TMotes runs at 8 MHz, has 48 KB of Flash memory and consumes 330 µA and 1.1 µA when active, respectively when in standby.

[4]The Flash memory on the standard TMote is inoperable when the power supply drops below 2.7 V. Instead, our 1 Mb FRAM can be operated when the supply is as low as 2 V.

- The hardware configuration of a fraction of the deer-borne motes includes the GSM modem. These play the role of mobile basestations. While in principle all deer can bear a GSM modem, this solution is not feasible in practice due to cost constraints.
- The basestations, including the deer with GSM modems, report the downloaded data to an application server using a TCP/IP connection established through the GSM modem. The server runs on a PC in our lab.

The wildlife monitoring application is built on top of our TeenyLIME middleware and has an architecture similar to the one in the ACube project. Figure 3.6 sketches our solution. Hereafter, we detail the components included in the configuration of deer-borne motes:

Proximity Detection This component is similar to the component bearing the same name in the ACube architecture described in Section 3.1. Its goal is to detect contacts between deer and between a deer and a basestation. The operation of this component is based on the neighbor discovery protocol described in Chapter 4.

Location Manager Essentially, this component is a driver for the GPS chip. The location manager acquires the position periodically to maintain fresh information regarding the satellites in view. Additionally, it refreshes the positioning information when a contact is detected, managed as described next.

Contact Manager A dedicated component is dedicated to manage ongoing contacts between deer. The purpose of this component is to maintain a list containing the other nodes in the range and detect when the contact with one of these is lost. When this occurs, contact information — identity, duration, fresh position sampled from the location manager – is persisted on the FRAM.

Mobile-To-Basestation Offload This component implements the policies of the protocol transferring data (i.e., contact and position) from mobile nodes to basestations. In this protocol, a basestation plays the role of an arbiter ordering the transmissions of the deer-borne motes in range. Specifically, it gives higher priority to the nodes with a lower amount of available memory and battery. To increase throughput, data transfers occur in bulks, i.e., a deer-borne mote offloads a predetermined amount of data, and without any form of duty-cycling.

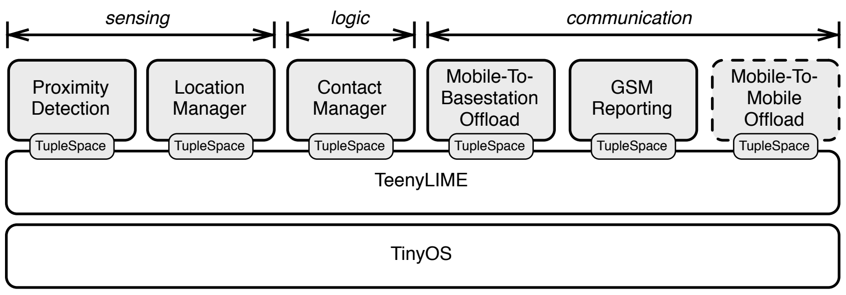


Figure 3.6: Software architecture for monitoring of wildlife.

GSM Reporting The GSM modem is managed by this component. Our experiments show that GPRS data connections are established at a high energy overhead[5]. Therefore, we prefer to transfer data to our application server in bulks during infrequent modem connections. All data collected between consecutive GSM offloads are stored on the high capacity µSD card.

Mobile-To-Mobile Offload We are currently investigating methods enabling an opportunistic multi-hop collection protocol similarly to [129]. Our goal is to enable deer-borne motes to offload their data to other deer-borne motes if the latter have higher changes of delivering data to a basestation. This is part of ongoing unfinished work which ultimately will become the mobile-to-mobile offload component.

The wildlife project provides a straightforward application for the research undertaken in this thesis. Moreover, it provides the possibility to analyze our neighbor discovery protocol in a context where system lifetime bears a higher weight w.r.t. to latency, which instead is more important in ACube. We revisit the wildlife application in Chapter 5, where we discuss the opportunities for integrating RUTh, our neighbor discovery protocol, with the group monitoring protocols.

3.3 Supporting Business Processes

Since the inception of the field, scientists leveraged wireless sensor networks (WSNs) to harvest large amounts of environmental data for off-line analysis [80, 83]. Alongside these applications, advances in power efficiency and miniaturization have also enabled the use of WSNs for online monitoring of business processes. The research on global invariants (i.e., properties over data sensed at different nodes) we present in Chapter 6 was originally motivated by scenarios in which nodes are partially mobile. One such scenarios is the management of supply-chains, discussed next.

Supply-chain management. Consider food transportation, a business where $35 billion are literally trashed yearly [50]. Increasingly often, shipped products are equipped with an RFID tag, which however only allows tracking the location of the item at specific points along the supply chain. WSNs devices enable continuous, fine-grained monitoring of the storage conditions, preventing deterioration of goods.

In this context, it is often essential to monitor the temperature of the packages where the food is stored. A typical invariant may concern the relation between the package temperature and the operation of the refrigeration system

> (**I1**) *When the temperature of packages is above a threshold, there must be at least one refrigeration fan active.*

Nevertheless, the transportation process is subject to further constraints. For instance, a non-uniform load in a container may cause stability problems during transportation. WSN nodes may be installed in a food container to ensure that

[5]We empirically determined that connecting to a GSM network and establishing a GPRS connection takes 15 s cca., time during which there is a current surge of 2 A.

(**I2**) *The weight difference between any two sampling points in a container must remain below a threshold.*

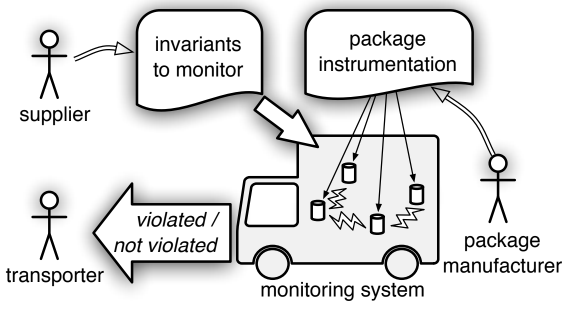


Figure 3.7: Supply-chain scenario.

Figure 3.7 illustrates how the WSN-based monitoring infrastructure integrates with the transportation process. Package manufacturers embed WSN nodes in their products. Food suppliers define the invariants ensuring correct shipping and handling of their products, and install them on the sensors monitoring packages. Transporters receive notifications of invariant violations, and take corrective actions in response. Different suppliers may provide different invariants for different products. Thus, multiple invariants must monitored simultaneously when products travel together. This scenario decouples the interactions among the various actors, simplifying management and increasing flexibility.

In the example above, the composition of the system is dynamic; all nodes are mobile. Furthermore, they join and leave the network as transporters load and unload freight. In Chapter 6, we describe a distributed approach to invariant monitoring; this is the key asset of a flexible deployment of mobile nodes, able to operate virtually anywhere in an autonomous fashion.

From Mobile To Fixed Nodes. Some mobile applications can easily be migrated to a fixed environment. Indeed, monitoring of invariants can be equally monitored by the nodes of fixed network. This is, for instance, the case of a scenario in make*Sense* (project-makesense.eu), an EU FP7 project to which our research group contributes and whose aim is at simplifying WSNs programming and their integration in business processes.

The aforementioned make*Sense* use-case is concerned with the ventilation systems deployed in green buildings. A fault of the Heating, Ventilation, and Air-conditioning Controller (HVAC) can be identified solely by checking whether the pressure exhibits variations throughout the building. In other words, by detecting violations of the invariant

(**I3**) *The difference in pressure between any two sampling points must remain below a threshold.*

which closely resembles the invariant I2 in the cold-chain application. However, differently from the previous application, the make*Sense* scenario is inherently fixed; once deployed, the HVAC and sensors inside the building remain immobile. While in principle the monitoring approaches for mobile nodes are also amenable to static networks, a different, more efficient solution can be adopted to tackle the case of fixed nodes. We address both in Chapter 6, where we present *i)* a decentralized solution suitable for dynamic networks, and *ii)* a centralized solution targeting static nodes.

3.4 Discussion and Outlook

In this chapter, we presented the applications that drive the efforts in this thesis. In each, we identified possible avenues for the research hereafter presented. We further identified that some of the problems we address for mobile nodes, specifically the need of monitoring application invariants, arise also in scenarios where nodes are fixed; these can be addressed with a dedicated, efficient solution.

Part II

Technical Contribution

Chapter 4

RUTh: Neighbor Discovery Made To Measure

Neighbor discovery, or knowing the identity of nodes in communication range, is a fundamental problem in mobile wireless sensor networks. Discovery *latency* and system *lifetime* are two key aspects along which this problem can be formulated. We address both, showing how an optimal maximum lifetime can be achieved for latency-constrained scenarios and, dually, how latency can be traded off to optimally meet strong lifetime requirements. Moreover, by offering a choice between *deterministic* and *probabilistic* guarantees on discovery latency, we give developers another knob for optimizing their lifetime- or latency-driven application.

Although lifetime, latency and probability have been previously identified in the literature as dimensions of the neighbor discovery problem, we are the first to combine all three in a single, unified framework. Our approach relies on an analytical model, used as the core of an optimizer. The latter takes discovery constraints as input from application developers and outputs configuration parameters for our discovery protocol, RUTh, currently implemented on top of TinyOS 2.1. In comparison to other approaches, RUTh shows improved performance: for instance, given the same energy budget, it yields up to a 50% reduction of the discovery latency.

4.1 Introduction

The reduced size and flexible nature of wireless sensor network (WSN) nodes open a multitude of application scenarios in which participants carry nodes. Being small, WSN nodes are non-intrusive and less likely to influence the behavior of the carrier. Nodes are designed to be extended with sensors customized to the application. Also, they are more privacy-friendly and less power-hungry than, say, video-cameras. Privacy is important in applications for sensing people's activities and environment. A reduced reliance on power availability increases the deployment options.

In applications involving mobile WSN nodes, one challenge is to manage the interactions among them and with nodes belonging to a fixed infrastructure. The key building block is *neighbor discovery*, or the process of determining when a pair of nodes is within communication range. This process must be both energy-efficient and tailored to the application needs. We concretely explore the space of alternatives through two applications that motivate this research.

Example 1: Assisted living. At an Alzheimer's day care center, we work with health care providers to support their daily routines. For example, WSN nodes are used to detect the proximity of patients to hazardous areas such as open exits and stairwells. This is accomplished by tagging each hazard with a node, and using low-power radio beacons to detect when a hazard tag and the mobile node carried by a patient are within range, thus reducing hazard proximity detection to neighbor discovery. The beacon rate influences the discovery latency, i.e., the time it takes two nodes in range to discover each other. In our application, latency is determined by the acceptable delay to detect proximity and the motion abilities of patients. However, different configuration of the beacons may provide the same detection latency; therefore, we must choose the configuration that yields the lowest energy consumption and, inherently, the largest system lifetime. In brief, the problem of *latency-driven* neighbor discovery is:

> *How should two nodes behave such that one discovers the other by a given maximum latency while minimizing energy consumption?*

Example 2: Wildlife monitoring. We also work with biologists studying wildlife behavior. Currently, GPS-enabled devices track animal movements [12] from which interactions are inferred. However, GPS is energy-hungry and requires clear sky, making it unsuitable for frequent sampling or underground deployments with burrowing animals. Also, in many studies (e.g., those concerned with disease spread or social interactions) precise GPS location is not required. Biologists need to know only which animals are close to one another, for which neighbor discovery is sufficient.

A limiting factor in these deployments is collar weight, which must be light enough to avoid affecting the behavior of the tagged animal. Battery weight dominates over all other electronics, placing firm limits on the total energy available. This is at odds with the fact that in many cases (e.g., for solitary, long-range animals) the notion of contact implies a much larger distance than in the assisted living example, thus requiring high-power beacons. Still, the system must guarantee a lifetime sufficient for biologists to glean statistically-relevant information, e.g., a season, a year, etc. Similar to the previous problem, different radio configurations that vary in the provided discovery latency may yield the same lifetime; out of these, we must choose the configuration that results in the lowest latency. Therefore, the problem of *lifetime-driven* neighbor discovery is:

> *How should two nodes behave such that, given a minimum expected lifetime, one discovers the other other with minimal latency?*

Nevertheless, it may be that, to meet the required lifetime, the resulting latency is unacceptably large for the application. In this case, and in general as a trade-off developers can play with, we offer an option that decreases the discovery latency for *some* neighbor discoveries. In other words, we have hitherto assumed that 100% of discoveries occur within the specified latency: *deterministic* detection. But, if the application can accept that *at least*, e.g., 80% of detections happen by the desired latency, we can lower energy consumption and increase system lifetime. These *probabilistic* discovery requirements can be applied both in latency- and lifetime-driven neighbor discovery.

Goals, contributions, and roadmap. In this work we define neighbor discovery in a way that makes explicit the trade-offs between energy consumption, discovery latency, and discovery probability. While others have explored energy-efficient neighbor discovery (Section 4.2), we are the first to address these trade-offs in a single, unified framework for which we offer:

- a neighbor detection protocol, RUTh (“R U There?”);
- a tool chain that accepts problem constraints from end-users and, based on an analytical model of RUTh that takes into account low-level characteristics of the radio, outputs the optimal protocol parameters.

RUTh itself has a simple, periodic behavior: at a certain point in each period, each node broadcasts a beacon, takes several channel activity samples to detect beacons originating from possible neighbors, and sleeps for the rest of the time. Each node has the same behavior, yielding uniform energy depletion. The system model and protocol are outlined in Section 4.3. The length of the beacon, the number of channel samples, and the length of the period are the fundamental configuration parameters, whose values determine determine the discovery latency, the system lifetime, and whether the discovery is deterministic or probabilistic.

The tool chain, outlined in Figure 4.1, allows a domain expert to specify a discovery probability and either the maximum latency (for an assisted-living-style problem) or the minimum lifetime (for a wildlife-style problem). We feed these constraints to an optimizer that exploits an analytical model of the problem to identify the appropriate protocol configuration, as described in Sections 4.4 and 4.5. The model takes into account, in a hardware-independent fashion, some radio details including ramp-up and ramp-down times and the operation of the channel activity recognition procedure. We show that these low-level details, neglected by related work, are fundamental not only to ascertain the trade-off between latency and lifetime, but also to determine if a protocol configuration can be implemented in practice.

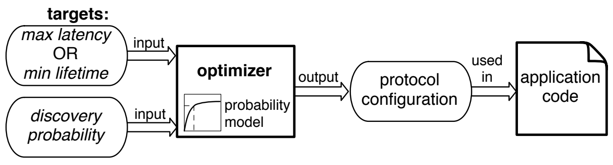


Figure 4.1: RUTh tool chain.

Although its formalization is mathematically complex, RUTh remains simple to understand — and to implement, as described in Section 4.6. Our goal is a neighbor discovery solution immediately usable in real-world deployments and on mainstream WSN platforms. Therefore, while the aforementioned model of RUTh is radio-independent, our implementation targets the popular CC2420 radio chip on Telos-like [97] hardware running TinyOS. Section 4.7 details our comprehensive analysis of parameters affecting our implementation and show how RUTh outperforms state-of-art protocols. We validate our analysis using simulation and experiments on real hardware. In Section 4.8, we show RUTh in the context of ACube, focusing on the integration with higher level routing protocols. Concluding remarks end the chapter in Section 4.9.

4.2 Related Work

In an energy-abundant system, a periodic broadcast targeting always-on receivers is a straightforward solution enabling neighbor discovery. The problem is similarly simple if nodes are time-synchronized, therefore exchanging discovery beacons at pre-determined times. Precise time synchronization, however, requires either dedicated protocols or external hardware (e.g., GPS). In this work, similarly to others mentioned below, we view neighbor discovery as a basic building block to be realized with minimal assumptions on the system. Therefore, our neighbor discovery relies only on wireless communication, and minimize power consumption by operating the radio with an efficient duty cycle.

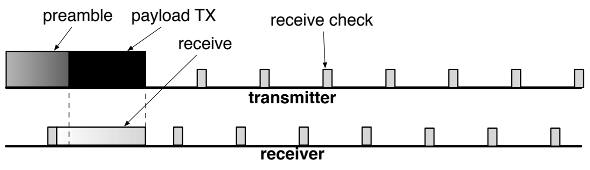
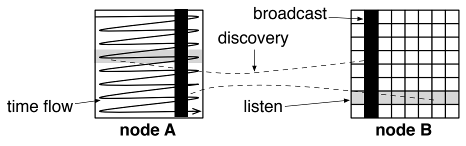


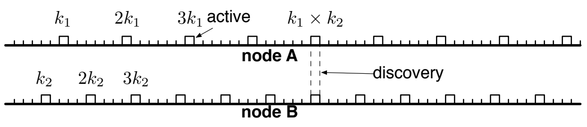
Figure 4.2: Low Power Listening.

Low-Power Listening. The reference duty cycle technique in WSNs, developed in the context of MAC protocols, is Low-Power Listening (LPL) [96]. As shown in Figure 4.2, in LPL mode a receiver periodically wakes up to perform a *receive check*, a procedure involving the Clear Channel Assessment (CCA) circuitry of the radio chip. This determines the presence of communication activity on the shared wireless medium. If activity is detected, the radio is kept on to enable subsequent packet receipt. This approach requires a transmitted packet to be preceded by a preamble, whose duration must span at least the entire period between two receive checks. The preamble guarantees that the receive check detects the transmission activity, enabling communication. Unlike MAC protocols, the focus of neighbor discovery is not on packet transmissions. Therefore, one can imagine a neighbor discovery scheme on top of LPL that relies on a periodic broadcast of preambles, used as beacons. Our work is indeed inspired by this very simple idea although, different from LPL, RUTh does not continuously probe the channel. This would unnecessarily waste resources especially in the absence of neighbors. In a sense, we use the same "building blocks" of LPL—preambles and receive checks—but *i)* compose them in a different way; *ii)* rely on a formalization of the problem to choose the optimal configuration meeting user-specified goals.

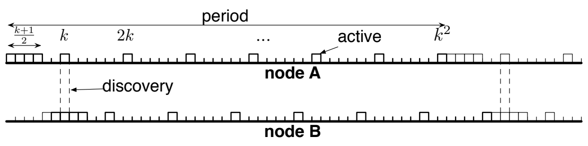
Neighbor discovery protocols. In contrast, state-of-the-art proposals all rely on some form of time-slotting where time is divided into periods of equal duration during which the node is either active (radio on) or inactive (radio off). For instance, in "birthday protocols" [85] a node chooses with a given probability whether a slot represents transmission, reception, or power-off time. Discovery occurs when a transmission slot of one node overlaps with a reception slot of the other. Birthday protocols solve neighbor discovery only probabilistically and, since the behavior in each slot is random, it is difficult to form an accurate energy model. This is instead possible in RUTh, thanks to its well-defined behavior. Moreover, alongside probabilistic discovery, RUTh can provide deterministic discovery with given latency bounds. The quorum-based protocol [116] also distinguishes between transmission and reception slots. Time is divided in sequences of m^2 contiguous slots, arranged in a $m \times m$ matrix. Each node randomly selects a column when it transmits, and a row when it listens. Detection is therefore guaranteed to occur within m^2



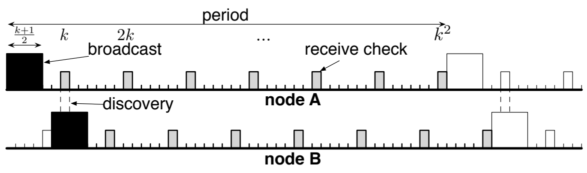
(a) Quorum.



(b) Disco.



(c) U-Connect in theory.



(d) U-Connect in practice.

Figure 4.3: Neighbor discovery protocols.

slots. Other approaches [32, 57, 131] do not make a functional distinction among slots, simply assuming that discovery occurs when the active slots of two nodes overlap. All these works determine the active slots using prime numbers. In Disco [32] each node is associated with a different prime, which determines the period between two active slots, as shown in Figure 4.3(b). Discovery occurs between two nodes at each common multiplier of their primes. Disco is asymmetric: nodes have different primes and thus different behaviors. Smaller primes lead to faster batter discharge complicating the estimate of lifetime.

U-Connect [57] is instead symmetric, i.e., the same prime k determines the period between active slots on all nodes, as shown in Figure 4.3(c). In addition, a train of $\frac{k+1}{2}$ contiguous active slots are scheduled every k^2 slots. The length of the train guarantees that there is at least one overlap of active slots during a period of k^2 slots. The U-Connect authors build upon the theoretical results in [131] to argue that, among all of the aforementioned protocols, U-Connect most closely approaches the optimal schedule. Moreover, the discovery latencies reported in their experimental section are significantly smaller w.r.t. the other systems. Consequently, in Section 4.7, we compare RUTh only against U-Connect. We show that RUTh is more efficient than U-Connect in terms of latency and lifetime, both in probabilistic and deterministic modes. Key to our superior performance are *i)* the many degrees of freedom our protocol allows w.r.t. the single, discrete parameter—the prime k—available for tuning U-Connect and, equally important, *ii)* the analytical model and optimization strategy that enables us to master this flexibility and easily determine the best RUTh configuration for the scenario at hand. In addition, in RUTh we considered a number of implementation details concerning the radio operation. These details, neglected by U-Connect, bear a strong impact on the configuration space and on the practical applicability of neighbor discovery protocols, as discussed next.

A Critical Look at the State of the Art. All of the above protocols assume that time is discrete and divided into slots. However, this leads to two problems:

- Slots are not aligned in reality. To ensure that transmissions overlap with receive checks, protocols often transmit for longer than nominally expected. For instance, in the actual LPL code of TinyOS 2.1, a "magic" 20 ms is added to the user-specified preamble duration. Similar workarounds increase power consumption well beyond what is expected by the associated models.

- In [32, 57, 131], for discovery to occur when active slots overlap, each node must both transmit and listen during its active slot. This fact is acknowledged by Disco, but neglected in the U-Connect *model*. In the *implementation*, however, this problem is worked around by broadcasting in the contiguous $\frac{k+1}{2}$ slots occurring with period k^2, and performing a receive check in all other active slots, as seen in Figure 4.3(d). This behavior can be regarded as pure LPL with periodic, shorter preambles and no actual data packet transmission.

U-Connect achieves superior performance by using very small slots of 250 µs, which coincide with the receive check duration. This creates two further implementation problems:

- In [57], the authors motivate their choice as follows: *"Although the minimum channel duration from the CC2420 datasheet is 128* µs*, we observed that for a reliable channel detection, 250* µs *is required."* As no further details are provided, we performed experiments to assess how the duration of the receive check affects its reliability. We performed the experiments on the CC2420, the radio used by both RUTh and U-Connect implementations. As further discussed in Section 4.6, we observed that a duration of about 5 ms already fails to recognize channel activity in 20% of the cases; a 250 µs receive check, as in U-Connect, fails in 80% of the cases. Our results are in line with common practice: the duration of the receive check in LPL (as reported in [100] and confirmed by our experiments) is about 10 ms, which indeed according to our measurements correctly recognizes channel activity in 100% of the cases.

- Transmissions and receive checks are preceded and followed by radio ramp-up and ramp-down phases whose durations are non-negligible—especially if one assumes a very short receive check as in U-Connect. To give an idea of the values at stake, Figure 4.4 shows the current drained over time by the aforementioned 10-ms receive check in LPL, as seen on a mote in series with a 10 Ω resistor. The radio ramp-up/down phases collectively account for ∼5 ms. These are neglected in U-Connect [57]. Nevertheless, radio ramp-up/down is important, as it introduces a

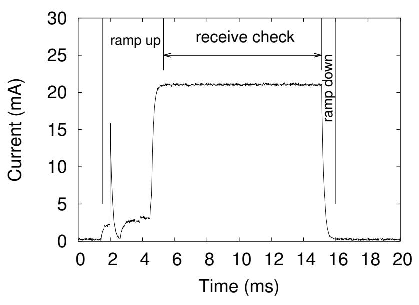


Figure 4.4: A receive check in LPL (and RUTh).

constraint on the minimum amount of time that must elapse between two consecutive radio activations. This in practice limits the set of configurations that are *implementable* for a given latency/lifetime goal.

Our model for RUTh explicitly considers the receive check duration and the radio ramp-up/down in a hardware-independent fashion, accounting for different radios. Capturing these low-level radio details, which directly affect the protocol configuration, yields latency/lifetime estimates reflecting more accurately the reality of implementation.

4.3 Overview and Assumptions

The operation of RUTh is very simple, yielding a lightweight implementation. Figure 4.5 shows a sample execution we use for illustration and Table 4.1 summarizes our notation. Discovery relies on beacons and receive checks, widely used in MAC protocols, as mentioned earlier. For all nodes, time is divided in *epochs* of equal duration T. Nodes are not synchronized: we assume a phase $\phi \in [0, T)$, i.e., the delay of node B's schedule w.r.t. A's, with the latter assumed as a time reference; intuitively, any epoch i begins at node B with a delay ϕ w.r.t. the same epoch i begun at A.

A *contact* occurs when a node enters the radio range of another. We assume this happens at $\gamma \in [0, T)$ w.r.t. A's time origin. This is the first moment when communication between A and B becomes possible. However, the two nodes in contact are oblivious of each other until discovery occurs; the difference between the contact and discovery times, hereafter referred to as *discovery latency*. Our goal is to ensure that the discovery latency is always smaller than a given maximum discovery latency L.

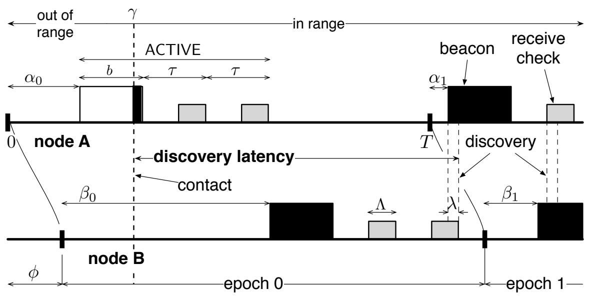


Figure 4.5: A sample execution of RUTh.

In each epoch i, every node runs the following sequence:

- picks a time (α_i for node A, β_i for node B) within the epoch, at which it transmits a beacon of duration b. After the beacon, the radio is switched off;
- performs s receive checks, each with duration Λ and repeated with a period τ. In between receive checks the radio is switched off;
- switches off the radio until its next beacon.

For discovery to happen, there must be a minimum overlap λ between the beacon of the discovered node and the receive check of the discoverer. The duration λ is a hardware constraint; it is lower-bounded by the amount of time it takes the CCA circuitry to sample the channel and reliably detect activity, i.e., 128 µs according to [24]. The upper bound is instead the entire duration Λ of the receive check, $\lambda \leq \Lambda$. We return to the relation between λ and Λ at the end of this section.

We choose the beacon duration $b = \tau + \lambda$, equal to the period τ between receive checks, augmented by the minimal overlap λ necessary for detection. This choice guarantees that, if a transmitter's beacon is sent during the receiver's sequence of s receive checks, the beacon is correctly detected, does not fall in the gap between receive checks, and there is an overlap of at least λ with one of two consecutive receive checks of the receiver.

Moreover, the interval τ between two consecutive receive checks cannot be arbitrarily small due to physical constraints of the radio. As mentioned in Section 4.2, the radio ramp up/down is not instantaneous: if τ is too small, the radio cannot be turned off to save energy. We therefore assume that a lower bound $\tau > \tau_{\min}$ is defined in practice.

From the protocol description above, it is clear that each node alternates an interval when it is *active* with another when it is inactive and the radio is switched off. The active interval $\textsc{active} = b + s\tau$, is constituted by the transmission of the beacon followed by the sequence of s receive checks, and is when discovery may occur. We require an active interval to be fully contained within an epoch, that is, with reference to Figure 4.5, $\alpha_i, \beta_i \in [0, T - \textsc{active})$. The radio is actually on only for a fraction of the active interval $\textsc{on} = b + s\Lambda < \textsc{active}$, yielding a duty cycle $\textsc{on}/T$. $\textsc{active}$ is a fundamental parameter of RUTh, and deeply affects its operation by determining whether it provides *deterministic* or *probabilistic* guarantees, as we discuss next.

Deterministic discovery. If the active interval is slightly longer than half of the epoch (i.e., $\textsc{active} \geq T/2 + \lambda$) and on both nodes A and B the active interval begins with the epoch (i.e,. $\forall i, \alpha_i = \beta_i = 0$) then discovery is *guaranteed* to occur within an epoch duration T, regardless of the value of the phase ϕ. Indeed, as illustrated in Figure 4.6, since $\textsc{active}$ covers more than half of the epoch, *i)* the active intervals of the two nodes are guaranteed to overlap during either node's epoch; *ii)* the constraint $b = \tau + \lambda$, tying the beacon duration to the spacing among receive checks, guarantees discovery.

Symbol	Description	Source
L	Maximum discovery latency	Application
$p_{\min}$	Discovery probability at time L	
T	Duration of an epoch	Optimizer output
τ	Period of the receive check sequence	
s	Number of receive checks during an active interval	
γ	Time of contact	Random variable
ϕ	Phase between two nodes A and B	
Λ	Duration of a receive check	Hardware constraint
λ	Minimum overlap between beacon and receive check	
b	Duration of a beacon	$b \triangleq \tau + \lambda$
$\textsc{active}$	Duration of active interval	$\textsc{active} \triangleq b + s \cdot \tau$
$\textsc{on}$	Time during which the radio is on within $\textsc{active}$	$\textsc{on} \triangleq b + s \cdot \Lambda$
$Q(\tau, s, T)$	The drain of electrical charge during one epoch	Equation 4.1
α_i, β_i	The start of an active interval in epoch i of node A, respectively B. $\alpha_i = \beta_i = 0$ for deterministic and random in $[T - \textsc{active})$ for probabilistic	
$A_i \leftarrow B_j$	The event of A receiving, in its i-th epoch, a beacon from B's j-th epoch	
$\mathcal{C}(i)$	The event of detecting a contact in the epoch i of A	

Table 4.1: Summary of notation.

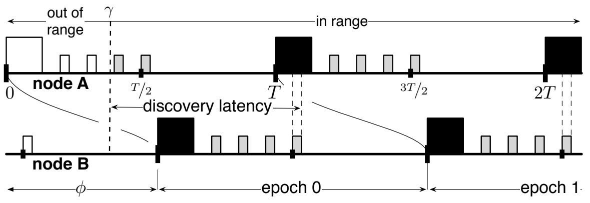


Figure 4.6: Deterministic discovery in RUTh.

If ϕ is such that the beacons of the two nodes overlap, a collision occurs. The problem is solved as in any CSMA/CA protocol, including LPL and U-Connect, by performing a CCA before transmission. This is easily achieved by enabling the appropriate hardware feature. Its (negligible) effects in terms of energy and timing can be accounted for by properly increasing the duration of the beacon b.

Probabilistic discovery. If $\text{ACTIVE} < T/2 + \lambda$ and the active interval begins at a random time in the epoch (i.e., $\forall i,\ \alpha_i, \beta_i \in [0; T - \text{ACTIVE})$ are independent random variables), energy can be spared, but we cannot guarantee that all contacts are detected by a determined latency. Nevertheless, a probabilistic bound is possible and acceptable in several applications, as we argued in Section 4.1. For instance, in our wildlife application, zoologists are satisfied with a discovery latency of 5 s for 80% of the contacts. Incidentally, this does not mean that the remaining 20% are never detected, rather their detection occurs beyond 5 s. In our application, the small impact on the quality of the retrieved data is overcome by the gain in lifetime.

The above reasoning holds as long as the application developer is provided with principled tools for configuring RUTh such that the required latency, lifetime, and probability targets are met. The formal framework and optimization tools to achieve this configuration, one of the main contributions of this chapter, are described in Sections 4.4 and 4.5.

Energy expenditure. To determine the optimal configuration, we must derive an estimate of the energy spent by a node during each epoch. In doing this, we neglect the energy drain by components other than the radio, such as CPU and sensors. Although these aspects can be taken into account when determining the energy budget [65], their contribution is dominated by the specifics of the application, while in this work we focus solely on the application-independent neighbor discovery functionality. We adopt an energy model for the radio similar to [96]. We assume that the radio consumes a current I_{tx} when transmitting, I_{rx} when receiving, and I_{off} when switched off. Moreover, we assume that the radio ramp-up and ramp-down have collectively a duration ρ, and consume an average current I_{ramp}. The drain of electrical charge during one epoch can then be estimated as:

$$Q(\tau, s, T) \triangleq bI_{tx} + s\Lambda I_{rx} + (s+1)\rho I_{ramp} + [T - \text{ON}]\, I_{off} \tag{4.1}$$

This formulation accounts for different types of radios. For instance, radios where the transmit power is higher than the receive power (e.g., the popular CC1000) can be easily accommodated by properly defining the values of I_{tx} and I_{rx}.

Receive check Λ and minimum overlap λ. Modeling explicitly the distinction between the receive check Λ and its minimal overlap λ with a beacon allows us to take into account more precisely their contribution, yielding more accurate estimates. Λ has a direct impact on energy expenditure, and thus lifetime, as shown in Equation (4.1) and by the duty cycle $\text{ON} = b + s\Lambda$. Instead, λ plays a key role in defining the discovery probability. λ represents the worst-case situation where a beacon “barely” overlaps with

```
  input  : $\tau_{\min}$, $L$
  output: a protocol configuration $\langle \tau, s, T \rangle$
  output: the (minimal) current consumption $Q$ per epoch
1 $T \leftarrow L$; $Q_{\min} \leftarrow \infty$
2 for $s \leftarrow 1$ to $\left\lfloor \frac{T}{\tau_{\min}} \right\rfloor$ do
3     $\tau \leftarrow \frac{T}{2(s+1)}$
4     if $\tau \leq \tau_{\min}$ then $\tau \leftarrow \tau_{\min}$
5     if $\tau (s+1) + \lambda > T$ then continue
6     if $Q(\tau, s, T) < Q_{\min}$ then $Q_{\min} \leftarrow Q(\tau, s, T)$
```

Figure 4.7: Latency-driven deterministic discovery.

a receive check. However, $b = \tau + \lambda$ ensures that a beacon overlapping with at least two consecutive receive checks (spaced by $\tau - \Lambda$) is detected reliably. Instead, if the beacon overlaps only with the last receive check of ACTIVE, the possibility of detection is determined solely by the extent of the overlap: detection is possible only if the overlap is at least λ. This explains why, although the receive check Λ is the radio parameter we can control directly, in our treatment of discovery probability it is only $\lambda \leq \Lambda$ that truly matters.

4.4 Latency-driven Discovery

Our objective is to find a proper RUTh configuration $\langle \tau, s, T \rangle$ such that battery discharge is minimal and neighbors are discovered with a maximum latency L. Hereafter we consider Λ and λ as configuration constants. First we address deterministic discovery where, as mentioned in Section 4.3, discovery is guaranteed to occur within one epoch and therefore $T = L$. Next, we turn to probabilistic discovery, and determine the probability of discovery for a given latency L.

4.4.1 Deterministic Discovery

We saw in Section 4.3 that the constraint ACTIVE $\geq T/2 + \lambda$ guarantees discovery within one epoch. Our goal is to minimize energy consumption for this deterministic discovery, based on Equation (4.1) and the protocol operation. By recalling $T = L$, we can state our optimization problem as follows:

$$\begin{aligned} \text{minimize:} \quad & Q(\tau, s, T) \\ \text{subject to:} \quad & \text{ACTIVE} \geq T/2 + \lambda \\ & \tau \geq \tau_{\min} \end{aligned}$$

Our solution is implemented by the algorithm in Figure 4.7. We observe a strong connection between s and τ. Consider that $s \geq 1$: nodes must perform at least one receive check per epoch to enable discovery. The maximum value is instead achieved for $\tau = \tau_{\min}$, yielding $\left\lfloor \frac{L}{\tau_{\min}} \right\rfloor$ as the upper bound for s. This sufficiently constrains the problem, allowing us to solve it by exhaustive search (lines 2–6 of Figure 4.7). The solution space is small for practical configurations, e.g., $L = 10$ s and $\tau_{\min} = 10$ ms yield only 1000 possible values for s.

Second, for each value of s, there is a unique value of τ minimizing $Q(\tau, s, T)$. To understand why note that, for a fixed s, energy consumption is determined by the only "free" parameter, the beacon duration $b = \tau + \lambda$. Also, given that $I_{\mathit{off}} < I_{tx}$ in Equation (4.1), Q is monotonically increasing with τ or, in other words, for a given value of s, the smallest discharge Q is always given by the smallest value of τ. Consequently, for each s we choose the smallest value of τ that complies with the two constraints of the optimization problem above, recalling that $\textsc{active} = b + \tau s = \tau(s+1) + \lambda$ (line 3–4). We skip those pairs (τ, s) that yield an active interval longer than one epoch T (line 5).

4.4.2 Probabilistic Discovery

If $\textsc{active} < T/2 + \lambda$, discovery is not guaranteed to occur within an epoch. This achieves energy savings by reducing the duty cycle, but at the expense of decreasing the probability to detect a contact within the target latency. This section lays the formal foundation that allows us to determine the protocol configuration that provides maximal energy savings while providing a lower bound for the discovery probability.

Next, we formalize this as an optimization problem, show how to solve it and illustrate the salient aspects of an analytical model that allows us to compute the discovery probability.

Finding the Optimal Configuration

When running in probabilistic discovery mode, during each i-th epoch every node picks a random time for the beginning of its active interval. We assume that these times, α_i and β_i in Figure 4.5, are independent random variables uniformly distributed on $[iT, (i+1)T - \textsc{active}]$. Let $\mathcal{C}(i)$ denote the event "a contact is discovered during epoch i". Since an epoch is in general not enough to guarantee detection, we must consider the overall probability $P\left[\bigvee_{i=0}^{n} \mathcal{C}(i)\right]$ (discussed in Section 13) to detect *at least one* contact during a sequence of n consecutive epochs. We assume that the maximum discovery latency is a multiplier of the epoch duration, $L = nT$.

The problem reduces to finding a protocol configuration $\langle \tau, s, T \rangle$ that guarantees discovery within $L = nT$ with probability $P\left[\bigvee_{i=0}^{n} \mathcal{C}(i)\right] \geq p_{\min}$ and minimizes energy consumption. Recalling that $\textsc{active} < T/2 + \lambda$ achieves probabilistic mode, we focus on solving the optimization problem:

$$\begin{aligned}
\text{minimize:} \quad & Q(\tau, s, T) \\
\text{subject to:} \quad & \textsc{active} < T/2 + \lambda \\
& \tau \geq \tau_{\min} \\
& P\left[\textstyle\bigvee_{i=0}^{n} \mathcal{C}(i)\right] \geq p_{\min} \\
& L = nT \text{ where } n \in \mathbb{N}
\end{aligned}$$

An exhaustive search of the solution space is feasible also in this case, using the solver algorithm in Figure 4.8 and based on the following observations. First, we impose a

```
input  : $\tau_{\min}$, $p_{\min}$, $L$
input  : the desired precision $\epsilon$ over the value of $\tau$
output: a protocol configuration $\langle \tau, s, T \rangle$
output: the (minimal) current consumption $Q$ per epoch
1  $Q_{\min} \leftarrow \infty$
2  for $n \leftarrow 1$ to $\left\lfloor \frac{L}{4\tau_{\min}} \right\rfloor$ do
3      $T \leftarrow L/n$
4      for $s \leftarrow 1$ to $\left\lfloor \frac{T}{2\tau_{\min}} \right\rfloor - 1$ do
5          $\tau_l \leftarrow \tau_{\min}$; $\tau_h \leftarrow \frac{T}{2(s+1)}$; $\tau' \leftarrow \infty$; $\tau \leftarrow \tau_h$
6          if $P\left[\bigvee_{i=0}^{n} \mathcal{C}(i)\right] < p_{\min}$ then continue
7          while $|\tau' - \tau| > \epsilon$ do
8              $\tau' \leftarrow \tau$; $\tau \leftarrow \frac{\tau_l - \tau_h}{2}$
9              if $P\left[\bigvee_{i=0}^{n} \mathcal{C}(i)\right] < p_{\min}$ then
10                 $\tau_l \leftarrow \tau$
11             else
12                 $\tau_h \leftarrow \tau$
13                 if $Q(\tau, s, T) < Q_{\min}$ then $Q_{\min} \leftarrow Q(\tau, s, T)$
```

Figure 4.8: Latency-driven probabilistic discovery.

bound on the smallest possible epoch T by observing that this corresponds to the case when ACTIVE is minimal, and subject to the first constraint. The active interval is minimal when $\tau = \tau_{\min}$ and $s = 1$, i.e., the node performs only one receive check. In this case, ACTIVE $= 2\tau + \lambda$, thus $T > 4\tau_{\min}$ must hold. Therefore, n can take integer values up to $\frac{L}{4\tau_{\min}}$ (line 2 of the algorithm). For a given n, T is easily determined from the constraint $L = nT$. To satisfy the constraint ACTIVE $< T/2 + \lambda$, we derive a maximal value for s based on the equivalence ACTIVE $= \tau(s+1) + \lambda$ (line 4). Given n and s, we need to find the proper value for τ. The lower bound is given directly by $\tau_{\min}$, while the upper bound can be derived again from the definition of ACTIVE and the constraint on its duration, $\tau = \frac{T}{2(s+1)}$. Since both discovery probability, discussed next, and energy consumption are monotonically increasing with τ, we can perform a binary search for the value of τ that minimizes energy consumption (lines 5–13).

Computing the Discovery Probability

The algorithm in Figure 4.8 relies on knowledge of the discovery probability $P\left[\bigvee_{i=0}^{n} \mathcal{C}(i)\right]$ for a given RUTh configuration. Computing this probability is complicated by the fact that the same contact may be detected multiple times across several epochs, as shown in Figure 4.9. We refer to this as an *x-discovery chain*, where x is the number of consecutive discoveries ($x = 6$ in Figure 4.9). Intuitively, the x-discovery chain results from the recursive application of the formula $P[A \vee B] = P[A] + P[B] - P[A \wedge B]$, where A and B are random events, on the epochs constituting the chain. Specifically, the chain corresponds to the joint probability $P[A \wedge B]$ in the previous formula. Schedules like this must be taken into account, to avoid incorrectly overestimating the discovery probability.

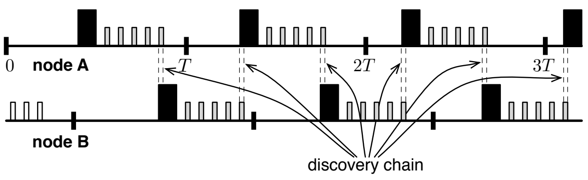


Figure 4.9: Schedule for a 6-discovery chain.

To compute the probability of a discovery chain, we make the simplifying assumption that the contact occurs at time $\gamma = 0$. With this assumption, the analytical formulation is already quite complex: without it, it would become essentially intractable analytically. The effects of this assumption on the accuracy of our method are analyzed in Section 4.7. We determine the analytical expression of the discovery probability as follows:

1. we show that the length of a discovery chain is finite;
2. we use the previous result to express the discovery probability at the end of epoch n as a recursive function;
3. the coefficients in the above function are basic probabilities such as $P\left[A_n \leftarrow B_n\right]$. We determine their probability density function (pdf) and then use the relation $P\left[X \in D\right] = \int_D f\left(x\right)\, \mathrm{d}x$, where f is the pdf of X, to switch to probabilities. To embed this analytical process in our solver, we rely on the numerical integration tools in [39].

We skip the rather tedious details of the three steps above. However, the full machinery, including proofs of the aforementioned lemmas and theorems, can be found in Appendix A.

4.5 Lifetime-driven Discovery

We now turn our attention to the problem of finding a configuration $\langle \tau, s, T \rangle$ of our protocol such that a target expected lifetime E is met by minimizing the maximal discovery latency L. The solution we provide here builds on the solvers described in Section 4.4: the one to use depends on whether we are solving the lifetime-driven discovery problem deterministically or probabilistically.

In our solution, we assume that each node is equipped with a battery of capacity $\mathcal{B}$. Then, the maximum average current a node is allowed to drain to meet the lifetime expectancy is defined by the fraction ${}^{\mathcal{B}}/_{E}$. A constraint of our problem is therefore that the battery discharge ${}^{Q}/_{T}$ during a single epoch cannot be higher than this value. Moreover, although our objective is to minimize the maximum discovery latency L, in practice the latter cannot assume arbitrarily large values: end-users will set an upper bound $L_{\max}$, acceptable for their application. In the deterministic case, the optimization problem can be formulated as:

$$\begin{aligned} \text{minimize:} \quad & L \\ \text{subject to:} \quad & \textsc{active} \geq {}^{T}/_{2} + \lambda \\ & \tau \geq \tau_{\min} \\ & {}^{Q}/_{T} \leq {}^{\mathcal{B}}/_{E} \\ & L \leq L_{\max} \end{aligned}$$

```
input  : $\tau_{\min}$, $\mathcal{B}$ and $E$
input  : upper bound $L_{\max}$ for maximum discovery latency $L$
input  : the desired precision $\epsilon$ over the value of $L$
output: a protocol configuration $\langle \tau, s, T \rangle$
output: a (minimized) maximum discovery latency
uses   : LatencySolver($L$), returns the smallest current consumption for a given
         maximum discovery latency $L$; in practice, it is either of the solvers in Figure 4.7
         or 4.8
1 $L_l \leftarrow 4\tau_{\min}$; $L_h \leftarrow L_{\max}$; $L' \leftarrow \infty$; $L \leftarrow L_h$
2 if LatencySolver($L$) $> T \cdot \frac{\mathcal{B}}{E}$ then return NIL
3 while $|L' - L| > \epsilon$ do
4     $L' \leftarrow L$; $L \leftarrow \frac{L_h - L_l}{2}$
5     if LatencySolver($L$) $> T \cdot \frac{\mathcal{B}}{E}$ then
6         $L_l \leftarrow L$
7     else
8         $L_h \leftarrow L$
```

Figure 4.10: Lifetime-driven discovery.

The formulation of the probabilistic case is identical, except for the first constraint on ACTIVE. The solver, in Figure 4.10, is the same for both cases, as it builds on previous results.

First, we define the domain of the latency L. As already seen in Section 4.4.2, the lower bound for T (and thus L) is $L_l = 4\tau_{\min}$, while the user-specified upper bound is $L_h = L_{\max}$. The case where no protocol configuration complies with the lifetime and latency goals is considered (line 2). To find the minimal L meeting the lifetime goal, we observe that a higher amount of energy is spent when targeting a smaller latency. Hence, we narrow down the $[L_l, L_h]$ interval by halving it until we reach a desired precision ϵ (line 3). At each step (line 5), we use either of the solvers for the latency-driven problem in Figure 4.7 and 4.8, to achieve a deterministic or probabilistic solution, respectively. These find the minimum energy spent for a maximum discovery latency $L = {}^{L_h - L_l}/_2$. If the energy spent for L complies with the constraint on average battery discharge, we continue by looking for an even smaller compliant L in the first half of $[L_l, L_h]$ (line 6), otherwise resume the search in the other half (line 8). This algorithm converges to the solution with logarithmic behavior, as it performs a binary search for the smallest value of L complying with the constraints.

4.6 Implementation

We implemented RUTh as a neighbor discovery service integrated with the CC2420 stack of TinyOS 2.1. Applications control the service behavior through the control interface `NeighborDiscovery` in Figure 4.11. Protocol configuration is achieved by supplying the `start` command with the values for T, b, s obtained from one of the solvers presented in the previous sections. Contact is signaled through an event that includes the identifier

```
interface NeighborDiscovery {
  command void start(uint32_t epochDuration, uint16_t beaconDuration,
                     uint16_t samples);
  command void stop();
  event void detected(am_addr_t neighbor);
}
```

Figure 4.11: The RUTh control interface.

of the discovered neighbor.

Internally, RUTh employs a timer that both triggers the periodic broadcast of the beacon and starts the LPL service to execute a number of receive checks, according to the RUTh configuration. The beacon is a packet of minimal size (i.e., it consists only of the 802.15.4 header [2]) repeatedly broadcast for a duration b. To distinguish beacons from application packets, the former carry a different PAN identifier. With the CC2420, as also noted in [90], strobing the beacon packet does not result in a continuous stream of activity on the channel, and leaves instead small gaps between packets[1]. These gaps are inherent in the hardware design: after a transmission, the CC2420 must undergo several state transitions before being able to transmit again [24]. To determine the gaps' duration, we build with an oscilloscope the current profile of a Telosb when sending a $b = 20$ ms beacon (Figure 4.12). This measurement shows that each gap lasts approximately 480 µs and a beacon transmission lasts about 720 µs. The duration Λ of a receive check should be at least greater than the inter-packet gap. Radios that, unlike the CC2420, are not packed-based do not suffer from the same limitation and can in principle employ shorter receive checks.

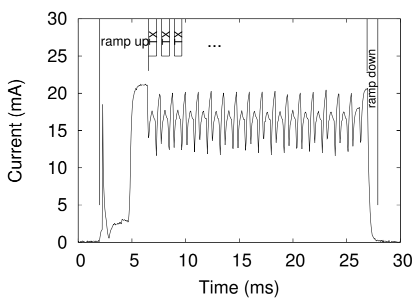


Figure 4.12: Current profile of a $b =$ 20 ms RUTh beacon.

To receive beacons, RUTh optimizes the receive check procedure inside the LPL stack, long enough to account for the gaps in the beacon. The behavior is illustrated in Figure 4.4. During this procedure, LPL repeatedly polls (400 times) a flag set by an interrupt handler to indicate that a *raw* signal has been detected by the CCA circuitry on the CC2420 radio. Channel activity is identified when the flag check returns true $t = 4$ times, which reduces the possibility of falsely detecting radio activity [90]. If a full packet is received during the poll loop, this is properly notified on a separate pin and serves as proof of channel activity. Our measurements indicate that this procedure takes $\Lambda = 10.04$ ms, excluding radio ramp-up/down, which collectively take ~5 ms. If activity is recognized, which in practice could also be channel noise, the radio is kept on for 100 ms to receive the actual data. The number t of required flag checks returning true is positively correlated with λ, the minimum overlap between a beacon and a receive check. The higher t is, the longer a beacon $b = \tau + \lambda$ must be—hence the "magic" extra 20 ms of LPL beacons in

[1]Although [24] describes a streaming mode, it is labeled as *for lab testing only*, hence we do not consider it further.

the TinyOS stack. To keep beacons as small as possible, in RUTh we use $t = 1$, at the risk of obtaining more false positives in the receive check. To mitigate their effect on the duty cycle, we keep the radio on for only 10 ms when activity is (supposedly) recognized, rather than 100 ms, as LPL does. For practical purposes, in our optimizer, we attribute $t = 1$ to a duration of $\lambda = 1$ ms, the rough duration of a packet transmission and an inter-packet gap.

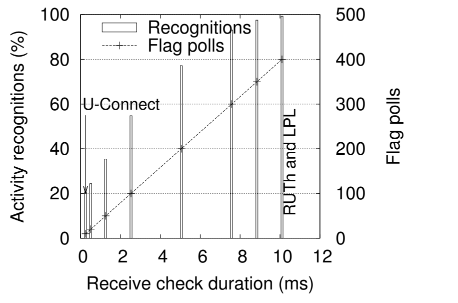


Figure 4.13: Reliability of channel activity recognition vs. receive check duration.

During the development of RUTh, we also attempted to reduce the duration Λ of the receive check to lower energy consumption. However, shortening the receive check negatively affects channel activity recognition. We observed this phenomenon in experiments with two nodes deployed 1 m apart, one programmed to broadcast continuously the same packet at the commonly-used power level of -1 dBm, and the other programmed to perform a receive check every 200 ms. The ability to correctly recognize channel activity was measured, for each data point, as the percentage of successful receive checks over a total 10,000. As shown in Figure 4.13, we used various settings for the number of CCA flag polls, ranging from 10 to 400, yielding the receive check durations (represented on the x-axis) which range from a minimum of ~250 μs (as in U-Connect) to ~10 ms (as in LPL). The number t of checks returning true required to recognize activity was set to 1 for each experiment. The results in Figure 4.13 show that the receive check duration used by LPL, and adopted in RUTh, always guarantees a correct activity recognition. Lower values fail to recognize activity: in particular, the 250 μs receive check succeeds in only in 20% of the cases.

4.7 Evaluation

In our evaluation, we first use the analytical model to study the behavior of RUTh when given different latency or lifetime targets, and to compare against U-Connect. The accuracy of our model w.r.t. the assumption that contact occurs at $\gamma = 0$ is evaluated using a simulator. Then, we assess how RUTh behaves on real Telos motes. Performing an in-field evaluation with mobile nodes is however difficult: obtaining ground truth requires localization devices (e.g., GPS) and is complicated by the vagaries of wireless communication. Indeed, none of the works in Section 4.2 is evaluated with mobile nodes, and all resort to a controlled environment with static nodes. At first, we follow the same route and perform a suite of micro-benchmarks on fixed nodes that confirm the simulation results. Then, we report on data from preliminary experimentation on mobile motes employed as social contact sensors in a less controlled environment. Finally, we share the lessons learned from a deployment of RUTh used as a proximity sensor.

4.7.1 Analytical Study

We use the analytical model to study the behavior of RUTh when given either latency or lifetime requirements. We investigate deterministic discovery, along with probabilistic discovery with values of $p_{\min}$ ranging from 0.5 to 0.9.

Parameters and Assumptions. Although our model does not depend on a specific radio, here we use the electrical characteristics of the CC2420 radio from [24], shown in Table 4.2, as this radio is used in [32, 57] and is arguably the most popular in WSN deployments. We assume currents are constant. For simplicity, we assume a battery with an initial capacity of 2 Ah and uniform discharge throughout its lifetime.

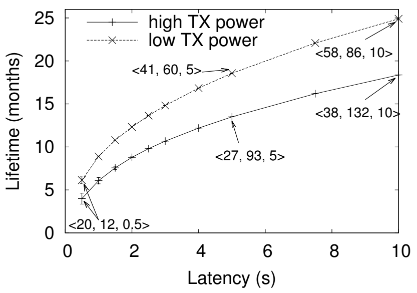
Here, we use $\Lambda = \lambda = 250$ µs, the same as in U-Connect. Although we pointed out a number of implementation issues with this value for Λ, this choice makes our results directly comparable to [57]. Section 4.7.3 provides an empirical evaluation with the actual values used in our implementation.

Radio ramp-up/down times place constraints on protocol behavior and energy consumption. These are neglected by U-Connect, whose model assumes that switching the radio on/off is instantaneous and consumes zero energy. As the CC2420, like other radios, exhibits non-negligible ramp-up/down times, we impose these constraints on U-Connect when we perform our comparison. The most serious consequence of this choice, dictated by the desire of evaluating *both* protocols against the reality of implementation, is that adjacent receive checks must have a minimal space between them. In other words, there must be enough time to allow the radio to actually switch off then reactivate. Our analysis shows only U-Connect configurations that comply with these spacing constraints. In RUTh, the spacing among receive checks is determined by $\tau_{\min}$, set to 5, 10, or 20 ms in our analysis. In light of the measurements in Figure 4.4 showing that ramp-up/down alone takes ~5 ms, $\tau_{\min} = 5$ ms is actually insufficient, and we show it only to provide a lower bound. In practice, $\tau_{\min} = 20$ ms is desirable to provide sufficient slack for hardware delays that affect timer accuracy. Finally, regarding energy consumption, although our model incorporates these phases through I_{ramp}, for a fair comparison, we set the current drain to I_{off} during ramp-up/down.

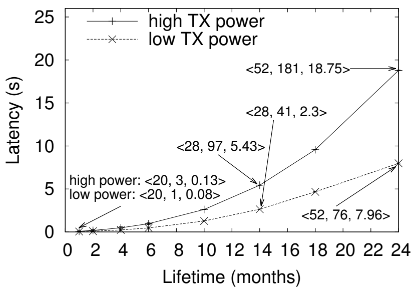
Application requirements determine the acceptable physical distance at which to detect contact. For example, distances are small in our indoor assisted living scenario, while in the outdoor wildlife scenario, relevant social interactions occur at larger distances, especially for solitary animals. Tuning our discovery protocol for this constraint corresponds to selecting the appropriate transmit power of the CC2420 and providing the corresponding radio consumption values to the analytical model. Here we show results for common

Symbol	Description	Value
I_{tx}	Transmission current	17.4 mA (high power, −1 dBm) 8.5 mA (low power, −25 dBm)
I_{rx}	Receive current	19.7 mA
I_{off}	Radio off current	0.02 mA
$\mathcal{B}$	Battery capacity	2 Ah

Table 4.2: Electrical characteristics.



(a) Latency-driven: expected lifetime.



(b) Lifetime-driven: achieved latencies.

Figure 4.14: Deterministic behavior: sample protocol configurations $\langle \tau(\mathrm{ms}), s, T(\mathrm{s}) \rangle$ are shown for $\tau_{\min} = 20$ ms.

values of high and low transmission power.

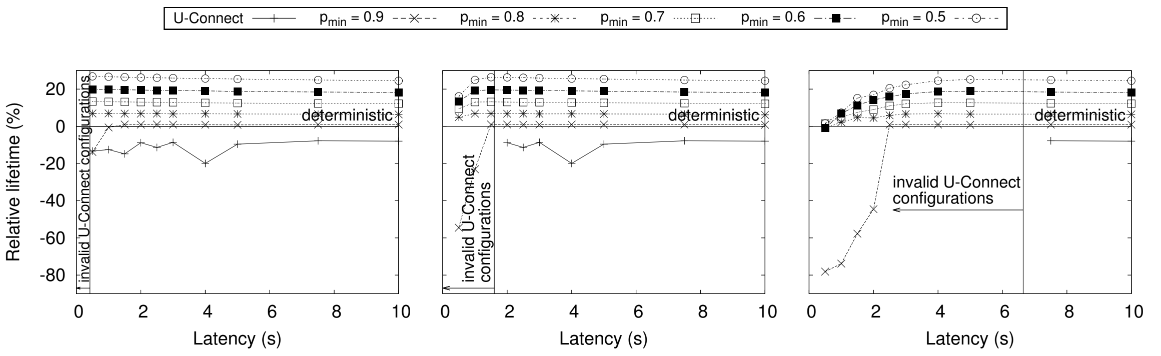
Latency-driven discovery. Our goal is to determine the RUTh configuration that offers the minimum energy consumption for a maximum, user-specified discovery latency. Hence, we feed our optimizer latencies from 0.5 to 10 s and the aforementioned values for $\tau_{\min}$ and transmission power.

For deterministic discovery with RUTh, the estimated lifetime is depicted in Figure 4.14(a) together with several protocol configurations $\langle \tau, s, T \rangle$ deemed optimal by our solver for $\tau_{\min} = 20$ ms. The chart shows that, as expected, low transmission power and higher input latencies yield longer lifetimes. The two curves are averages of the values $\tau_{\min} \in \{5, 10, 20\}$ ms, as this parameter has little impact on the energy consumption. Deviation is marked by error bars, with a maximum of 18% for high transmission power and a 0.5 s desired latency. For target latencies above 2 s, deviation drops to less than 1%. Indeed, this is expected: as the beacon time b (approximately equal to τ) remains constant, the period of the protocol shrinks and thus the duty cycle is increased.

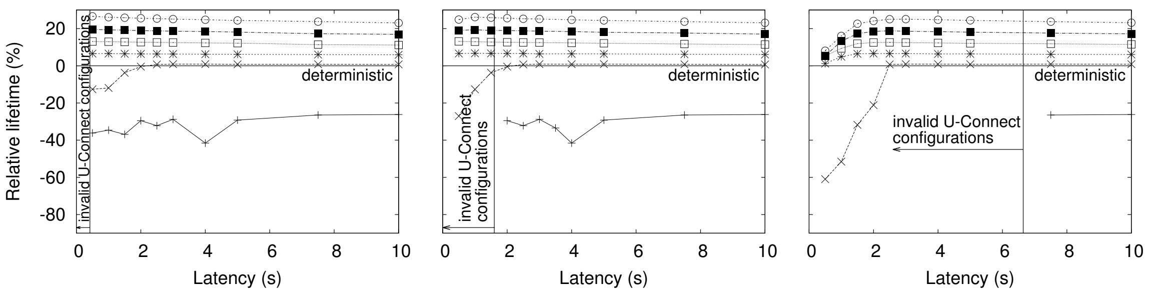
We next use the expected lifetime just obtained for the deterministic discovery behavior as a reference point for comparing against U-Connect as well as the probabilistic version of RUTh. Specifically, we plot $^{R_D - X}/_{R_D}$, where R_D are the values of the deterministic RUTh protocol and X is either U-Connect or the probabilistic version of RUTh. In the resulting charts of Figure 4.15, negative values mean that nodes will deplete their batteries faster than the deterministic case. Note that U-Connect always has higher energy demands, while reducing the discovery probability offers energy savings.

Our charts also show that certain low latencies cannot be met by U-Connect. This stems from the radio ramp-up/down constraint outlined earlier, which forces an interval $\tau_{\min}$ between consecutive radio activities. Recall from Section 4.2 that the behavior of U-Connect is dictated by a prime number k: all protocol periods of duration $k^2 \cdot 250$ µs start with a beacon of $^{k+1}/_2 \cdot 250$ µs and then every $k \cdot 250$ µs a receive check is performed. Between the end of the beacon and the first receive check, there is a period of $^{k-1}/_2 \cdot 250$ µs the radio spends in power-off mode. As an example, to guarantee a discovery latency of 1 s, k should be set to 61, and thus the first energy saving duration should last for 7.5 ms. Such a short interval violates a $\tau_{\min}$ constraint of 10 ms. In other words, for gaps less than 10 ms, the radio would not be able to turn off, and therefore the U-Connect configuration is considered invalid.

U-Connect's use of prime numbers also prevents it from adapting to the beacon transmission power. As the configuration samples in Figure 4.14(a) indicate, at low power RUTh achieves energy savings by reducing the number of samples s and increasing the



(a) High power, $\tau_{\text{min}} = 5$ ms (b) High power, $\tau_{\text{min}} = 10$ ms (c) High power, $\tau_{\text{min}} = 20$ ms



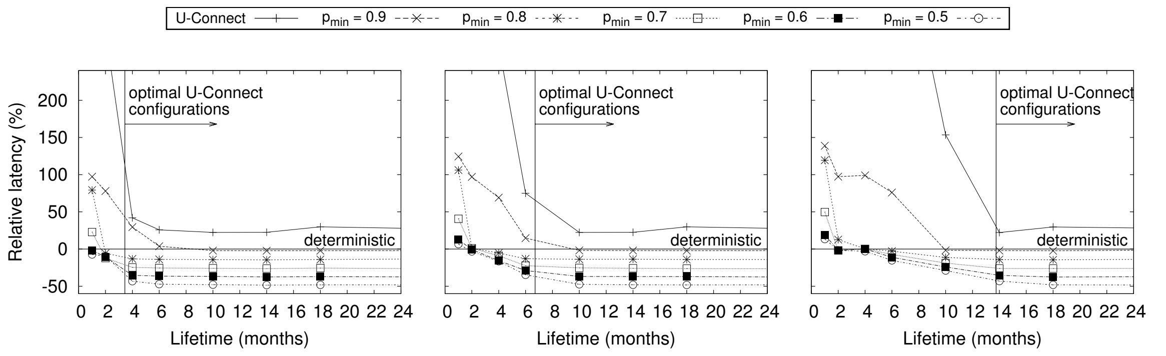
(d) Low power, $\tau_{\text{min}} = 5$ ms (e) Low power, $\tau_{\text{min}} = 10$ ms (f) Low power, $\tau_{\text{min}} = 20$ ms

Figure 4.15: Latency-driven: relative expected lifetime for U-Connect and various RUTh configurations.

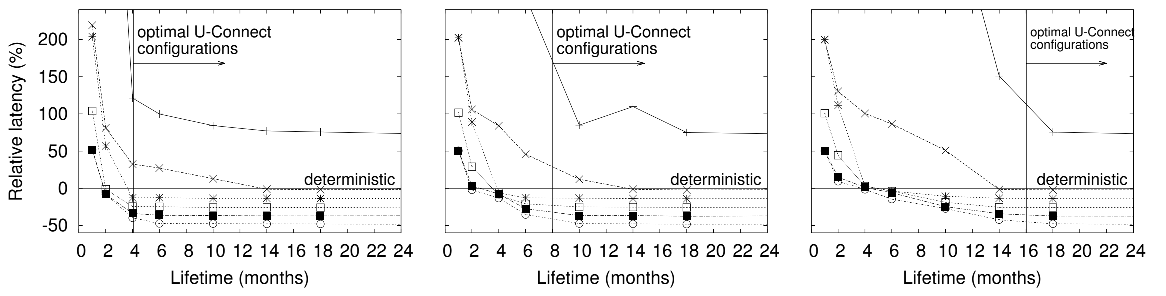
beacon length b in such a way that ACTIVE remains constant. Instead, in U-Connect the ratio $^b/_T$ is fixed to $^{k+1}/_{2k^2}$, thus the beacon length cannot be tuned. The result is visible by comparing the two rows of Figure 4.15: for high power, the benefit over U-Connect is 10–20%, while for low power it ranges up to 50%. The distribution of prime numbers also leaves fewer options for selecting k, resulting in the ruffled behavior of the U-Connect curve, e.g., the "dip" at $L = 4$ s. Instead, RUTh enjoys many degrees of freedom, smoothly adapting lifetime to the desired latency.

Finally, we note that $p_{\text{min}} = 0.9$ yields small energy savings over deterministic discovery. Further, at times this curve drops below zero, indicating that deterministic discovery actually spends less energy. To understand why, recall that the probabilistic optimizer works under the assumption ACTIVE $< {}^T/_2 + \lambda$, as values above the half-of-period threshold denote deterministic behavior. Further, for some values of τ_{min}, it is impossible to find a configuration that guarantees high detection probability during one period, T. Therefore, to reach the high 0.9 probability by the desired latency L, two periods are sometimes required, i.e. $T = {}^L/_2$. Because a beacon is transmitted in *each* period, such configurations consume more energy. From Figure 4.15, we see that this phenomenon is exacerbated for larger values of τ_{min}, as these configurations place more constraints on the amount of activity that can occur within less than half of the period.

Lifetime-driven discovery. We take a similar approach to evaluate RUTh for lifetime-



(a) High power, $\tau_{\min} = 5$ ms (b) High power, $\tau_{\min} = 10$ ms (c) High power, $\tau_{\min} = 20$ ms



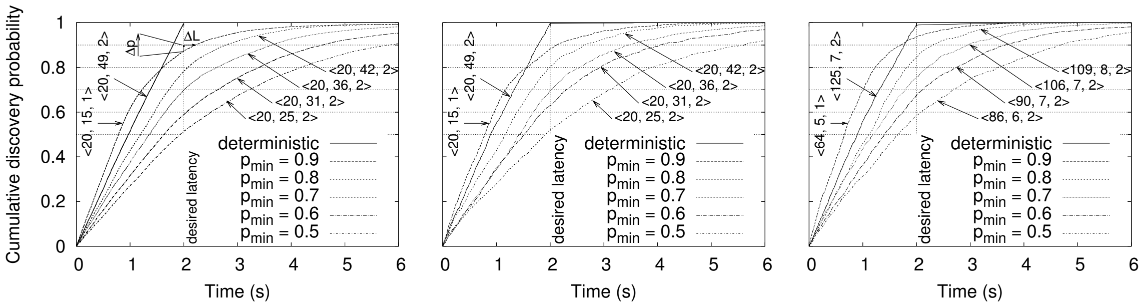
(d) Low power, $\tau_{\min} = 5$ ms (e) Low power, $\tau_{\min} = 10$ ms (f) Low power, $\tau_{\min} = 20$ ms

Figure 4.16: Lifetime-driven: relative discovery latencies for U-Connect and various RUTh configurations.

driven discovery. We start from the curves that depict the smallest discovery latency possible for a given lifetime, as shown in Figure 4.14(b). Again, results are averages of $\tau_{\min} \in \{5, 10, 20\}$ ms and sample configurations for $\tau_{\min} = 20$ ms are provided. $\tau_{\min}$ has a larger impact at the beginning of the lifetime scale, up to 41% deviation, than at the end of scale, where it reduces to 0.1%. However, because latencies are very small at the beginning of the scale, the absolute value of deviation is also small, and not visible.

To understand why, consider that if shorter lifetimes are desired, a node can afford to increase its activity by reducing the epoch duration T. However, in every epoch, a node must broadcast a beacon once and perform at least one receive check. As sampling rate and beacon duration are both determined by τ, the epoch T (and thus the latency L) has a lower bound of approximately $2\tau_{\min}$, and thus $\tau_{\min}$ has more influence on the latency. The behavior is different when a larger lifetime is desired. In this case, it is preferable to increase the number s of receive checks, as they are short and thus energy-efficient. Therefore, the influence of τ over latency decreases to the right of the lifetime scale.

Figure 4.16 compares U-Connect and RUTh with several discovery probabilities against deterministic RUTh showing, as before, $\frac{R_D - X}{R_D}$. Again, the $\tau_{\min}$ constraint has a negative impact on U-Connect. In latency-driven discovery, U-Connect fails to provide valid configurations for certain latencies. Here, for lifetime-driven discovery, U-Connect configurations are valid, but yield very large detection latencies.



(a) Simulated: RUTh configuration generated for $\Lambda = \lambda = 250$ µs.

(b) Experimental: RUTh configured as in Figure 4.17(a), implementation uses $\Lambda = 10$ ms.

(c) Experimental: RUTh configured for $\Lambda = 10$ ms, $\lambda = 1$ ms.

Figure 4.17: Discovery probability for $\tau_{\min} = 20$ ms, high power. The configuration output $\langle \tau, s, T \rangle$ is shown for each curve.

The prime k that dictates U-Connect's behavior is again the cause of this phenomenon. To achieve shorter lifetimes, smaller k values can be used that ultimately shrink both the protocol period and the discovery latency. We say that the smallest k that meets the lifetime goals is the *optimal* U-Connect configuration for that lifetime goal. However, as shown earlier, k cannot be arbitrarily small because of the constraint imposed by $\tau_{\min}$. Whenever $\tau_{\min}$ demands the usage of a prime k larger than the optimal value, U-Connect behaves in a sub-optimal fashion, achieving large detection latencies. For these values, U-Connect still meets the lifetime goals, but the discovery latency it provides is larger than in cases where the $\tau_{\min}$ constraint does not come into play.

Unlike latency-driven discovery, where time constraints between radio activations determined the valid U-Connect configurations, here the transmission power influences where the optimality line should be set. For U-Connect, lower transmission power means a slight decrease of k is possible while maintaining the same overall power consumption. Therefore, smaller latencies are theoretically achievable for larger lifetime goals. Nevertheless, if the limits on $\tau_{\min}$ are still in place, the only result of the lower transmission power is that the optimality line is shifted to the right w.r.t. higher transmission power. To understand why RUTh can only offer large discovery latencies for small lifetime goals, we recall that when $\text{ACTIVE} < {}^{T}\!/_{2} + \lambda$, RUTh works in probabilistic mode. This can be written as $T > 2\,(\text{ACTIVE} - \lambda)$, which means that the period, and thus the discovery latency, is lower-bounded by ACTIVE, which itself is subject to variations of $\tau_{\min}$. By switching to deterministic mode, this constraint no longer exists, leading to faster discoveries.

4.7.2 Impact of Model Assumptions

The model for probabilistic discovery in Section 4.4.2 assumes a contact time $\gamma = 0$, i.e., contact occurs at the beginning of the epoch of one of the nodes. In reality, this initial contact occurs randomly, thus this assumption introduces some error in the model. To

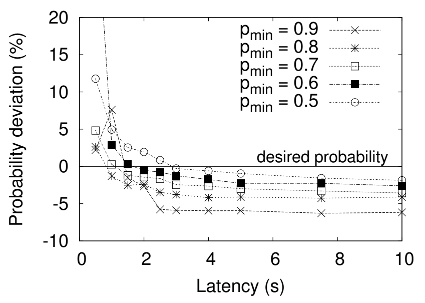


Figure 4.18: Probability deviation of simulation vs. model (Δp in Figure 4.17(a)) for high power and $\tau_{\min} = 20$ ms.

evaluate this aspect, we developed a simulator that, given a RUTh protocol configuration, produces the cumulative discovery probability over a large number (e.g., 1 million) of tests. The simulation results presented here are of interest mainly to assess the accuracy of the RUTh model, therefore U-Connect is no longer shown.

Figure 4.17(a) shows the simulation results for the configurations $\langle \tau, s, T \rangle$ output by our optimizer for maximum latency $L = 2$ s, $\tau_{\min} = 20$ ms, high transmission power and various values of $p_{\min}$. In these configurations, the optimizer chooses $T = 2$ s, except for $p_{\min} = 0.9$, for which $T = 1$ s. As with any probabilistic process, our cumulative discovery probability approaches asymptotically the horizontal 1. Worth noting is the fast evolution: all lines are above 0.9 probability within $3T$. The chart shows also the output for deterministic discovery where, as expected, contact is always detected within T. In this case, the cumulative probability has a linear behavior because the only source of randomness is the contact time γ, chosen uniformly at random in $[0, T)$.

The effect of the assumption about γ is that the simulated curves do not intersect the 2 s vertical line exactly at the desired probabilities (Figure 4.17(a)). For some configurations (e.g., $p_{\min} = 0.9$), the actual discovery occurs with a slightly lower probability, while for other configurations (e.g., $p_{\min} = 0.5$) the simulation shows a greater probability of discovery at 2 s. We consider the *error* of our model to be the difference between the simulated curve and the target probability set as input to the solver. A positive error means that the probability shown through simulation for a given latency is larger than the desired $p_{\min}$. This measure is denoted by Δp on Figure 4.17(a). The dual deviation in time ΔL is also a measure of error, but because its values are similar to Δp, we report only the error measured on the probability axis.

To quantify Δp we ran simulations for both high and low transmission power, $\tau_{\min} \in \{5, 10, 20\}$ ms, and target latencies between 0.5 and 10 ms. Figure 4.18 shows a sample result for the interesting scenario of high power and large $\tau_{\min}$. All other plots are analogous, and not shown for space reasons. In all cases, at low latencies our model yields higher probabilities than $p_{\min}$. There is however a point after which the relative error is constant and slightly below the target, with a maximum deviation of 6.13%. Depending on the application, this small deviation can either be ignored or used to modify the constraints on the latency/lifetime goals.

4.7.3 Empirical Evaluation

To validate the previous results using real hardware, we conducted a set of micro-benchmarks with Telos motes. We analyze the discovery latency *i)* between a pair of nodes in isolation *ii)* with varying neighbor densities.

The results are averages over 1500 repetitions. Each node is controlled by, and reports results to, a PC connected through a wired channel. The PC selects a random phase uniformly distributed in $[0, T)$ for all nodes. The nodes execute RUTh for 15 s, recording all discoveries. The logs are then parsed to determine the effect of a contact, as follows. For each of the 1500 experiments, a contact time γ is selected uniformly at random in $[T, 2T]$. The discovery latency is computed as the difference $\delta - \gamma$ between the contact time γ and the first detection δ in the log such that $\delta > \gamma$.

Cumulative discovery probability. To evaluate the discovery latency between two motes, we re-create experimentally the cumulative discovery probability. In doing this, we have two conflicting goals. On one hand, we want to compare with the simulated results, which in turn relate to the analytical ones. These, however, assume a receive check $\Lambda = 250$ µs that, as discussed, is impractical. On the other hand, we want to show that our implementation, relying on a $\Lambda = 10$ ms, meets the latency and probability targets when configured by the optimizer for this receive check duration.

Therefore, we ran two sets of experiments, whose results are shown in Figure 4.17(b) and 4.17(c). In both cases, we use the same optimizer requirements as in the simulated curve in Figure 4.17(a), i.e., latency $L = 2$ s, $\tau_{\min} = 20$ ms, high power and various values of $p_{\min}$, including the deterministic case.

In Figure 4.17(b) we configured the optimizer for $\Lambda = \lambda = 250$ µs, obtaining the *very same* configuration used in simulation. However, we run it on the implementation with reliable $\Lambda = 10$ ms receive checks. Recall that the receive check duration bears a negligible effect on discovery probability, which is determined by ACTIVE, independent of Λ. The results match closely the simulated ones in Figure 4.17(a), with a very small probability increase. In deterministic mode, the only challenge is packet loss, negligible in our case.

The experiments in Figure 4.17(c) are instead based on the configuration generated by the optimizer for $\Lambda = 10$ ms. A *direct* comparison with the previous ones is no longer possible: as shown in the figure, the configurations output by the optimizer are different. However, the goal here is to verify if the optimizer matches the *same* latency and probability *targets* when configured for the implementation. The results confirm this, although the achieved discovery probability always exceeds the target one—from 8.6% for $p_{\min} = 0.5$, down to 3.6% for $p_{\min} = 0.9$. The reason lies in the error of our model, due to the $\gamma = 0$ assumption. Contrary to the configuration we analyzed in Section 4.7.2, where the model provided an overestimate w.r.t. the simulations in Figure 4.17(a), in this case we verified through analogous simulations that the model *underestimates* probability. As mentioned earlier, this can be easily discovered (and compensated for) by checking the configuration output by the optimizer against the simulator (whose computation lasts a few minutes) and, if necessary, by adjusting slightly the latency goals. What is important, however, is that the probability increase of the experimental curves in Figure 4.17(c) w.r.t. the simulated ones remains very small (within 3%) as in the previous comparison between Figure 4.17(b) and 4.17(a). This small (and positive) error is mostly caused by the imprecision of TinyOS timers and by rounding approximations, and confirms that the reality of implementation matches the configuration tool chain enabled by our analytical and simulation results.

Neighbor density. To study the impact of multiple nearby nodes we employ 12 motes, a relatively high number for our application scenarios, and measure the (average) time for a randomly-selected mote to discover or be discovered by its neighbors. In this experiment

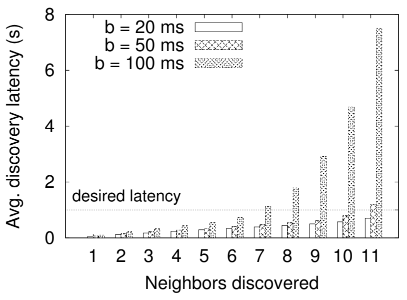


Figure 4.19: Neighbor discovery time for $T = 1$ s, $s = T/2b$.

we set $T = 1$ s, $b \in \{20, 50, 100\}$ ms, and $s = T/2b$. The last setting defines this as deterministic behavior hence, in a perfect world, discovery between all pairs of nodes should occur within one epoch.

Results in Figure 4.19 show that $b = 20$ ms enables discovery of all neighbors within the deadline, and $b = 50$ ms causes a small delay only for the last neighbor. When $b = 100$ ms is used, however, the discovery of neighbors after the 7^{th} is significantly delayed: it takes almost 7 epochs to detect the last neighbor. This is easily explained by considering the node behavior when channel activity is detected. Before any beacon transmission, the radio performs a CCA and sends only if the channel is clear. If not, the node waits for a random period of time, then attempts sending its own beacon. With a long 100 ms beacon and a relatively short epoch of 1 s, the channel is frequently busy, and it is not possible for all 12 motes to transmit in each epoch. In general, the risk for N nodes to find the channel busy is proportional to $\frac{Nb}{T}$. The other protocols in Section 4.2, also relying on beacons, are subject to similar constraints in the case of dense scenarios. In our case, however, applicative knowledge can be easily used to further constrain the optimization problem w.r.t. the maximum beacon length, by imposing $b < b_{max}$. Thus, for instance, in cases where RUTh is used in crowded settings (e.g., a group of Alzheimer patients in the nursery) a small b_{max} should be used, while larger values can be allowed when nodes are sparse (e.g., in the case of solitary wildlife).

4.7.4 RUTh as a Social Contact Sensor

In this section, we investigate the opportunity of using RUTh as a human social contact sensor. Our goal is to understand how contacts perceived by humans match contacts detected by human-borne motes. There are two dimensions of this problem; specifically, we measure:

- the number of *false positives*, that is, the number of contacts detected by RUTh

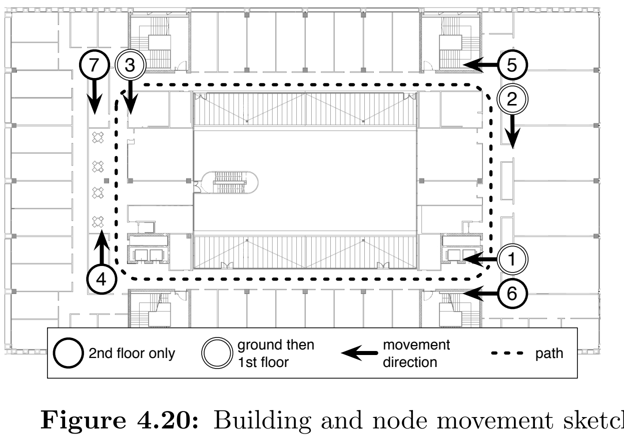


Figure 4.20: Building and node movement sketch

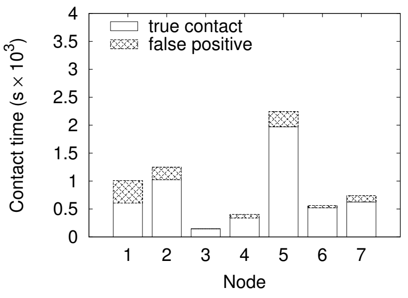


Figure 4.21: Total duration of contacts for each node.

and not reported by humans, and, dually

- the number of *false negatives*, that is, the number of contacts reported by humans and not detected by RUTh.

This experiment summarizes data gathered from 7 volunteers. Each volunteer carried a mote running RUTh and moved on a predefined path in a three-story building. Unlike people, motes may incorrectly detect contacts between two bearers located in different rooms or on different floors; thus, to study the impact of walls, we split the participants in two groups. Bearers of nodes 1, 2, and 3 first completed five laps of the path in Figure 4.20 on the ground floor, and then completed other five laps of the same path, but on the first floor. Bearers of 4, 5, 6, and 7 looped continuously on a similar path on the second floor. To guarantee that participants meet, some participants moved clockwise, while other moved counter-clockwise, as indicated by the arrows in Figure 4.20.

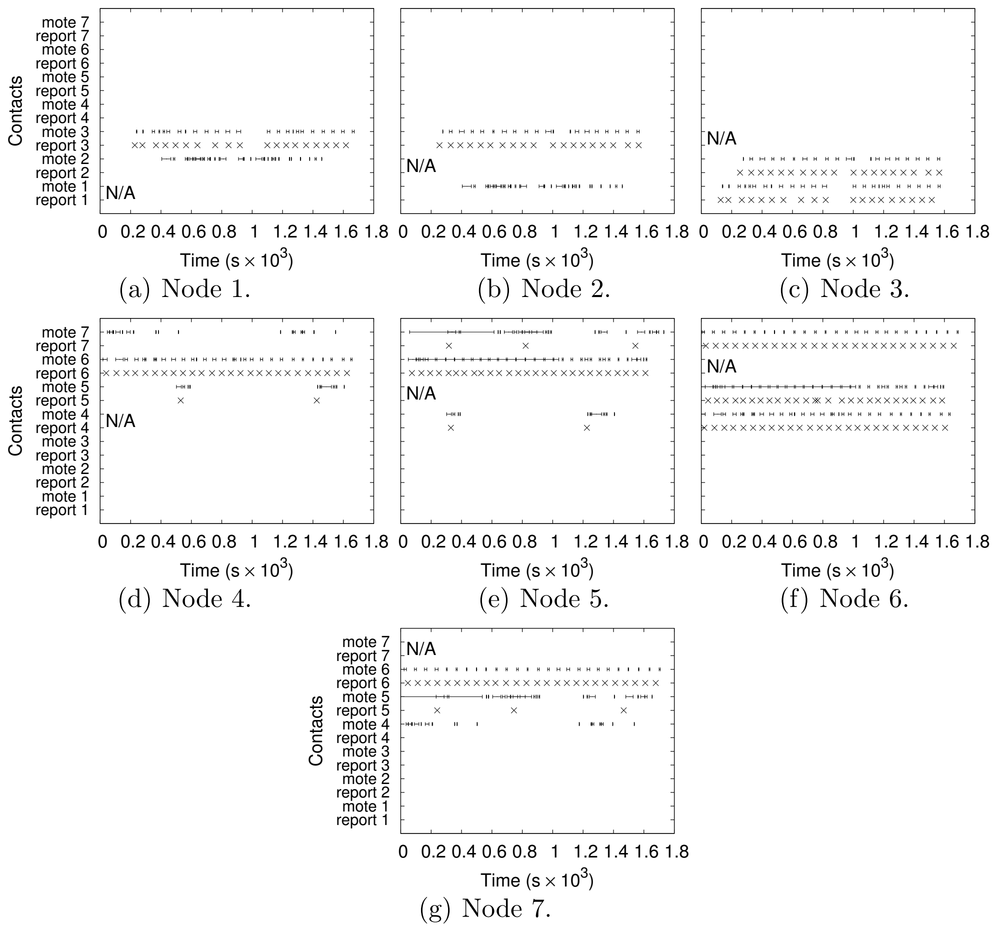


Figure 4.22: Timeline showing the contacts for each node in the experiment. Contacts recorded by volunteers are on the horizontal lines called "report", while contacts recorded by RUTh are on the lines called "mote".

Each participant recorded the time when he/she met any other participant. We matched these records against traces collected from the nodes. The radio of the nodes was configured to use low transmission power, i.e., -25 dBm. Previous experiments showed that this power roughly corresponds to a 9 m line-of-sight transmission range. We configured RUTh to achieve 100% detection probability within a 2 s latency. The parameters corresponding to this configuration are $\langle \tau, s, T \rangle = \langle 167, 5, 2 \rangle$.

In Figure 4.21 we plot the total duration each node reports as being in contact with any other node. We can use this figure to outline the amount of false positives. Some of these are explained by the paths followed by the volunteers in this experiment. Consider, for instance, the case of nodes 1 and 2: these have been both traveling in the same direction, on the same floor. As the two volunteers were not facing each other, they did not record a contact, although the two motes were in sufficient range to detect a contact.

Figure 4.22 confirms this argument. Here, we illustrate a timeline for each node on which we mark both the contacts reported by volunteers and those detected by motes. The former are marked by dots, as participants recorded a single timestamp for each contact[2], while the latter are marked by intervals containing a series of consecutive timestamps that together form a contact as detected by RUTh. In addition to the false contacts between nodes 1 and 2, we observe a similar issue in the case of nodes 5 and 7, which also moved in the same direction on the same floor. Moreover, we observe that *i)* no contacts were detected across floors, and *ii)* most of the contacts recorded by volunteers have also been detected by RUTh. In fact, the percentage of false negatives out of a total of 160 contacts recorded by volunteers is only 1.25%.

From this experiment, we conclude that RUTh can be efficiently used as a human contact sensor.

4.7.5 RUTh as a Proximity Sensor: Lessons Learned From A Dense Deployment

In the assisted living scenario, we use RUTh to detect when a person carrying a mobile node is "close" to a hazard tagged by a fixed node. In this section, we describe the experience we accumulated during an attempt to recreate this scenario in our testbed. When designing these experiments, we believed that we could use RUTh to obtain approximate proximity information. From this aspect, this section goes beyond the scope of evaluating RUTh as a protocol for neighbor discovery. Indeed, ranging is subject to imperfections of the wireless environment such as interference, collisions, and the non-linearity of the signal propagation.

As it turned out, the realities of wireless communication have a strong impact on the number of contacts that can actually be detected. Although we met our goal only partially, the hereafter described experiments provide valuable insights on the inner-workings of RUTh. Moreover, they are an excellent opportunity to study the factors affecting the accuracy of our protocol.

[2]If a contact is reported by both volunteers, it is marked once on Figure 4.22.

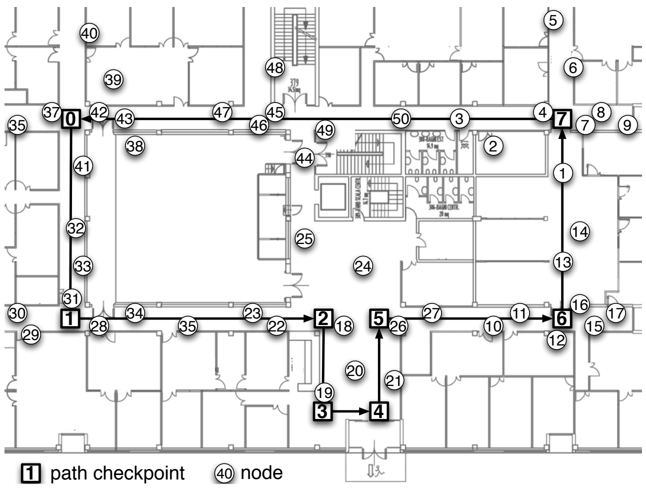


Figure 4.23: Testbed and mobile node path.

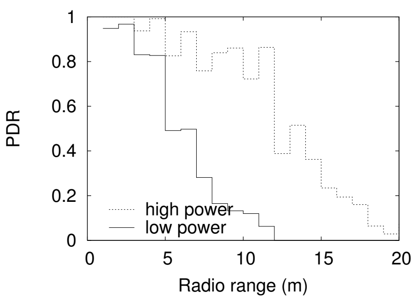


Figure 4.24: Histogram of the PDR function of the distance sender–receiver for our testbed. Averages over 300 packets / node, each broadcast in a dedicated time slot for each node.

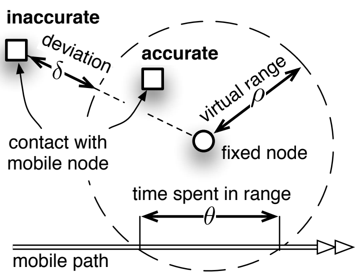


Figure 4.25: Definition of contacts.

Experiment settings and goals. In our attempt, we use a testbed consisting of 50 fixed nodes deployed under the floor of a 60 m × 40 m office building. The nodes of the testbed play the role of hazard tags, while a person moving around the testbed plays the role of a patient. The person loops for about 30 min at normal pace in the vicinity of the testbed nodes, as sketched in Figure 4.23.

We use a virtual range ρ to define the distance threshold at which the mobile mote is considered as being "close" to a fixed mote. We vary ρ to match possible values of interest in the assisted living scenario. As partially depicted by Figure 4.25, we distinguish several types of contacts, which ultimately become the metrics we employ in our assessment:

- the number of contacts between a fixed node and the mobile node detected *inside* the virtual range ρ. From the perspective of our application, these are contacts that are "accurately" detected.
- the number contacts detected outside ρ. These are regarded in our application as "inaccurate". For these, we report the distance δ from the virtual range, as illustrated in Figure 4.25.
- the number of *missed contacts*, that is, the count of how many times a mobile node enters the virtual range ρ of a fixed node and no contact is detected. From these, we exclude the instances in which the mobile nodes passes "too fast" through the virtual range for RUTh to detect contact, that is, when $\theta < T$ in Figure 4.25.

We focus on configurations $\langle \tau, s, T \rangle$ optimized for low power (i.e., -25 dBm), as these are more relevant to the ranges required in the proximity detection problem. We configure RUTh to achieve 100% of detections within a latency $L = 2$ s, i.e., we use $\langle \tau, s, T \rangle = \langle 167, 5, 2 \rangle$. Nevertheless, to better understand the impact of density and for completeness, we additionally repeat the experiment with the radio configured in high power mode (i.e., -1 dBm).

Characteristics of the testbed. First, we assess the factors of the wireless environment that RUTh by using only the fixed nodes of the testbed. For simplicity, here we report only on the experiments that employed high power, which increase density, attaining thus the worst-case scenario for our testbed and providing more interesting insights. In this setting, the average neighborhood size N in our testbed contains 13.65 nodes. Channel contention is high under these conditions: ideally, all nodes in a neighborhood take $N \times b =$ 2279.55 ms to broadcast their beacons. Note that even in an ideal world the allotted time for communication is insufficient, as all beacons must fit in a single epoch $T = 2$ s; thus, some contacts are prone to be undetected. Next, we look at this aspect in more detail, focusing on possible ways to alleviate the problem.

Unfortunately, ground truth cannot be easily established even in the case of our testbed, due to the factors affecting wireless communication and, in a later case, due to the human inability to follow a precise path. Therefore, we must compare the results against a reference curve that is unbiased by the duty-cycle scheme in RUTh. We empirically determine the reference line through an experiment in which nodes broadcast periodic beacons according to the RUTh schedule but, unlike RUTh, they keep the radio always on and listen continuously.

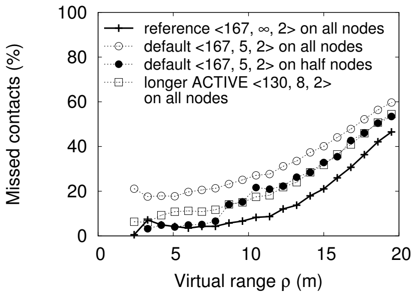


Figure 4.26: Contacts among fixed nodes missed when using high power and configurations $\langle \tau, s, T \rangle$.

Figure 4.26 shows the results for 40 min long experiments using high power. We first observe the general trend of missing contacts as ρ increases. Note that, the higher ρ is, the more nodes are included in the virtual range and the greater the distance between sender and receiver is. However, in practice message exchange among these nodes is far from being guaranteed, due to the characteristics of the wireless communication. These characteristic can be observed in Figure 4.24, which shows how the packet delivery ratio (PDR) drops in our testbed as the distance between sender and receiver increases. Thus, *the limitations of the wireless communication are more likely to be observed when expecting a large proximity range ρ.*

We analyze the cause for which the number of contacts missed by RUTh (i.e., the curve for $\langle 167, 5, 2 \rangle$) is comparatively higher w.r.t. the reference line (henceforth denoted by $\langle 167, \infty, 2 \rangle$). To this end, we computed the inter-contact delay, i.e., the average time elapsed at each node between two consecutive detections of the same node. Ideally, this should match the epoch $T = 2$ s. In practice, an inter-contact delay longer than 2 s indicates that epochs are skewed due to back-offs of the beacons. Indeed, our logs show that, in the case of the experiment corresponding to the line $\langle 167, 5, 2 \rangle$ in Figure 4.26, the inter-contact delay is 2096 ms. Comparatively, when repeating the experiment with low-power, we measure an inter-contact delay smaller by 35 ms. From this, we infer that *the higher density is, the more skewed epochs are* and the more contacts escape undetected as the epochs of sender and listener are no longer synchronized. However, when the radio is always on as in our reference, nodes can always receive beacons, irrespective of how long the beacons were delayed. Thus, as observed, deployment density has little impact on the reference curve.

We further investigate if we can work around the problem by increasing the duration of the ACTIVE interval and, at the same time, keep T constant. Thus, we change the constraints in our optimizer to require a 15% longer than normal ACTIVE interval, i.e., ACTIVE $= 1.15 \times T/2$. We repeat the experiment on the fixed nodes to obtain the line in Figure 4.26 corresponding to the resulting configuration $\langle 130, 8, 2\rangle$. Although we notice an improvement w.r.t. to the case of the configuration $\langle 167, 5, 2\rangle$, this is not enough. We infer that *compensating density with longer* ACTIVE *intervals mitigates contact loss, but not completely.*

Then, we repeat the experiment with half of nodes offline to reduce density. In this experiment, the average neighborhood size is $N = 6.12$. Also in this case loss is expected; indeed, as $T > N \times b = 1022 > T/2$, although now there is enough time to broadcast beacons, the receiving node may not be awake to receive all beacons. Interestingly, the results, also shown in Figure 4.26, overlap almost perfectly with the reference line for small ρ, validating thus that a listening node does receive all beacons from a small subset of its neighbors. However, a greater ρ covers a larger number of nodes with which contacts are expected; in practice, not all of these are detected and hence the deviation from the reference line.

Therefore, to detect all contacts in dense environments, we should *i)* set an upper bound for the duration b of beacons (specifically, we set $b < b_{\max} \triangleq T/N$), and *ii)* use longer ACTIVE intervals to compensate for epoch skew (i.e., ACTIVE $= 1.15 \times T/2$). These constraints can be embedded in our optimizer to obtain the configuration yielding the lowest current consumption, respectively the longest lifetime for latency-driven, respectively lifetime-drive neighbor discovery. We re-run the experiment with all nodes active configured for high power and desired latencies $L \in \{2, 4, 6, 8, 10\}$ s under these two "compensating" constraints. In the results depicted by Figure 4.27 we observe that there are virtually no missed contacts for ranges $\rho < 10$; these results are in line with the limits of wireless communication illustrated by the PDR in Figure 4.24. Nevertheless, this gain

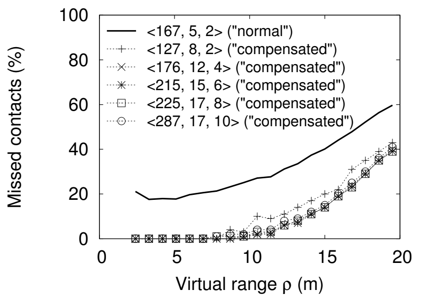


Figure 4.27: Contacts among fixed nodes missed when using high power, various desired latencies $L = T$, and configurations $\langle \tau, s, T\rangle$ compensating for epoch skew.

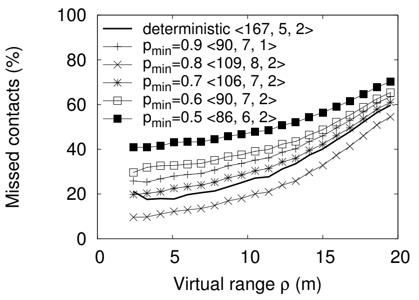


Figure 4.28: Contacts among fixed nodes missed when using high power, a desired latency $L = 2$ s and configurations $\langle \tau, s, T\rangle$ for probabilistic discovery.

comes with a trade-off in energy consumption; in this respect, our optimizer estimates a 7% increase when comparing the estimated current consumption of the "compensated" configurations w.r.t. the equivalent non-"compensated" configurations for all values of L used in the experiment.

Probabilistic discovery. Interestingly, there are cases when *probabilistic discovery achieves a greater number of detected contacts w.r.t. to deterministic discovery* in dense environments. In Figure 4.28 we illustrate the results of an experiment run on all fixed nodes using high power, no compensation for epoch skew and various probabilities. We observe that when setting $p_{\min} = 0.8$ we detect more contacts w.r.t. the case of deterministic discovery. This is due to the fact that the duration of the beacon b is smaller and inherently so is the channel contention. However, for the same desired latency[3] $L = 2$ s, as expected and as observed in the same figure, the lower $p_{\min}$ is, the greater the number of missed contacts is. In fact, in our experiment, there is a demarcation line at about $p_{\min} = 0.7$ where probabilistic RUTh detects as many contacts as deterministic RUTh; it does so by using a lower amount of energy consumption and consequently probabilistic discovery may be preferred in a real-world application. For lower probabilities, i.e., $p_{\min} < 0.7$, significantly more contacts are missed.

The mobile node. When experimenting with the mobile node, we use both low power, as dictated by the requirements of the proximity detection application, and, for completeness, high power. The average neighborhood size when using low power is 5.44, which is also close to the reliability limit: consider that in TinyOS back-offs are randomly distributed between 10 ms and 50 ms. If we add an average of 30 ms to b, even now $N \times b > {}^{T}\!/_{2}$ and thus there is not enough time to receive the beacons of all neighboring nodes.

In Figure 4.29 we plot the percentage of missed contacts. We observe the impact of the deployment density is more evident in the case of high power: due to the epoch skew, RUTh loses a monotonically increasing number of contacts with respect to the reference line which remains rather constant. We also observe that the reference line looses a very small amount of packets (due to the previous argument). However, the inflection point of the reference line at which contact losses appear is about $\rho = 13$ m. Beyond this distance, we conclude that the wireless signal attenuates considerably. This is also indicated by the PDR drop for high power in Figure 4.24 that appears approximately at the same range.

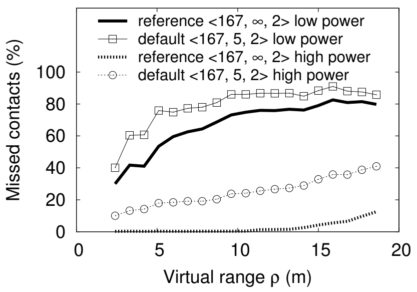


Figure 4.29: Missed contacts between the mobile node and the fixed nodes.

Differently, the signal attenuation is visible from the start for the two curves associated to low power transmissions. From Figure 4.24, we expect a considerable amount of packets to be lost even for small virtual ranges ρ which is the case, as indicated by Figure 4.29. Moreover, we also notice that the difference between RUTh and the reference line remains roughly constant for all virtual ranges ρ, an indicator that den-

[3]The optimal configuration that achieves $p_{\min} = 0.9$ at a desired latency $L = 2$ s employs a duration of the epoch $T = 1$, unlike the configurations for lower $p_{\min}$ that all have $T = 2$ s. Also, the smaller T is, the more significant the channel contention is and thus the greater number of missed contacts observed in Figure 4.28 when RUTh is configured for $p_{\min} = 0.9$.

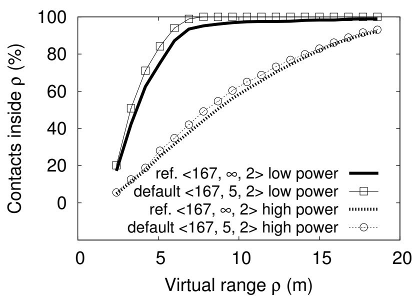


Figure 4.30: Mobile node contacts detected inside the virtual range.

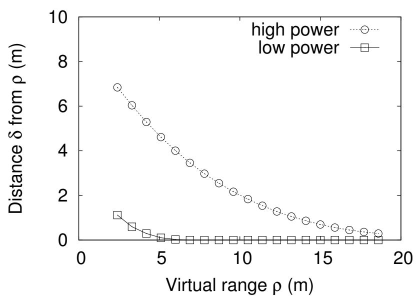


Figure 4.31: Distance from ρ for $\langle \tau, s, T \rangle = \langle 167, 5, 2 \rangle$ measuring δ in Figure 4.25.

sity has a lower impact.

As a general comment, we observe that the numbers reported here are smaller than these reported in the case of fixed network, i.e., in Figure 4.26. This is mainly due by the fact that, in Figure 4.29 we exclude the case in which the mobile passes too fast through the range of a fixed node. This is a subject for further experimentation, for instance, by demanding the carrier to move at smaller velocities.

Then, we focus only on the contacts that are detected. In Figure 4.30 we show the ratio between the number of the contacts between fixed nodes and the mobile detected *inside* the virtual range ρ and the total number of detected contacts. We observe that RUTh detects roughly the same amount of accurate contacts as the reference line, both in the case of low power, but also in the case of high power. Indeed, we expected similarities to the reference lines, as the breakdown of contacts according to ρ is beyond the control of any protocol; rather, it is caused only by the properties of the wireless environment. Even so, when using low power, we observe that RUTh can accurately detect whether contacts occur within a virtual range $\rho = 7$ m. Furthermore, as indicated by Figure 4.31, the distance at which inaccurate contacts occur w.r.t. to the virtual range is small, i.e., on average below 2 m for low power and below 8 m for high power.

These preliminary experiments indicate that the density of our testbed negatively impacts contact loss in RUTh. Nevertheless, through a careful planning of the deployment, RUTh could be employed in practice as proximity sensor when using low power transmissions. As we observed in Figure 4.30, low power has the nice benefit that the proximity threshold can be more accurately set w.r.t. high power.

4.8 RUTh in Action: The Assisted Living Application

In Chapter 3, we sketched a solution for the assisted living application in which mixture of fixed and mobile nodes are used to detect proximity and social contacts. In this solution, all nodes are leveraging on RUTh, although the implementation of beacons and the duty-cycle scheme is done at a higher level, i.e., using the interfaces provided by our TeenyLIME middleware. In ACube, in addition to sensing contacts, the radio is employed to collect data along a multi-hop tree. Hereafter, we discuss in detail the integration of RUTh with the collection protocol that we employ. The goal of this section is to show that a simple routing structure can be maintained on top of RUTh which arguably employs a rather aggressive duty-cycle scheme. To this end, in Section 3.1 we detail the operation of the collection protocol in the presence of RUTh, while in Section 4.8.2 we report results that we obtained from early experiments in a laboratory testbed.

4.8.1 RUTh and Data Collection

Recall from Section 3.1, that in ACube mobile nodes offload the data to a fixed network organized as a collection tree. Also, both mobile and fixed nodes are running RUTh. Data collection from mobile nodes occurs in two steps. First, mobile nodes offload the events they generate to neighboring fixed nodes. And second, once the events are unloaded, fixed nodes relay them towards the sink across a multi-hop path. We detail these steps next.

Mobile-To-Fixed Offload. All events generated by mobile nodes are sent in broadcast and received by any fixed node in the neighborhood. Due to the peculiar operation of the radio in RUTh, broadcasts must be repeated with a period of $T/2$, where T is the neighbor discovery epoch, to ensure that at least one receiver is listening at the time of the transmission, as depicted by Figure 4.32. The periodic broadcast stops when either a fixed node acknowledges the transmission, or when a count threshold is reached, after which the message is dropped.

This protocol is prone to creating event duplicates, that is, two fixed nodes receive and relay the same event from a mobile node. One step we take in this direction is to require all fixed nodes to wait a short timeout before acknowledging the broadcast of a mobile node. During the timeout, fixed nodes sniff the channel for ACKs to the same broadcast[4] from other nodes. If such an ACK is received, the event is dropped. Also, the duration of the timeout is inversely proportional to the height in the tree. In result, nodes closer to the root take precedence in capturing mobile events.

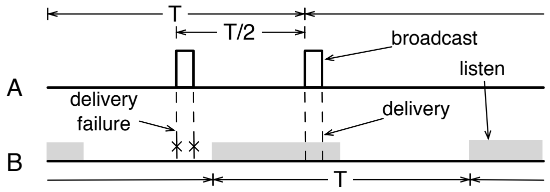


Figure 4.32: Periodic broadcast scheme to ensure delivery in ACube.

Collection over fixed nodes. Once an event is successfully delivered to a fixed node, we route it using a protocol previously employed by our group successfully in other deployments (i.e., [16, 17]). In this protocol, the sink periodically rebuilds the collection tree by flooding the network with a control message. The tree structure is built in such a way that the path from any node to the root minimizes the end-to-end LQI [1]. Forwarding reliability is ensured by a simple hop-by-hop recovery scheme, where transmissions are acknowledged as the events they carry travel upstream.

We adapted the protocol as follows. First, the broadcast of beacons must be repeated twice with a space between retransmissions of $T/2$, as per Figure 4.32. We guarantee thus that any neighboring node is listening at the time when the beacon is broadcast. Second, all data transmissions are synchronized with the schedule of parents. When broadcasting tree refresh beacons, node also piggyback information regarding their neighbor discovery

[4]We uniquely identify each event by a pair (originator node; sequence number).

schedule. Thus, whenever a node changes its parent, it has information on when the parent is listening and can schedule transmissions accordingly. Finally, as an additional step to reduce duplicates, all nodes maintain a circular buffer in which they store identifiers of previously forwarded events; the buffer prevents any forwarding of event duplicates.

4.8.2 Results[5]

We evaluate RUTh in the context of ACube using 16 nodes deployed in our laboratory testbed that mimic the environment of retirement houses, as illustrated in Figure 4.33. In early experiments, we observed that motes are sometimes screened by the body of patients. Therefore, in ACube, as well as in our testbed, nodes are deployed in pairs to achieve redundancy in the detection of proximity.

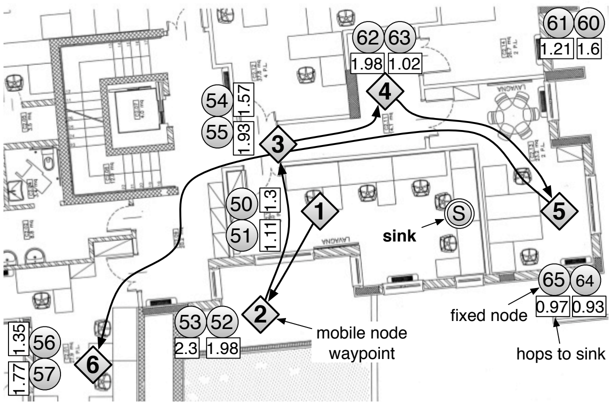


Figure 4.33: ACube test deployment.

The events reported by the WSN can be critical, e.g. a "fall down" alarm triggered by a patient-borne mote, and require the immediate attention of a care giver. Therefore, we focus on delivery *reliability* and *latency*. We additionally assess the network overhead, which is a measure of energy efficiency and, indirectly, of maintenance costs. All previous metrics are composed of two measures that can be accumulated to obtain the overall cost:

1. the cost to offload from fixed to neighboring fixed nodes.
2. the cost on fixed nodes to route the events to the sink.

In this evaluation we study each of the costs independently w.r.t. the other.

Mobile-to-fixed Reporting

We evaluate the performance of the mobile-to-fix offload protocol. To this end, we use a mobile node whose movement is summarized by the path illustrated in Figure 4.33. Between two consecutive waypoints, the mobile node was carried at normal walking speed; to each of these waypoints corresponds a motion pause of approximately 30 s. The mobile nodes generates an event every 5 seconds.

As discussed previously, our focus is on the delivery latency and reliability. To analyze the former, we measure the round trip time, i.e., the time on a mobile node between an event is broadcast and an acknowledgment is received back from a fixed nodes. We use $\langle \tau, s, T \rangle \in \{\langle 100, 5, 1\rangle, \langle 100, 10, 2\rangle, \langle 100, 15, 3\rangle\}$, which yield a deterministic discovery schedule. Because a mobile node is usually in the range of several fixed nodes that may have complementary schedules, the round-trip time is small, i.e., always smaller than 200 ms. Because of redundancy, we also experienced an insignificant event loss, i.e., only 0.32%.

[5]The data in this section previously appeared in [21].

One of our concerns was that the repeated broadcast to offload messages to fixed nodes can lead to a high overhead. In this respect, we counted a merely 2.13 retransmissions required by a mobile node to offload its events to the static network.

Cost of the Fixed Network

We then evaluate the behavior of the collection tree built on top of RUTh. In this set of experiments, we did not use events generated by mobile nodes. Instead, we configured all fixed nodes to generate synthetic events. The measures henceforth reported must be combined with the cost of the mobile-to-fixed reporting scheme to obtain the overall costs.

Tree collection latency. We expect the delivery latency of our protocol to change as function of the proximity detection period T and by the number of hops traversed. We verify this by reconfiguring our deployment with $\langle \tau, s, T \rangle \in \{\langle 100, 5, 1 \rangle, \langle 100, 10, 2 \rangle, \langle 100, 15, 3 \rangle\}$. Then, we setup the network topology to remain unchanged throughout the experiment in order to know precisely the depth of each fixed node in the collection tree. We measure delivery latency as the time spent between the generation of the event and the moment this is delivered to the sink. To accurately measure latency, we connect the sink and the generating nodes to machines synchronized using NTP [86].

The results are illustrated in Figure 4.35. First, we observe the very small latency of events generated by nodes one-hop from the sink. This is explained by the fact that the sink has the radio always on. Instead, the events generated at two hops must pass through only one intermediary node. We observe that the delivery latency is directly proportional to T. However, this observation does not stand for longer paths: while latency generally increases with the number of traversed hops, the latency is not any more proportional to T due to the several synchronizations parent-child that appear across the longer paths.

Tree collection reliability. We use $\langle \tau, s, T \rangle = \langle 100, 10, 2 \rangle$. We initially assumed that the inactive periods employed by RUTh will exacerbate the reliability of collection. In this experiments, all fixed nodes generated events with a rate of 1 event every 5 s. In total, each node generated approximately 70000 events during 95 h of testing, all collected over a tree rebuilt every 10 s. Note that this traffic profile is unrealistic and is actually a stress-test for our system. Despite this, during the 95 h stress-test, our protocol managed to deliver 95.4% of all the events generated by the fixed nodes.

An interesting aspect is how event loss is correlated with the number of hops an event

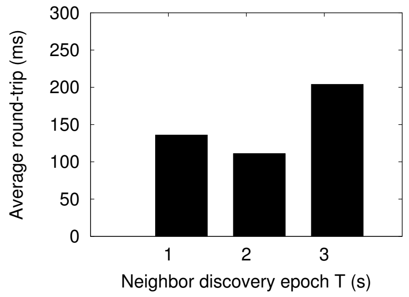


Figure 4.34: Round-trip for mobile-to-fixed.

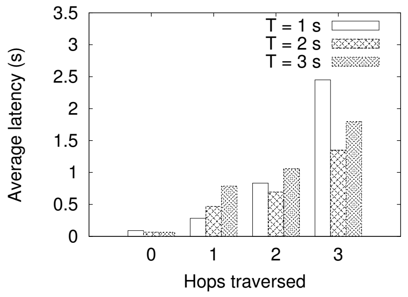


Figure 4.35: Latency function of path length.

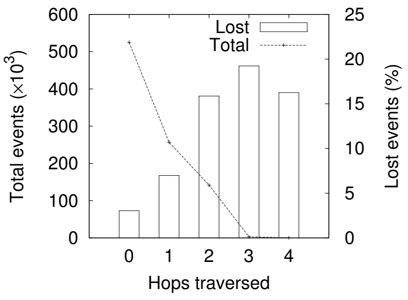


Figure 4.36: Event loss analysis.

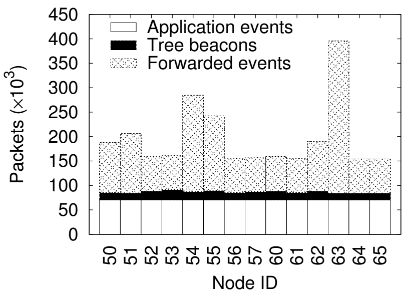


Figure 4.37: Network overhead.

must travel to reach the sink. For the indoor scenarios that we target (retirement houses), these are likely to be small. In fact, as can be seen from Figure 4.36, the majority of events traverse at maximum 2 hops to reach the sink, and only a negligible amount of packets traverse 3 or 4 hops. Also from Figure 4.36, we can see that the loss rate increase with the hop count.

Tree collection overhead. As an indirect measure of the energy efficiency, we distinguish between packets based on the type of information they carry. These can be *i)* the overhead to maintain the routing structure, represented by beacon employed to refresh the collection tree, *ii)* the overhead to route data, represented by events generated by other nodes and forwarded to the sink, and *iii)* the actual data, that is, events generated locally by a node. Figure 4.37 illustrates this breakdown. We can see that under our stress-test, the tree maintenance cost is negligible as compared to traffic carrying events.

In conclusion, ACube serves as example of a scenario in which we maintain a routing overlay on top of a RUTh-like duty-cycle scheme. Data exchange is efficient only as long as the packet sources have information on when the next hop is awake. In our application, as data flows in one direction, it is sufficient to synchronize the transmissions of children with the schedule of their parent. However, if no information on the schedule of the receiver is available, transfer may become twice as expensive, as in the case of the mobile-to-fixed offload scheme and of the tree refresh rounds.

4.9 Discussion and Outlook

We formulated the neighbor discovery problem as an optimization problem where the objective, based on the application requirements, is to either minimize current consumption (therefore maximizing lifetime) given a goal in terms of maximum discovery latency or, dually, to minimize the maximum discovery latency given a goal in terms of lifetime. We have shown that allowing end-users to express the requirements on latency in probabilistic terms provides not only additional flexibility but also margins of improvement in achieving the desired goal. This formulation allowed us to devise a protocol that brings a significant advance over state-of-art neighbor discovery protocols. Our implementation takes into account low-level characteristics of the mainstream hardware, to ensure that the protocol is readily available for use in real-world deployments. We will use RUTh in two such deployments, concisely described in Section 4.1: a (latency-driven) assisted-living application concerned with the care of Alzheimer's patients, and (lifetime-driven) monitoring of the social behavior of wildlife, specifically roe deer.

Chapter 5

The Group Membership Problem[1]

The need to monitor groups of mobile entities arises in many application contexts. Examples include the study of the social behavior of humans and wildlife, the shepherding of livestock, the care giving to people that are not self-sufficient.

Human- or animal-borne wireless devices can be used to detect the joining or leaving of group members, even in infrastructure-less scenarios. In this work, we apply wireless sensor networks devices to this problem that has hitherto received little attention. We analyze three points of the solution space. At one extreme, group membership information is *proactively* and collectively maintained by each node in the group. At the other extreme, the dissemination of group membership updates is triggered *reactively* by relying on a lower-level neighbor discovery protocol. In the middle lies a solution borrowing ideas from the two extremes. We compare our solutions through simulation of synthetic scenarios and real-world mobility traces of humans.

5.1 Introduction

The miniaturization fostered by wireless sensor network (WSN) enables scenarios where these tiny, untethered devices are carried by mobile entities. In these applications, the radio is often used as a sensor, enabling the detection of “contact” (i.e., physical proximity) among nodes, and in turn the study (or monitoring) of the dynamics of groups of mobile entities. The goal of this chapter is to design an efficient communication protocol providing applications with knowledge about who is member of a group at any given time.

Application examples. The research we present here was originally motivated by two projects we are involved in that, albeit targeting different application domains, share the common challenge of group monitoring:

- *Social care.* In the first project, WSNs are used in an Alzheimer’s daycare facility to monitor the patients’ activities. A concern of the caregivers is the safety of patients when outside the facility: they need to ensure that patients move together and be alerted immediately when some patient goes astray. Similar requirements are found in other contexts, e.g., shepherding or school trips.

[1]Earlier versions of this chapter appeared in [14, 15].

- *Wildlife monitoring.* In the second project we collaborate with biologists studying the social behavior of roe deer. Wildlife is studied either through direct observation or by using localization technologies such as GPS [12]. The former cannot be done on a large scale and is too invasive (or impractical) for some species. The latter is energy-hungry, requires a clear sky, and provides only an indirect measure of interaction: to study social groups [77] biologists need to know which animals spend time together, while GPS forces them to infer interaction from position traces—often quite sparse, to save energy.

The group membership problem. We assume that each of the potential group members carries a battery-powered WSN device (e.g., a TMote Sky [97]), consisting of a microcontroller, memory and storage capabilities, and a radio transceiver. In this context, a *group* is a set of mobile entities that are "in contact" with each other either directly or indirectly. In other words, it is the set of WSN nodes that are either in wireless range of each other, or for which a multi-hop path connecting them exists. Our objective is to identify who is *currently* part of a given group. The group composition changes over time, either because existing members leave or new members, possibly previously unknown, join the group.

Solving the problem when all the nodes are in direct communication among themselves, or towards a central node, is simple: it suffices for the WSN nodes to periodically broadcast a beacon announcing their presence. Otherwise, the problem is considerably more complex, but the application scenario is significantly richer and less constrained. For instance, caregivers need not be in direct communication with all patients: it is sufficient that they are in range of some, and that each patient is in range of at least another one. Similarly, a flock of animals may stretch over a considerable area, rendering impractical solutions that are centralized or rely on 1-hop connectivity. Therefore, we focus on a fully *decentralized* solution.

Because of mobility, a group may split, or separate groups may merge in a larger one. Ideally, nodes should learn about changes as soon as they occur. This, however, is unfeasible in a distributed system, and even more so in the highly dynamic, resource-constrained scenario we target. Therefore, we ensure instead that the group view of each node *eventually* mirrors the physical one. In other words, transient inconsistencies are allowed to occur, but these must disappear "as soon as" the network topology stops changing. In practice, we found this requirement to be sufficient, at least in the scenarios we target.

Finally, applications will use the group membership information differently. For instance, the case where one or more nodes disappear from the node's group view may trigger an alarm to the caregiver in the social care scenario, or be simply logged on flash memory in wildlife monitoring. These aspects are orthogonal to our contribution, and not mentioned further.

Roadmap and contribution. From our survey of related work in Section 5.2 it appears that no solution is immediately applicable to the above requirements. In this chapter we design and compare three protocols. At one extreme, Section 5.3 describes a solution in which nodes *proactively* and collectively refresh soft-state group information, based on vector clocks. At the other extreme, the protocol in Section 5.4, inspired by link state routing, *reactively* triggers the dissemination of group information based on a lower-level neighbor discovery layer. Finally, the protocol in Section 5.5, inspired by distance vector routing, attempts to strike a balance between the two.

The protocols we design build upon well-known techniques. Nevertheless, these are applied to a novel problem in a challenging context—mobile WSNs. Section 5.6 evaluates and compares these alternatives, implemented in Contiki [30], to determine their feasibility and tradeoffs. Section 5.7 discuss the problem of group monitoring from the perspective of RUTh, our neighbor discovery protocol. Finally, Section 5.8 ends the chapter with brief concluding remarks.

5.2 Related Work

Wireless sensor networks. In a 1-hop environment, the problem of group membership reduces to the simpler problem of neighbor discovery, already covered in the WSN literature. The works in [32, 57, 85, 116, 131] carefully schedule the broadcasting of beacons, the listening on the channel, and the radio power-down to obtain an energy-efficient duty cycle. We regard these protocols as a stepping stone to solving the group membership problem in a multi-hop environment.

The problem becomes considerably complex when nodes are to be discovered across several hops. This is because distinguishing between mobility (i.e., the node is still part of the group although with different links) and link or node failure is a non-trivial problem, impossible to solve with only local knowledge [55, 110]. These works suggest the use of gossiping or restricted flooding as a viable approach. In this chapter, we design dedicated protocols able to identify whether a moving node remains member of the same group, even if its links change because of mobility.

Distributed systems and ad-hoc networks. Group membership is a problem long studied by the distributed systems community [8], although with a slightly different focus. Indeed, the emphasis is on providing high-level abstractions enabling group communication, and therefore on their semantics and expressiveness in the presence of faulty processes [118].

In mobile ad hoc networks, somehow closer to our context, the main emphasis has been on unicast and multicast routing protocols. A notable exception is [102], which addresses how multi-hop group communication can be kept consistent with node movement. The authors introduce the concept of "safe distance", defined as the number of communication tasks a pair of nodes can complete in a given time and for a given mobility pattern. The paper extends this concept to multi-hop routes and presents protocols that allow a node to identify which are the safe groups it can communicate with. In contrast to these works, our emphasis is not on communication, rather on the simpler problem of providing each node with an up-to-date view of the current group, in a resource-constrained WSN.

Alternative devices. As we mentioned in the introduction, using the on-board radio is only one, and arguably the most recent, alternative. GPS has already been applied to study the interaction patterns of wildlife [12, 56]. However, GPS is energy-hungry: frequent positions locks do not play well with lightweight batteries. Moreover, GPS is limited to a clear-sky environment and inappropriate for burrowing animals. To this end, RFID based solutions [33] have been designed. However, RFID readers are expensive, both cost- and energy-wise; the technology is limited to small ranges; and finally, it is non-trivial to perform truly mobile sensing, as the latter happens only in the range of the, usually fixed, RFID readers.

In all these solutions, group membership is determined indirectly, typically through a post-deployment analysis of position traces and contact logs. A fundamental asset of our approach is that the protocols we describe next empower each node with a *run-time* global view of the group membership.

5.3 Using Logical Clocks

Our first protocol is based on a "soft-state" approach. Each node periodically advertises its presence. After a period when no advertisement is received, the advertising node is considered missing. We further refer to this solution as CLOCKS.

Our solution is inspired by vector clocks [84]. Each node maintains *i)* a local (logical) clock and *ii)* an array containing the (logical) clocks of all the nodes currently part of the group—the vector clock. Every node broadcasts the vector clock periodically with the same frequency, yet asynchronously. When a message is received, the incoming vector clock and the local one are merged to preserve only the largest timestamps. However, in our case, differently from the original formulation, a node's local clock is incremented periodically and not when the node receives or sends messages.

Node leaving and joining. To identify a member that left the group, a node compares its local clock against every entry in the vector clock. If the difference between the local clock and a vector clock entry crosses a predefined threshold, the corresponding node is deemed departed and the entry evicted from the vector clock, to save space. When a node joins the group, its local clock is appended to the vector clock. However, before this, all local clocks must be realigned.

Managing local clocks. The local clock of a node may accumulate enough offset w.r.t. those at other nodes to lead the former to incorrectly believe that some of the latter are departed. This is particularly troublesome for nodes joining with a "fresh", small value of the local clock. The problem is solved if clocks reflect absolute time. However, a time synchronization protocol (e.g., [82]) is not only expensive but also overkill. In our context, it is sufficient that, upon receiving a vector clock, the local clock is set to the largest among the clocks in the vector, therefore "realigning" the local clock with the logical time in the rest of the system.

Message failures. As all nodes broadcast periodically and indefinitely their vector clock, its delivery is guaranteed to be eventually performed at neighboring nodes. □

CLOCKS has a reasonable memory footprint: it maintains a vector of pairs ⟨node ID, clock⟩ that grows linearly with the network size. On the negative side, this entire data structure must be exchanged periodically. For big networks this may require fragmentation into multiple packets, increasing overhead. Moreover, vector clocks are exchanged even in absence of configuration changes, wasting communication. This last observation motivates the design choices of the next protocol.

5.4 Using Link State Information

At the other extreme, group membership information can be managed as "hard-state", i.e., a node considers the group composition as stable until it receives a message saying otherwise. In this protocol, called LINKS, periodic messages to refresh state are not required, and nodes communicate only upon group configuration changes. To detect these, nodes rely on a lower-level neighbor discovery service, notifying the presence of new neighbors or the departure of existing ones. LINKS is independent of the neighbor discovery layer: any of the protocols in [32, 57, 85, 116, 131] can be used. Nevertheless, these generate their own traffic, which must be accounted for in the energy expenditure. We come back to this issue in Section 5.6, where we take into account the overhead introduced by Contiki's neighbor discovery [31] used in our implementation.

We take inspiration from link state routing protocols such as OSPF [92] and build each node's group membership view using topology information received from other nodes. Each node maintains a *topology set* summarizing all links in the network. Nodes run a topological sort on this set to determine all reachable nodes, i.e., the group members. Figure 5.1 illustrates how changes in the topology are discovered and propagate. In the example we assume nodes are initially connected in a chain and know all network links (step 1). For now, we also assume reliable links: later in this section we describe how communication faults are dealt with.

Node leaving and joining. Whenever the underlying neighbor discovery layer notifies a neighbor departure, LINKS disseminates link information. For instance, in step 2 upon the break of the link between B and C, both nodes broadcast a link update ⟨#sequence, DROP, source, neighbor⟩.

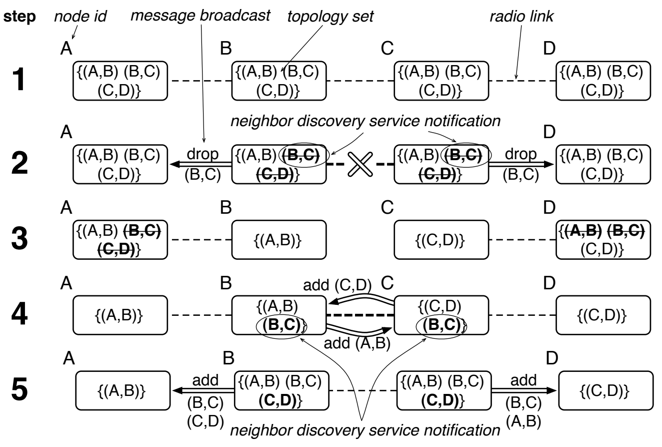


Figure 5.1: Operation of LINKS.

Upon receiving an update tuple, a node checks its topology set to identify links that must be removed, and waits of a short time interval, under the assumption that the reconfiguration may have generated additional updates. Then, it packs the performed topology changes in as few messages as possible and broadcasts them. Any other receiving node repeats the process. Updates travel far away from the node that observed the change: in step 3 this allows A to determine that C and D are no longer reachable, i.e., outside A's group.

Joining nodes are dealt with in a similar way. When a new neighbor is discovered, or a previously-dropped link is re-established as in step 4, the link end-points exchange updates in the form ⟨#sequence, ADD, source, neighbor⟩. In the figure B notifies C about the existence of the link (A, B), allowing C to reconstruct the full topology. After the buffering interval, nodes re-propagate all changes in their topology set, e.g., in step 5 C

broadcasts an update containing (A, B) and (B, C).

A node receiving an update may determine that no changes to the topology set are required. This happens, for instance, in the case where updates about the same links, triggered by different nodes, are received by the same node through distinct paths. In this case, the node does not re-propagate the update, therefore avoiding unnecessary communication.

Message failures. Unlike CLOCKS, in LINKS broadcasts are not idempotent and therefore cannot be sent repeatedly. Updates are generated and broadcast only once, i.e., when a change appears in a node's topology set. Unfortunately, losing an update may lead to permanent errors. For instance, imagine that the update ⟨#sequence, ADD, A, B⟩ in step 4 of Figure 5.1 is lost. As B will never re-broadcast the update, neither C nor D will become aware of A's existence.

To alleviate this problem, we implemented a positive acknowledgment scheme based on the update sequence numbers, ensuring reliable 1-hop broadcast. A node receiving an update (e.g., A in step 5) replies with a message containing pairs ⟨#sequence, ACK⟩ to all updates received in a predefined time window. If the update originator (B in this case) does not receive an acknowledgment from all of its neighbors in a given time frame, it repeats the operation for a limited number of times, hoping that the update is eventually received. □

LINKS enjoys the desirable property that communication is proportional to the number of group changes. However, the need for reliability comes at a high cost, as described in Section 5.6. This problem is overcome by the next protocol.

5.5 Using Distance Vectors

Our last protocol, called DIST, is inspired by distance vector routing protocols such as RIP [48]. Each node maintains a *distance vector*, i.e., a set of tuples ⟨node ID, next hop, minimum hop count⟩ for every other node in the WSN. This information is managed as hard state, similarly to LINKS, therefore communication is required only upon group changes. Moreover, these are detected by an underlying neighbor discovery layer, as we already described in Section 5.4 for LINKS.

We rely on the example in Figure 5.2 to illustrate the operation of DIST, initially assuming reliable links. In step 1 the network is partitioned: A is out of range w.r.t. the group formed by B, C, and D. Nodes in the latter group have already exchanged their distance vector, represented as an array for simplicity. For example, the one on B indicates that this node is 0 hops from itself, 1 hop from C, and 2 hops from D (reachable through C). The ∞ denotes that A is not a member of B's group.

Node joining. In step 2, node A approaches the group and the neighbor discovery layer notifies both A and B accordingly. The two nodes update and exchange their distance vectors in broadcast. B's broadcast reaches also C, which identifies A as a new group member, reachable through B over 2 hops. Next, C updates its own distance vector with the proper hop count and broadcasts it (step 3), enabling D to discover A.

Node leaving. When neighbor discovery notifies a node that a neighbor is no longer reachable, the node searches for alternative paths towards the departed node and the routes originally including it. If these exist, they include one of the remaining neighbors. If no alternative is found, the entry is purged from the distance vector. To find out, the node can either pull the distance vector from its neighbors or cache them locally. In our implementation, we chose the latter to limit complexity and communication overhead.

For instance, upon A's departure in step 4, neighbor discovery notifies A that B is no longer in direct radio range. As B was A's only link to C and D, no alternative paths to these nodes exists and therefore their entries are purged from A's distance vector. Similarly, B searches for alternative paths to A; its only option is a path through C. Still, this is invalid as B previously linked A and C. B is able to determine that the path is unsuitable by inspecting the next-hop field in the distance vector. Thus, B correctly purges A from its distance vector and broadcasts an update to its neighbors. The update allows C, after searching through the distance vector of its neighbors, to identify that no viable path to A exists. C prunes A from its distance vector and broadcasts an update (step 5). Eventually, D also evicts A, and the network reverts to the state in step 1.

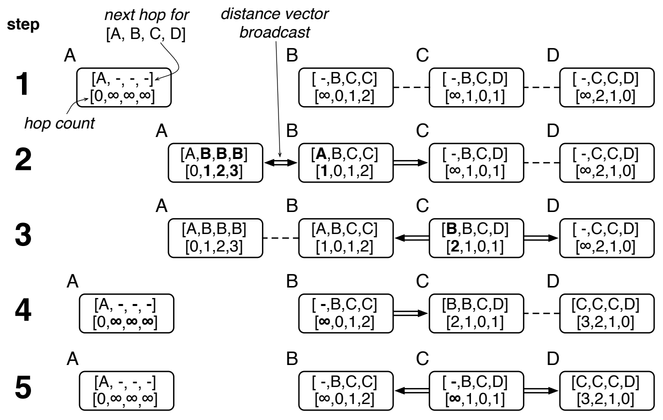


Figure 5.2: Operation of DIST.

Handling loops. To avoid counting to infinity when loops occur, DIST measures the hop count to a destination on the shortest path. For instance, in step 3 B discards C's broadcast due to its direct link to A, and correctly identifies A at 1 hop. Beyond a maximum hop count, a node is deemed unreachable.

More interesting is the case of finding alternative paths over routing loops. Imagine a ring configuration as in Figure 5.3. At some point in time, the link (A, B) fails. According to the previous reasoning, B advertises an infinite distance to A, forcing C to discover the alternative path to A that passes through D. C advertises the found path further, and thus B identifies A as being still a member of the group.

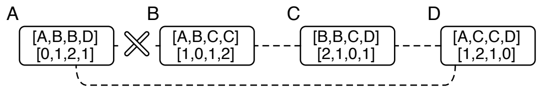


Figure 5.3: DIST: finding alternative paths over loops.

Message failures. For DIST to work properly, messages containing distance vectors cannot be lost. An easy solution is to retransmit them periodically, but all advantages over CLOCKS would be lost. However, unlike LINKS where updates are incremental, in DIST a *single* retransmission can make up for an earlier loss, restoring a node's entire group view. Based on this observation, we employ a negative acknowledgment scheme that guarantees that nodes *eventually* update their distance vector in the correct way.

A version number is associated to the distance vector of each node, incremented on every change and piggybacked on all broadcasts. Each node broadcasts periodically a *digest*, that is, a list of pairs ⟨node ID, last known version number⟩. When another node receives a pair containing its identifier, it compares the received version number to its own—the true one. The receiving node is then able to determine gaps in the sequence, denoting that the sender missed earlier updates. If and only if this is the case, a rebroadcast of the full distance vector is required to bring the outdated node on par.

5.6 Evaluation

In this section we evaluate and compare our three protocols, implemented in Contiki [30], and simulated using COOJA [94].

We perform the evaluation in two settings. The first one is a synthetic scenario that allows us to control precisely the mobility patterns inside groups. The results gathered in this scenario, described in Section 5.6.1, are then validated in Section 5.6.2 against simulations where the mobility patterns are instead derived from real-world GPS traces of humans.

Goals. Our first objective is to assess the impact each protocol bears on the energy budget. In principle, one could use LPL [96] or other duty-cycling mechanisms in conjunction with our protocols, however biasing the overall energy figures. We decided to avoid this bias by performing our experiments with the radio always on and without taking into account the contribution of idle listening. Therefore, we compare protocols in their "purest" form, by focusing solely on the cost of message exchanges, and consequently by measuring energy consumption indirectly through the total number of bytes transmitted and received by each node:

$$\text{energy} \triangleq V \times (I_{TX} \times \frac{\text{bits}_{TX}}{b} + I_{RX} \times \frac{\text{bits}_{RX}}{b})$$

For the popular TMote Sky platform we used in our experiments, $V = 3$ V, $I_{TX} = 21$ mA, $I_{RX} = 23$ mA, and $b = 250$ kbps [107]. For LINKS and DIST, our evaluation includes the traffic generated by the underlying neighbor discovery layer, in our case the one in Contiki's RIME stack [31].

Our second objective is to evaluate the protocols' accuracy, represented by two metrics: *i)* the *error*, i.e., the difference between the reported group size and the actual one— an instantaneous property whose value changes while protocols are detecting group changes; *ii)* the *detection latency*, i.e., the (average) delay between the group change and the time at which nodes become aware of it. The two metrics are related: given an initial error of zero, detection latency can be regarded as the time a protocol takes to bring the error back to zero after a group change occurs.

Parameters. Key to the configuration of all protocols is the neighbor discovery latency, i.e., the delay after which a node becomes aware of the arrival or departure of a neighbor, which in turn may trigger a group membership change. There are two concerns that must be taken into account, as follows.

First, while discovering a new neighbor is as simple as receiving a beacon from it, ascertaining a neighbor's departure is significantly more difficult. Indeed, missing a beacon from a neighbor can be caused by movement, node crashes, or packet losses. In practice,

Table 5.1: Protocol parameters.

Description	**Value**
Beaconing period (for neighbor discovery and vector clock exchange)	$b \in \{5, 10\}$ s
Timeout for declaring a neighbor missing	$\mathcal{T} = 120$ s
Radio range	30 m
CLOCKS	
Vector clock element size	2 B
LINKS	
Update tuple size	3 B
Minimum timespan between retransmissions	1.5 s
Maximum retransmissions	5
DIST	
Distance vector element size	3 B
Digest retransmission period	$\mathcal{D} \in \{30, 120\}$ s

neighbor discovery services declare a neighbor missing only after no beacons are received for a given predefined time interval $\mathcal{T}$. In the discovery service we used [31], $\mathcal{T}$ =120 s.

Second, the value of $\mathcal{T}$ affects the behavior of LINKS and DIST, which rely directly on neighbor discovery. However, in practice the configuration of CLOCKS is also affected by $\mathcal{T}$, to ensure that results are comparable. We configured CLOCKS to identify departed nodes after the same timeout $\mathcal{T} = 120$ s. Although the latter is managed as logical time, as described in Section 5.3, this choice strikes the best trade-off between a fair comparison and practical implementation concerns. We evaluate the impact of this value in more detail in Section 5.6.1. We also match the vector clock broadcast rate with the beacon rate of the neighbor discovery service, noted hereafter b. We evaluate scenarios with $b \in \{5, 10\}$ s.

The DIST protocol has an additional parameter, namely the period $\mathcal{D}$ with which the digests are broadcast in our negative acknowledgment reliability scheme. We evaluate two instances of DIST, corresponding to $\mathcal{D} \in \{30, 120\}$ s. We show the other simulation and protocol parameters in Table 5.1. The results for a given simulated scenario are averaged over 40 repetitions.

5.6.1 Synthetic Mobility Patterns

Energy Consumption

We simulate our protocols using a 25-node group moving in a single direction with a constant velocity (1 m/s) for 3500 s, long enough to retrieve interesting insights about the protocols' operation. Figure 5.4 illustrates our synthetic setting. The group is initially arranged in a grid configuration. Along the way, the group passes through a series of checkpoints (only one is illustrated). In between two consecutive checkpoints most nodes keep the same position within the group, except for a given, simulation-controlled number of nodes that swap positions, e.g., A and B. The swaps take place in between checkpoints, with the selected nodes moving at 1 m/s towards their target positions, and must complete

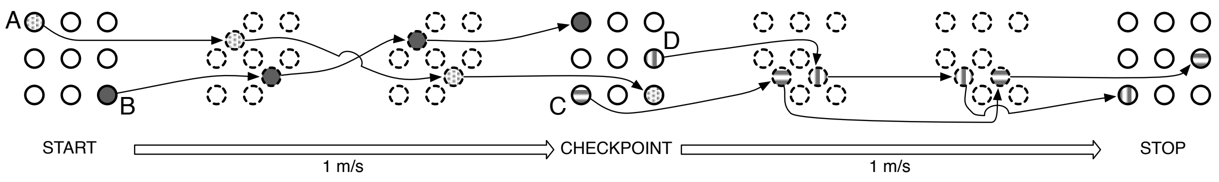


Figure 5.4: Our synthetic scenario for group movement.

at the destination checkpoint. At a checkpoint, a different set of nodes (e.g., C and D) are selected and begin swapping positions. Note that, in this scenario the group members change their positions within the group, but the group does not change its composition, i.e., no node joins or leaves the group. Indeed, our objective in this section is to analyze the impact of mobility alone on group maintenance. We analyze joining and leaving nodes in Section 5.6.1, as this bears a direct impact on the accuracy of our protocols.

Our synthetic scenario allows us to define a simple measure of the *churn* caused by mobility, as a combination of number of swaps and the distance between consecutive checkpoints:

$$\text{churn} \triangleq \frac{\text{number of position swaps}}{\text{distance between checkpoints}} \times 100$$

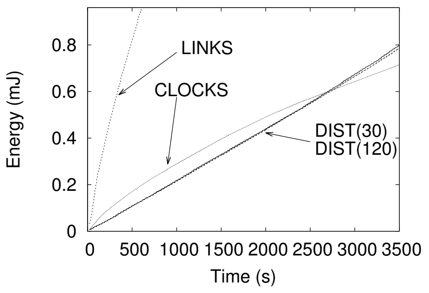
We setup simulations for all dimensions affecting churn. The first one is the distance between checkpoints, which we consider to be either 10, 50, or 100 m. The second dimension is the number of position swaps: we take into consideration groups without swaps and groups experiencing 1 to 5 swaps between checkpoints. The last dimension we explore is the network density, which we control through the relationship between the grid unit (distance between nodes) and the radio range that, as shown in Table 5.1, is fixed to 30 m. The configurations we simulate are:

- a *"sparse"* one where the grid unit of 15 m allows nodes to communicate 2 hops (grid units) away.
- a *"dense"* one where the grid unit of 8 m allows nodes to communicate 3 hops (grid units) away.

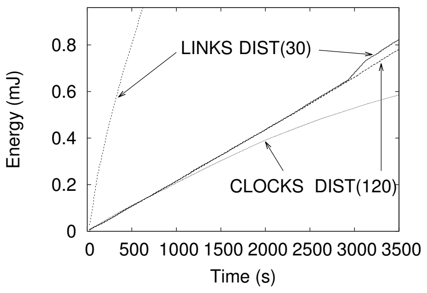
The combination of all these parameters yields 64 different simulated scenarios per protocol, each repeated 40 times.

Results. Figure 5.5 compares the protocols for churn = 20, corresponding to 2 position swaps every 10 m. In all scenarios, LINKS is the most energy-hungry among our protocols, especially in the case of a dense network. Logs show that the culprit for the high consumption is the acknowledgment scheme ensuring reliable broadcast. In fact, approximately 40% of the traffic is due to acknowledgments and retransmissions.

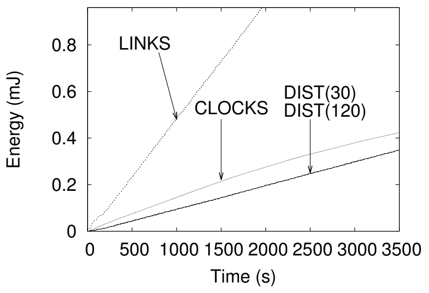
Figure 5.5 also shows that the impact of the b parameter is bigger in CLOCKS than in DIST. This, however, is expected. In CLOCKS, b represents the period with which vector clocks are exchanged, and thus bears a direct influence on traffic, and in turn energy consumption. Instead, in DIST b is the beacon period used for neighbor discovery, which is only a fraction of the total traffic: the update and recovery messages in the network



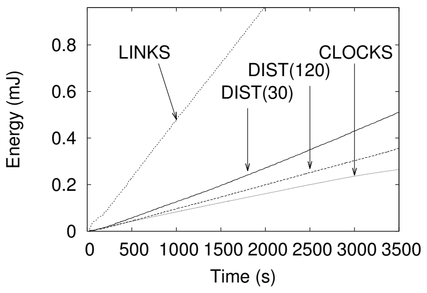
(a) dense network, b = 5 s.



(b) dense network, b = 10 s.



(c) sparse network, b = 5 s.



(d) sparse network, b = 10 s.

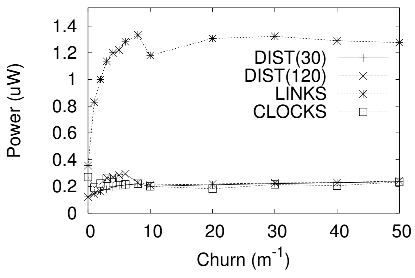
Figure 5.5: Cumulative energy for churn = 20, corresponding to 2 swaps every 10 m.

cannot be directly correlated to changes in the local neighborhood, yielding a reduced influence of b.

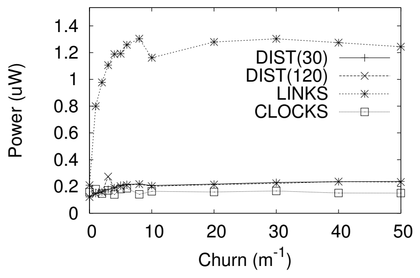
Figure 5.5 also highlights the impact of density on energy consumption. For the dense scenario, each broadcast (i.e., a neighbor discovery beacon or the transmission of a vector clock, link update, or distance vector) reaches more nodes and therefore consumes more battery w.r.t. sparser networks.

We now turn our attention to the impact of churn (i.e., mobility) on energy consumption. Instead of plotting the cumulative energy for all the aforementioned 64 scenarios covering all parameter combinations, we chose to analyze the slope $^{\Delta\text{energy}}/_{\Delta\text{time}}$ of the energy curves (i.e., the power consumed) as a function of churn, as shown in Figure 5.6. This approach is more concise, and captures effectively the impact of mobility on energy consumption. These charts report results only for the time interval between 500 and 3500 s, to exclude the initial traffic required to "bootstrap" a network with an empty group view. Also, note that the "humps" around churn = 10 (evident for LINKS but present also in the other protocols) is caused by the non-linearity of our definition of churn. For instance, churn = 8 may correspond to 4 swaps every 50 m, and churn = 10 to a single swap every 10 m.

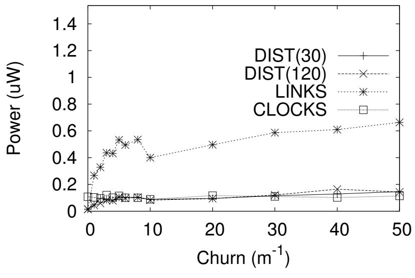
The charts in Figure 5.6 show that LINKS is greatly affected by churn even in a static dense group, and becomes a viable alternative only for quasi-static groups in the sparse configuration. In our dense configuration, the energy consumption of LINKS is dominated by the reliability scheme, due the increased likelihood of collisions. On the other hand, the



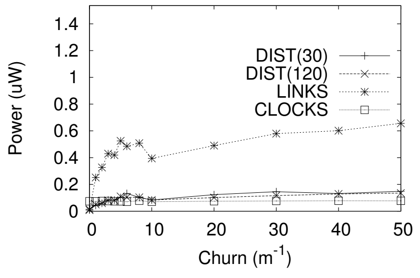
(a) dense network, $b = 5$ s.



(b) dense network, $b = 10$ s.



(c) sparse network, $b = 5$ s.



(d) sparse network, $b = 10$ s.

Figure 5.6: Power consumed as a function of churn.

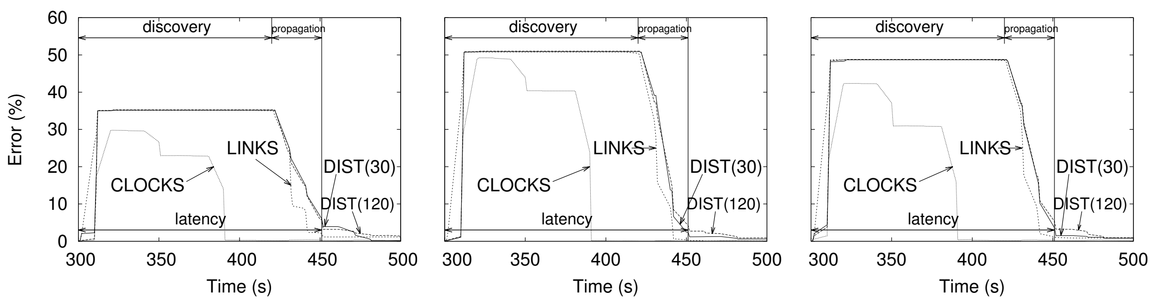
performance of Clocks and Dist is very similar for $b = 5$ s, where the short periodicity (of beacons and vector clock exchanges) affects directly the overhead. For $b = 10$ s, instead, Dist has an extra overhead caused by the exchange of distance vectors and their recovery, the latter clearly more marked in the dense scenario.

Error and Detection Latency

As mentioned at the beginning of Section 5.6, detecting node departure is more difficult than detecting node join. Consequently, we design our scenario around node departures and use it to measure accuracy. We start from a 25-node network displaced randomly in a 100×100 m^2 area, with a radio range of 30 m. The network bootstraps and identifies all group members. Then, at 300 s, a number of randomly-selected nodes are suddenly separated from the rest of the network, creating a new partition. This actually represents a worst-case w.r.t. the more realistic scenario where nodes move away, instead of being instantaneously relocated, because in the former case the departure can be discovered gradually. We perform simulations where 25%, 40%, and 60% of the nodes are separated from the main group.

Results. Figure 5.7 illustrates the accuracy of our protocols by showing *i)* how the error changes over time, and *ii)* the detection latency, i.e., the time necessary to reconstruct a correct group view on all nodes. We show only the experiments with $b = 10$ s, as those with $b = 5$ s yield similar results.

Links and Dist both rely on the neighbor discovery layer for detecting the departed



(a) 25% nodes leave. (b) 40% nodes leave. (c) 60% nodes leave.

Figure 5.7: Error and detection latency vs time (b = 10 s).

nodes. Nevertheless, as discussed earlier, this layer declares a neighbor as unavailable only upon a timeout, whose value in the RIME stack is $\mathcal{T}$ = 120 s. This explains the constant error in the initial part of curves: only at 420 s the nodes, alerted by the neighbor discovery layer, begin the propagation of information of the group change.

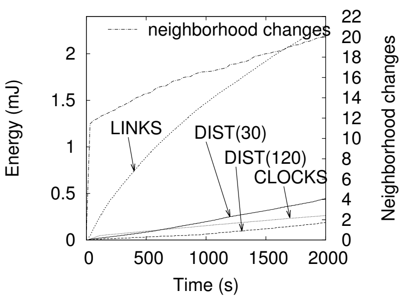
The same timeout is used for CLOCKS, but in this case there are two interesting differences: *i)* in LINKS and DIST, no action other than local neighbor discovery is taken until the timeout $\mathcal{T}$ expires. Instead, in CLOCKS the nodes continuously propagate their vector clocks with a period $b < \mathcal{T}$, therefore actively and continuously cooperating in reconstructing the global view; *ii)* recall that in CLOCKS the (logical) clocks in the network continuously realign themselves to the largest one. Therefore, it is as if "time runs faster" for CLOCKS, reaching the deadline set by $\mathcal{T}$ faster than in physical time.

The net effect of the considerations above is that, in all configurations of Figure 5.7, CLOCKS is much faster than the other protocols in converging to a consistent group view. However, this should not be taken as a direct comparison given that, in the absence of better choices, we use the same $\mathcal{T}$ = 120 s for both physical and logical time, and therefore a direct comparison is somewhat biased. In principle, one could set the value of $\mathcal{T}$ appropriately for LINKS and DIST, to reduce their initial "waiting time"; indeed we have simulations for $\mathcal{T}$ = 30 s which show a much shorter initial plateau for these protocols. Still, because of the first consideration above, during this time CLOCKS actively works to reconcile the global view: this is an intrinsic property of the protocol that may help speed up convergence w.r.t. LINKS and DIST when these use relatively high values of $\mathcal{T}$. Remember that if $\mathcal{T}$ is too small, packet losses are mistaken for neighbor departures.

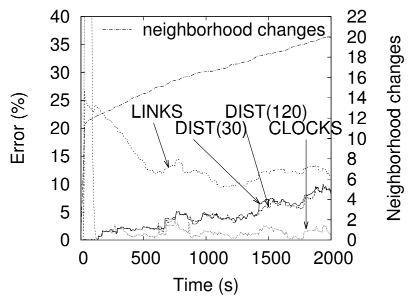
Figure 5.7 shows another interesting difference between CLOCKS and the other two protocols, in terms of how the network converges to the same global view, ultimately determining the group detection latency. Detection latency is a function of the number of hops in the network, but the delay at each hop depends on the protocol. In CLOCKS, the global view of all nodes is maintained in synchronous steps: as evidenced in the figure, convergence occurs by an alternation of plateaus (where nodes wait for $\mathcal{T}$ to expire) and bursts of node evictions. Instead, the other two protocols propagate the global information entirely asynchronously, at a faster pace but with longer tails.

5.6.2 Real-world GPS Traces

We now seek validation of the results in our synthetic scenarios through experiments on real-world GPS traces. We use the CRAWDAD KAIST data [101] representing daily records of students in a university campus over 3 months. We pick randomly 25 out of the 92 traces, and scale them into a 300×300 m^2 area. We use b = 10 s and a 30 m radio range. Due to the nature of the traces, the scenario is very challenging because *i)* nodes are continuously moving, i.e., the system never stabilizes; *ii)* nodes move of their own volition, not as a group: therefore, the ratio of joining and leaving nodes is much higher than in the applications we target.



(a) Cumulative energy.



(b) Error

Figure 5.8: Experiments with the real-world GPS traces in [101].

The energy consumption is reported in Figure 5.8(a) along with the average number of neighborhood changes. We notice trends similar to Figure 5.5: an excessive overhead for LINKS, similar performance of CLOCKS and DIST, a slightly better performance of DIST (120) due to the lighter recovery traffic.

More interesting is the error, reported in Figure 5.8(b) along with neighborhood changes. Because nodes are continuously moving at a high rate the error never stabilizes at zero, unlike the experiments in Section 5.6.1. This is also the reason why the detection latency is not shown, as it cannot be computed. Nevertheless, the chart is interesting because the three protocols exhibit very different behaviors. LINKS is able to reduce progressively the error and stabilize it around 10-15%. In contrast, both versions of DIST cannot keep up with the mobility rate, although they have the same b and $\mathcal{T}$ configuration of LINKS: their error appears to grow with time. The best performance is achieved by CLOCKS, which keeps the error very close to zero despite the high mobility.

5.7 Applying RUTh to Group Membership

In this section, we discuss the opportunity for integrating RUTh, our neighbor discovery protocol, with the protocols for monitoring group membership. At first, one can be tempted to employ RUTh as the foundation of one of the two reactive protocols, i.e., DIST and LINKS. Indeed according to their design, these protocols both exploit an underlying neighbor discovery protocol.

Unfortunately, neither RUTh, nor many state-of-art protocols can immediately be used as an underlying building block for complex communication demands. The reason is that most of the *neighbor discovery protocols focus on the time during which nodes are isolated.* Consider for instance, the behavior of RUTh and other similar protocols. At any point in time a node can either *i)* broadcast a beacon, *ii)* sample the channel for activity, *iii)* sleep. In this operation, a node can only receive data when it is sampling the channel. Intuitively, communication is not always possible and thus there is an inherent limitation on the protocols that can be built on top of neighbor discovery. This is of particular relevance to the group membership protocols that all exploit broadcasts as, for broadcasts to succeed, it requires all receiving nodes to be sampling at the same time.

If the higher-level application (e.g., maintenance of group membership information) requires communication that ultimately leverages on a MAC such as [96]. As most neighbor discovery protocols (and also RUTh) are implemented at the same level as MACs are, their operation is mutually exclusive. Therefore, once contacts are detected, *the radio operation must be switched from neighbor discovery to MAC mode*, although a form of periodic beacon broadcasts may be still required by protocols such as Links and Dist to manage one-hop neighborhoods.

Interestingly, as RUTh employs the same receive checks as the LPL MAC layer in TinyOS, we can view the mode switch as a mere extension of the RUTh intervals in which nodes are sampling the channel. Consequently, we can apply all the reasoning in Chapter 4 regarding the optimal detection latency and lifetime. Using the RUTh-specific notation, the extension of the sampling intervals implies that the duration of active is equal to the protocol epoch T. With this constraint, we can reason along the same lines with and modify the problem in Section 4.4.1 to determine the optimal configuration using the following formalism:

$$\begin{aligned} \text{minimize:} \quad & Q(\tau, s) \\ \text{subject to:} \quad & \textsc{active} = T \\ & \tau \geq \tau_{\min} \end{aligned}$$

In the previous problem, we select the configuration of $\langle \tau, s, T \rangle$ that guarantees all neighbors to be discovered within a latency $L = T$, minimizes the energy consumption Q and sets an additional constraint on the minimum beacon duration τ. The dual of the problem can also be applied, that is, we can imagine a configuration that trades-off detection latency to meet constraints on the node lifetime.

5.8 Discussion and Outlook

In this chapter we tackled the problem of monitoring groups of mobile nodes equipped with WSN devices, a problem that has hitherto received little attention. We presented and compared three protocols covering a big fraction of the solution space. Our study indicates that Clocks is resilient to the changes induced by mobility, has comparatively low energy demands and quick convergence time. Clocks is the protocol of choice if the sole goal of the network is to monitor groups. However, in many mobile WSN applications a neighbor discovery service is necessary for other purposes. For instance, in our applications it is used to detect proximity to hazards or log "contacts" with other mobile nodes. If this is the case, the Dist protocol allows one to build upon this already-present functionality with promising results: Dist has energy demands comparable to Clocks (and even lower for slowly-changing groups) and, if properly configured, can achieve similarly fast convergence, although this aspect must be studied further. This issue is already on our research agenda, along with a concrete test of CLOCKS and DIST in the real-world deployments we concisely discussed in Section 5.1, namely, the care of Alzheimer's patients, and the study of the social behavior of roe deer.

Chapter 6

Distributed Monitoring of Application Invariants[1]

Cyber-physical systems, including wireless sensor networks (WSNs), foster decentralization of sense-and-react systems, increasingly pushing the application intelligence inside the network. Therefore, the task of monitoring the system invariants to ensure safe behavior also becomes inherently distributed. In this chapter we present DICE, a system enabling WSN-based distributed monitoring of global invariants of physical processes. A DICE invariant is expressed as a first-order logic formula over the state of multiple sensing/actuating units, e.g., the expected state of actuators based on given sensed environmental conditions. The modular design of our system supports two alternatives for the distributed protocol detecting invariant violations, each targeting scenarios with different environment and network dynamics. We characterize and compare the two protocols using large-scale simulations and a real-world testbed. Our results indicate that invariant violations are detected in a timely and energy-efficient manner. For instance, in a 15-hop network, violations are detected in less than a second and with only a few packets sent by each node.

6.1 Introduction

The tight link between computational and environmental elements characterizing Cyber Physical Systems (CPS) fosters their use both to monitor and to drive physical control processes [111]. In this context, Wireless Sensor Networks (WSNs) can be used as a minimally-invasive tool to detect deviations from a specified "safe" behavior. Most often, this behavior is expressed by *global* invariants over the state of different computational units attached to sensors or actuators.

Example scenarios and motivation. Consider a demand-driven building ventilation system. Although it represents a setting in which nodes are fixed, this scenario represents a simple example advocating for global invariants. In this system, sensors feed periodic CO_2 and pressure readings to a Heating, Ventilation, and Air-conditioning Controller (HVAC). This operates fans in the building to maintain the inhabitants' comfort. The

[1]An early version of this chapter appeared in [27].

relation between the state of sensors and actuators dictated by control laws in the scenario above can be specified as an invariant:

> (**I4**) *When the CO_2 readings are above a threshold, there must be at least one active fan.*

Nevertheless, monitoring the correct HVAC operation is subject to further constraints. For instance, a non-uniform pressure profile in the building may indicate a malfunctioning HVAC. This condition can be checked as:

> (**I5**) *The difference in pressure between any two sampling points must remain below a threshold.*

The invariants in the scenarios above are *i) global*, i.e., they cannot be evaluated based on the state of a node alone, and *ii)* they possibly change over time.

Similar needs for distributed monitoring of global invariants are likely to arise in an increasing number of application scenarios (e.g., logistics, factory automation, and healthcare [111]). Some of these, such as the cold-chain management application illustrated in Chapter 3, entail mobile nodes. Currently-used RFID tags only allows tracking the location of the item at specific points along the supply chain. WSNs devices may enable continuous, fine-grained monitoring of the storage conditions, both during transportation and at warehouses. When traveling, the system shall autonomously monitor invariants stating, e.g., the correct operation of a refrigeration system in response to different environmental conditions. The invariants to monitor may change as a function of the specific transportation means along the supply chain.

CPS technology, and WSNs in particular, foster increasing degrees of decentralization in sense-and-react systems. As the application intelligence is increasingly distributed into the network, and empowered with increasing degrees of autonomy in affecting the environment, the mechanisms ensuring its correct operation under any circumstance and coping with various degrees of mobility must also become distributed. This latter vision motivates our work.

Contribution. We present DICE ("Distributed Invariant CheckEr"), a system to monitor global invariants in a distributed fashion, using WSN nodes. Solving this problem in general requires global knowledge of the system state [41] and is further complicated by the resource-scarce nature of WSNs. Therefore, we focus on invariants expressed as first-order logic formulae whose predicates, quantified over individual nodes, are however constrained to simple Boolean expressions or linear inequalities. This type of invariants strikes a balance between practical usefulness and resource limitations.

For this type of invariants, we identify the minimal set of global state elements needed for evaluation at each node—the *local view*. Based on this concept, we design, implement, and evaluate two distributed protocols for monitoring violations in scenarios with different environment and network dynamics, and with different degrees of redundancy in detection:

- our FLAT protocol allows *any* node to detect invariant violations, achieving increased failure tolerance against node crashes. It is particularly efficient for monitoring slowly-changing or mobile processes;

- on the contrary, our TREE protocol only allows a *subset* of nodes to detect invariant violations, but it does so very efficiently even in scenarios with high churn in application data. It explicitly targets scenarios in which nodes are fixed.

Both solutions realize invariant monitoring efficiently: in a 225-node network, both FLAT and TREE detect violations in less than a second and with only a few packets sent per node.

Roadmap. Section 6.2 describes the language to specify global invariants. Section 6.3 analyzes the types of invariants we target and defines the local view concept. We first tackle the problem from the perspective of fixed networks. Specifically, in Section 6.4 we leverage on the notion of local view to describe the two protocols in the context of static nodes. Section 6.5 illustrates the DICE tool-chain. Section 6.6 reports quantitatively on the performance and correctness of the two protocols in static networks, using both simulations to verify the behavior in large-scale networks and a lab testbed to profile the traffic patterns using real-world sensed data. Mobility comes back into play in Section 6.7, where we argue that although FLAT can cope with mobility, an integration with one of the group membership algorithms in Chapter 5 is desired in the case of highly dynamic networks; in the same section, we additionally report on results obtained while experimenting with such an integration on real motes. Section 6.8 contains a concise survey of related work. Section 6.9 ends the chapter with brief remarks.

6.2 Specifying Invariants

We describe the constructs of the declarative language, inspired by first-order logic, for specifying invariants in DICE.

Attributes. Physical processes are monitored through WSN nodes, which directly report either sensed environment data or the state of actuators through *attributes*. These are the link between the WSN node and the physical process under control. Each node is characterized by one or more attributes, each a typed mapping between a name and value. Different nodes can have different attributes. DICE supports three kinds of attributes. *Constant* attributes are set by the programmer and remain unchanged. With reference to the HVAC scenario,

```
attribute int type = FAN;
```

declares an attribute representing the type of node, in this case one controlling a fan. The value of *periodic* attributes, instead, is automatically updated by the system at a programmer-specified rate. For instance,

```
attribute int co2 every 3;
```

declares an attribute for a CO_2 reading, whose value is refreshed every 3 s. However, polling the value of slow-changing attributes can be inefficient. Therefore, we also allow declarations such as

```
attribute bool isActive on event;
```

where an update to the active status (e.g., of a fan) is triggered by the control logic, outside DICE, as discussed in Section 6.5.

Invariants. DICE invariants are first-order logic formulae whose variables are WSN nodes quantified universally or existentially. The `@` operator allows one to select which attribute of a given node is referred to in the invariant. As an example, the invariants in the introduction can be specified as follows:

```
invariant I4 {forall m: type@m = CO2 and co2@m > T
              -> exists n: type@n = FAN and isActive@n}

invariant I5 {forall m,n: pressure@m - pressure@n <T}
```

Besides the quantifiers `forall` and `exists`, the usual logical operators `and`, `or`, `not` also apply. Invariants can also refer to a specific node by using its identifier, as in `co2@52`. DICE invariants can have an arbitrary number of node variables, limited only by memory and packet sizes.

Transient violations. Short-term violations are inevitable in some scenarios. For instance, should I4 be applied on a per-room basis, a gathering of people for a scheduled meeting may trigger a violation before the fan is activated. However, if the HVAC operates correctly, the CO_2 readings should eventually return below `T`. Using the `tolerate` clause, DICE allows to specify transient deviations along two dimensions: time and repetitions. For instance,

```
invariant I4 {forall m: type@m = CO2 and co2@m > T
              -> exists n: type@n = FAN and isActive@n}
              tolerate 10 min for 5 times in 24 h;
```

is a variation of invariant I4 where the CO_2 readings are allowed to cross the threshold for at most 10 minutes, supposedly enough for the HVAC to react. However, this transient violation should not happen repeatedly, as this may indicate a failure in the control system. Thus, the invariant also specifies that transient violations are allowed at most five times per day, the expected frequency at which meetings occur in a room.

6.3 Monitoring Invariants: Local View

Global invariants are specified in DICE as a logical composition of predicates over node variables, that is:

$$Q_1 x_1, \dots Q_r x_r : P_1(x_1, \dots, x_r) \circ \dots \circ P_s(x_1, \dots, x_r)$$

where $Q_i \in \{\forall, \exists\}$ $(1 \leq i \leq r)$, $P_j(x_1, \dots, x_r)$ $(1 \leq j \leq s)$ is a predicate whose truth value depends on the attributes values at nodes $x_1, \dots, x_r$, and $\circ \in \{\wedge, \vee, \rightarrow\}$. We focus on two types of invariants characteristic of our target applications:

- in Type I invariants, each predicate involves only one node variable. An example is invariant I4, where the antecedent only involves node m, and the consequent only deals with node n. Type I invariants can be equivalently written as:

$$Q_1 x_1 : P_1(x_1) \circ \dots \circ Q_r x_r : P_s(x_r), \quad r = s.$$

- Type II invariants possibly involve multiple node variables, but there is only one predicate (i.e., $s = 1$), as in invariant I5 involving both node m and n. Type II invariants can be equivalently written as:

$$Q_1 x_1, \ldots, Q_r x_r : P(x_1, \ldots, x_r).$$

In practice, however, undesired deviations from specified behaviors are often expressed as comparisons against known thresholds [70], as in our motivating scenarios in Section 6.1. Therefore, we focus on invariants where $P(x_1, \ldots, x_r)$ is a Boolean predicate or an inequality $f(x_1, \ldots, x_r) < T$ or $f(x_1, \ldots, x_r) > T$, where f is a *linear* function of the attribute values at $x_1, \ldots, x_r$ and T is a numerical constant.

These two types of invariants cover a large fraction of the application landscape for DICE, including the motivating scenarios in the introduction. Moreover, as evident in the rest of the chapter, their monitoring already entails a significant degree of complexity. More sophisticated invariants (e.g., involving aggregates based on values at multiple nodes or non-linear expressions) are the subject of our ongoing work.

Our goal is to design a solution that detects violations if and only if they occur, as long as nodes do not fail. Detecting invariant violations is an instance of predicate detection in distributed systems, which generally requires global knowledge of the system state [41], and is therefore hard to achieve in the resource-scarce, unreliable setting of WSNs.

In this chapter, we reduce the amount of global information necessary at each node by properly defining its *local view*, i.e., the minimum amount of information enabling local detection of global violations. Next, we describe how the local view is populated at each node based on the two aforementioned invariant types. Maintenance of the local view at each node, however, must be complemented by an efficient dissemination of its relevant changes. This latter aspect, which in our approach is orthogonal to the former, is discussed in Section 6.4 where we illustrate two distributed protocols striking different trade-offs w.r.t. the characteristics of the monitored phenomena, the resilience to node and communication faults, and the fraction of nodes able to detect violations.

6.3.1 Type I Invariants

Intuition. For simpler illustration, consider a slight variation of invariant I4 where only universal quantifiers are used, i.e.:

$$(\forall m : \mathit{type}@m = \mathit{CO2} \ \wedge \ \mathit{co2}@m > T) \Rightarrow$$
$$(\forall n : \mathit{type}@n = \mathit{FAN} \ \Rightarrow \ \mathit{isActive}@n)$$

To detect violations of Type I invariants, we take inspiration from the notion of *communication silence* [132]. Every node attached to a CO_2 sensor does not send any information to other nodes as long as its reading is *above* T. The rest of the system implicitly takes the "collective silence" of CO_2 nodes as an indication that the predicate they monitor holds true. Similarly, nodes controlling a fan do not send any information as long as the fans are *active*, which implicitly indicates that every such node is currently operating the fan. Therefore, if the entire system remains silent, the invariant is complied with.

Whenever either the CO_2 reading drops below T or a fan becomes inactive, the corresponding node notifies this event. Breaking the silence reveals a change in the truth value of some predicate. Since the two predicates in I4 are joined by implication, the invariant is violated if a notification arrives from a node controlling a fan (i.e., there exists at least one inactive fan) and CO_2 nodes remain silent (i.e., their reading is still above threshold).

Algorithm. The technique above is applicable to any Type I invariant. We use timeouts to infer if a node monitoring a given predicate remains silent. Consider predicates of the form $\forall x_i : P_h(x_i)$. We identify the nodes where P_h is *not* false because of the constraints on constant attributes, e.g., a node monitoring $\mathit{type}@n = \mathit{FAN} \wedge \mathit{isActive}@n$. Every such node remains silent as long as P_h is true. Then, according to the logical operators connecting two predicates $\forall x_i : P_h(x_i)$ and $\forall x_j : P_k(x_j)$, a violation is detected when *i)* the two predicates are in disjunction and both node x_i and x_j send a notification; or *ii)* the two predicates are in conjunction and either node x_i or x_j sends a notification. In all other cases the invariant is complied with.

Existential quantifiers and logical implications are straightforwardly mapped to the cases above using known transformations of logical formulae. Hence, the local view for Type I invariants includes only a Boolean flag for every predicate in the invariant. The flag denotes if, since the last timeout, any node has notified the others that the predicate it is monitoring is no longer true.

Note that a node remaining silent may also indicate failure of that node. There is no remedy to this if the failed node is the only one allowing a violation to be detected. This is unlikely in the scenarios we target, where WSN nodes are expected to be deployed in large numbers, thus providing redundancy, and the monitored phenomena span significant portions of space, involving several nodes.

6.3.2 Type II Invariants

Intuition. To detect violations of invariant I5, one should consider all combinations of pressure readings at any two nodes. This is unnecessary if one identifies the *worst-case* combination, i.e., the two nodes corresponding to the highest pressure difference. If this is below the threshold, then the invariant is complied with, because any other pair of nodes has a smaller pressure difference. Otherwise, the invariant is violated. In the case of I5, the highest pressure difference is determined by the two nodes sensing the maximum and minimum pressure.

Algorithm. The intuition above can be generalized to any Type II invariant. Initially, let us consider universally quantified invariants $\forall x_1, \ldots, x_n : f(x_1, \ldots, x_n) < T$. First, we determine if the attributes at nodes $x_1, \ldots, x_n$ are positively correlated with f, i.e., if the evaluation of f increases when the attribute value increases. For instance, in I5 $\mathit{pressure}@m$ is positively correlated. Similarly, $\mathit{pressure}@n$ is negatively correlated, i.e., a decrease causes an increase in f. Based on this information, each node builds a local view containing the network-wide maximum (minimum) values for positively (negatively) correlated attributes. This is sufficient to determine an invariant violation. The same technique is straightforwardly applied when the invariant requires f to be *above* a threshold.

When node variables are existentially quantified, we flip the reasoning. Consider $\forall m, \exists n : x@m + y@n < T$. Given the network-wide maximum of x, this invariant is satisfied if we can find a node n where $x@m + y@n < T$. To determine the worst-case combination of nodes m and n, it is sufficient to identify the network-wide minimum value for y. If this value is such that $x@m + y@n \geq T$ when x is maximum, there exists no other node in the network where the value of y satisfies the invariant, therefore we detect a violation.

6.4 Local View Dissemination: Protocols

In this section we describe the design of two protocols for efficiently disseminating the local view updates enabling global invariant evaluation in DICE. Nevertheless, before we illustrate the protocols details we need to briefly put them in context w.r.t. the architecture of the DICE run-time, as local view dissemination depends on other system components.

6.4.1 Run-time architecture

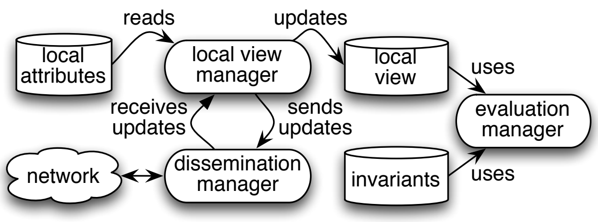


Figure 6.1: The architecture of the DICE run-time.

Figure 6.1 illustrates the main components of the DICE run-time. At the core is a data structure implementing the local view as defined in Section 6.3. For each attribute, the local view includes the set of values necessary to the evaluation of the corresponding invariants. Every value is associated to a tuple ⟨*name, value, source, timestamp*⟩ that binds the value to a specific attribute name. The source and timestamp are used to ensure correct update of the local view. The evaluation manager checks the local view against user-specified invariants, determining their violations. The local view manager determines the appropriate changes to the local view, based either on value changes of the local attributes, or updates received through the network. The latter are performed according to the protocol in the dissemination manager. A single protocol is used to disseminate local view updates, regardless of whether they belong to Type I or Type II invariants.

Evaluation manager. A change in the local view triggers the evaluation manager to check if any of the monitored invariant is violated. If the current local view indicates a violation and the invariant does not include a `tolerate` clause, the evaluation manager immediately generates a notification. Otherwise, the evaluation manager starts a timer with duration equal to the tolerance period. If a subsequent local view change brings the invariant back to compliance before the timer expires, no violation is notified. Otherwise, the violation is notified as if it were detected at the end of the tolerance period.

Local view manager. Changes to the local view are determined by the local view manager and are caused either by a local update (i.e., an attribute value change) or by a remote update (i.e., attribute value changes coming from the network). In turn, these changes may require further dissemination of updates. Figure 6.2 exemplifies the interplay of the local view manager and the dissemination manager, focusing on the local processing performed by the former on a given node. Here and in the rest of this section we focus

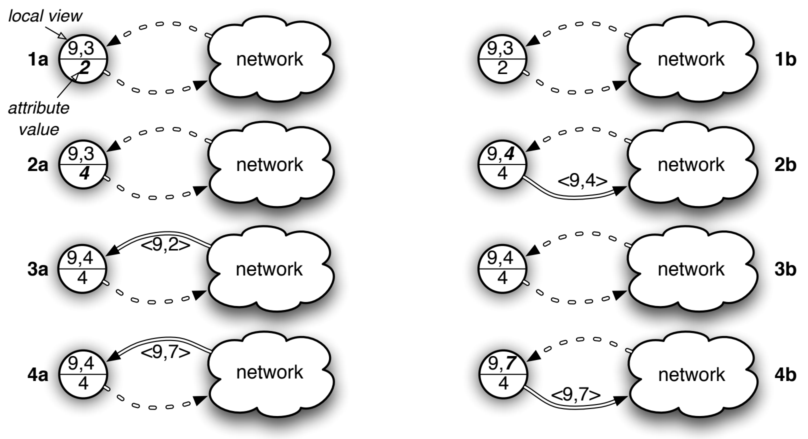


Figure 6.2: Local view processing for a invariant whose signature requires monitoring two network-wide maximum. Value changes are shown in bold.

on a sample monitored invariant $\forall m, n : x@m + x@n < T$, which requires a local view including the two network wide-maximum of x. Four cases are possible:

1. a local value update does not affect the current local view (1a). In this case, no further processing is needed. The local view on the other nodes can indeed remain the same, and no communication is required (1b);
2. a local value update must replace a value in the local view (2a). In this case, the local view is updated and propagated to the rest of the network through the dissemination manager (2b);
3. the values in a local view update from the network do not affect the current local view (3a). No further processing is required in this case, as the local view remains unaltered (3b);
4. a local view update from the network carries at least one value that must replace an entry in the current local view (4a). The new values are merged into the local view and the latter is disseminated to the other nodes (4b). Note that, in this case, the network has an inconsistent view on the global maximum, and further dissemination is required to converge.

Dissemination manager. The rest of this section illustrates two protocols to disseminate local view updates—i.e., determining how they are propagated inside the "network" bubbles of Figure 6.2. The FLAT protocol, described in Section 6.4.2, is designed to cope with failure-prone scenarios. In FLAT, all nodes can detect violations, achieving increased reliability through redundancy. The TREE protocol, described in Section 6.4.3, is optimized for scenarios with high variability in the application data. In these scenarios, TREE significantly reduces the communication overhead w.r.t. FLAT, at the expense of limiting the detection of violations only to a subset of nodes. Finally, Section 6.4.4 describes how these protocols deal with the challenges posed by asynchronous communication.

6.4.2 Flat

To allow any node to detect violations, all nodes must eventually agree on the most recent local view. To achieve this, we disseminate every local view update to the entire network. This functionality resembles well-known dissemination protocols [68, 69]. However, these are usually designed to propagate data from a *single* source, one data at a time. In DICE, every node is a possible source and multiple dissemination processes triggered at different nodes may overlap in time. This prevents off-the-shelf re-use of existing protocols.

To tackle the problem in our specific scenario, we adapt the polite gossiping technique [68]. The dissemination manager periodically broadcasts the current local view. It also receives local view updates from other nodes, storing them in a *network cache* containing the last received local view. As long as the network cache is the same as the local view, the broadcast period is exponentially increased up to a maximum τ_h to reduce traffic, as the network has reached convergence. Otherwise, the dissemination manager informs the local view manager of new data from the network. The local view manager determines whether the received information should be merged with the current local view, according to the processing described in Section 6.4.1. If so, re-propagation occurs with the broadcast period reset to the minimum τ_l, to speed up dissemination. A node suppresses the periodic broadcast if at least γ neighbors already broadcast the same information. τ_l, τ_h, and γ are protocol parameters.

Value insertion. We deal with concurrent dissemination processes from different nodes by merging local view updates as they propagate. Consider Figure 6.3 as an example, again with the monitored invariant $\forall m, n : x@m + x@n < T$. We assume all nodes have the same local view $\langle 9,\ 6\rangle$ in step 1. In step 2, a local value change at node A causes the propagation of an updated local view, which reaches node B and D. While these further rebroadcast the update, the value of x at F jumps to 8, as in step 3. The two updates originating at A and F "compete" for the propagation. We stop the propagation of the smaller update from A wherever the larger update from F has already been processed. This would happen at E, if the update from F is received before those from B or D. As a result, the larger value overcomes the other, providing eventual consistency of local views, as in step 4.

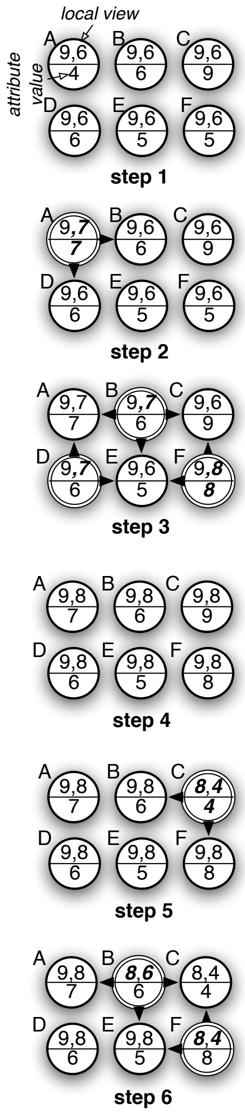


Figure 6.3: Disseminating local view updates in Flat.

Note that, if we were to use "as is" an existing dissemination protocol (e.g., DIP [69]) every node would disseminate either of the two new values in Figure 6.3. Indeed, these protocols are based on a globally-consistent version number. The dissemination of the two updates would occur with the same version number. Without the ability to process local view updates as they propagate, intermediate nodes would not discern which of the two values should be disseminated. To address inconsistencies, the source of the larger value may eventually recognize that its update did not propagate throughout the network, and repeat the dissemination. However, this causes higher network overhead.

Value deletion. The above processing is not sufficient for nodes whose state belongs to the local view. The last two steps of Figure 6.3 illustrate the problem. In step 5, the value of x at C changes from 9 to 4. Contrary to the previous case, here one of the values in the local view disappears as a consequence of a state change. In this case, based on local information, C determines that the new maximum are 8 and 4. If this local view at C, LV_C, were propagated as described above, the system would reach an inconsistent state. Indeed, B and F would infer from $LV_C = \langle\langle x, 8, F, t_4\rangle, \langle x, 4, C, t_5\rangle\rangle$, and $t_5 > t_4$ that the value of x at C dropped from 9 to 4. However, once this information is merged with their local views, yielding $LV_B = LV_F = \langle\langle x, 8, F, t_4\rangle, \langle x, 6, B, t_0\rangle\rangle$ and rebroadcast, the receiving A, D, E would obtain no information on the last update at C, and incorrectly conclude that $\langle x, 9, C, t_1\rangle$ ($t_1 > t_0$) is still valid.

We address the problem by appending an *eviction entry* to local view updates, enabling the removal of stale values. The entry is a tuple ⟨*attribute, source, timestamp*⟩ re-propagated at each hop along with the local view update. In our example, the eviction entry $\langle x, C, t_5\rangle$ allows nodes to determine that entry $\langle x, 9, C, t_1\rangle$ is no longer valid, and eventually converge to $\langle\langle x, 8, F, t_4\rangle, \langle x, 7, A, t_2\rangle\rangle$.

We use the periodic broadcasts of the current local view also to determine if a node is unreachable and its state should be evicted. When a node A misses a given number of broadcasts from a neighbor B contributing to the local view, A assumes that B crashed. Then, A recomputes its local view and propagates it along with an eviction entry, with the same structure and processing discussed earlier. In this case, however, the entry $\langle *, B, t\rangle$ causes the eviction of all attribute values provided by the crashed node. A node joining (or re-joining after failure) starts with an empty local view, eventually made consistent through the periodic broadcast process.

6.4.3 Tree

In scenarios characterized by high variability in application data, the Flat protocol is likely to exhibit a high communication overhead. Therefore, in the Tree protocol we leverage a tree-shaped routing topology to propagate local view updates. Note however that, unlike in data collection protocols, in our case the root is not physically attached to a base station. Indeed, the tree overlay is meant only to provide *structure* to an otherwise flat network, to improve on communication overhead; further, the root is actually expected to change, to provide load balancing and therefore increased WSN lifetime.

In Tree, each node pushes updates only upwards, towards the root, rather than to the entire network. As shown in Section 6.6, with frequent local view updates this yields a significant reduction in network traffic w.r.t. Flat. On the other hand, with this technique only the root of the tree is guaranteed to obtain the information necessary to detect violations. The other nodes may detect violations only whenever the local view updates determining the violation are generated within their own subtrees.

Value insertion and deletion. Figure 6.4 shows an example of update dissemination in Tree. The monitored invariant again requires a local view including the two network-wide maximum. Nodes maintain in their network cache one entry for each of their children, as illustrated for parent A and children B and C in step 1. This entry stores the most recent local view update from the corresponding child, representative of the state of the subtree rooted at it. In step 2, a local value change at node B causes a modification to B's local view, and the subsequent propagation towards the parent. Propagation occurs after a short timeout that allows nodes to process local view updates possibly coming from their own children, further reducing network overhead. Upon receiving the update, A rebuilds its local view based on the content of the network cache and the local attribute values, as in step 3, and propagates the update further, as the local view has changed.

The processing for deleting values from the local view is similar. In step 4, a local attribute value drops at node C. This causes a local view update at C, and the corresponding propagation towards the parent A. This again recomputes the local view based on the new content of the network cache and the local attribute values, as in step 5, and propagates the update further because of the changes in the local view.

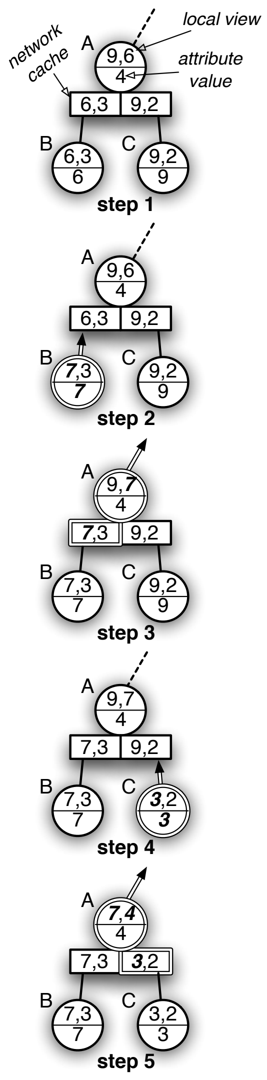


Figure 6.4: Disseminating local view updates in TREE.

The key difference, easily seen by comparing Figure 6.3 and 6.4, is that TREE does *not* bring convergence of the entire network to the same local view—quite the opposite. Indeed, local view updates are directed only towards the root, while in FLAT they spread along arbitrary paths in the network. This difference is key to the improvements in communication overhead enjoyed by TREE, which are however counter-balanced by the fact that only a subset of the nodes (possibly only the root) is guaranteed to detect violation. Moreover, the overlay topology used by TREE makes this protocol less robust in the presence of communication or node failures and induces an uneven load on nodes. Next, we illustrate the mechanisms we employ to tackle these issues.

Network cache consistency. Changes in the routing topology, packet losses, and node failures may render the content of the network cache no longer consistent with the system state. This may cause both false negatives, e.g., when a local view update reporting a new maximum is lost, and false positives, e.g., when a maximum drops and nodes higher in the tree miss the local view update because of a topology change.

We adopt a soft-state approach to maintain the consistency of network caches. Each cache entry is associated with an expiration timeout that causes its removal unless the node responsible for it refreshes the entry within the timeout. To further improve performance in case of changes in the routing topology, upon changing parent the child node sends an explicit eviction message to the former parent. If this is still reachable, the message causes the removal of the child's cache entry before the timeout expires.

An extreme case of node failure is the crash of the tree root, which is a single point of failure in TREE. Nevertheless, when the root stops acknowledging packets, the neighboring nodes can *locally* detect the crash and elect a new root, e.g., using existing procedures [37].

Load balancing. In TREE, the local view at every node filters the updates from the descendants based on aggregate information obtained from the corresponding subtree. As a result, nodes down in the tree are more likely to be relaying local view updates towards their parents, as their local views cover the system state to a lesser extent compared to nodes closer to the root, yielding an uneven load among WSN nodes.

We design a simple load balancing scheme to address this issue. Our scheme rotates the root role among the nodes to achieve a more even energy consumption. The decision to rotate is taken based on an estimation of the current energy budget of the network, as perceived at a given node. We periodically determine the node where this quantity is maximum, and hand over the root role to it.

To estimate the energy budget of a node n, we periodically compute such value as:

$$w_n(i+1) = \frac{1}{2} \cdot w_n(i) + \frac{1}{2} \cdot \frac{\sum_{m \in Neigh(n)} w_m(i)}{|Neigh(n)|} \tag{6.1}$$

where $w_n(i)$ is the energy budget at node n at the current step i and $Neigh(n)$ denotes the 1-hop neighbors of n. Observe that in Equation 6.1 every node contributes to the evaluation of $w_n(i+1)$ at every other node, as the formula is recursively applied to the entire network. We achieve this by periodically exchanging the current value $w_n(i)$ with the nodes in $Neigh(n)$ and computing the next value $w_n(i+1)$ until the metric becomes stable (i.e., $w_n(i+1) = w_n(i) \pm \varepsilon$, ε being an approximation constant) or we reach a maximum number of iterations. Note that each iteration requires only a single broadcast of the current $w_n(i)$ from each node n.

Interestingly, identifying the node with the largest stable value of the metric in Equation (6.1) can be achieved by reusing the same machinery we use to compute in-network the global minimum and maximum necessary to monitoring invariants. As a result, the current root eventually knows whether there exists another node n with a larger $w_n(i)$. If so, it hands over the root role to node n using an eventually-consistent dissemination protocol similar to [68]. The hand-over message also includes the last sequence number used by the former root to refresh the routing tree. When node n receives such notification, it starts building a new routing tree rooted at itself using a strictly greater sequence number. Based on this information, every other node in the network recognizes that a root change has taken place, and stops the propagation of the hand-over message.

6.4.4 Communication Delays

Communication delays may cause specific interleavings of local view updates that cause either protocol to miss some violations. Figure 6.5(a) illustrates the problem. Say that updates u_1 and u_3 yield a state violating the monitored invariant, whereas any other system state complies with it. Ideally, nodes A and C would propagate their updates as soon as they occur. In practice, however, the network stack may cause communication delays, e.g., because of collision avoidance mechanisms at the MAC layer. The intermediate node B may then receive the updates in the order u_1, u_2 and u_3. If we used only the most recent update from every node to evaluate the invariant, the system would not detect the violation.

Solution. We tackle the problem above with two combined mechanisms: *i)* a protocol maintaining a consistent ordering among timestamped local view updates, and *ii)* a circular buffer (*history*) storing recently-received updates, including those that have been superseded by new ones.

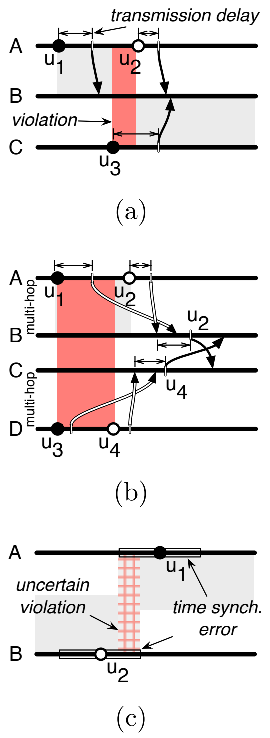


Figure 6.5: History buffer: motivation and limitations.

Our timestamp synchronization is inspired by [103], adapted by neglecting communication and network stack delays. At each hop, the data is sent along with the timestamp of generation and the sender's (physical) clock at send time. At the receiver, the latter is used to convert the data timestamp to the receiver's (physical) clock before inserting the data in the history buffer. With this information, a node can reconstruct the proper sequence of updates and detect violations. For instance, B in Figure 6.5(a) is able to match u_3 with u_1 stored in its history.

Limitations. This mechanism is effective only if there is at least one node with access to both u_1 and u_3, and therefore able to reconstruct the correct order causing the violation. This is not always the case in our protocols, which do not guarantee FIFO ordering on multi-hop paths. Consider Figure 6.5(b), where the monitored invariant is again violated only in the state following u_1 and u_3. The violation may go unnoticed if communication delays and update reordering cause the most recent updates u_2 and u_4 to supersede u_1 and u_3 at intermediate nodes B and C. As further re-propagation of the updates causing the violation is "quenched" at intermediate nodes, it may happen that no node in the network has enough information to detect the violation. However, addressing this problem would require either the propagation of the entire history, or stricter assumptions on communication delays and FIFO ordering—both too expensive in a WSN setting.

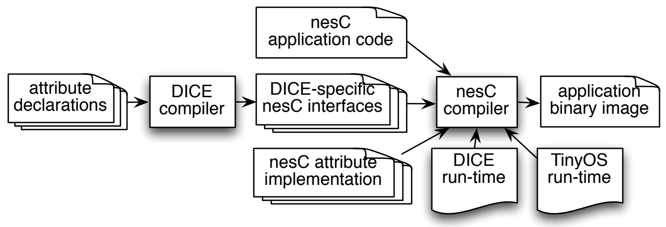
In addition, inaccuracies in timestamp synchronization may produce situations where given combinations of local states are *uncertain* [104]. Consider Figure 6.5(c). Assume the state of B before u_2 at physical time t_2 combined with the state of A after u_1 at physical time t_1 would violate the monitored invariant. If the timestamp synchronization error ε_{TS} is greater than $|t_2 - t_1|$, then it is not possible to determine the correct ordering of such updates when inserting them in the history buffer. Using our timestamp synchronization protocol, ε_{TS} may be as large as 200μs per hop [103], although protocols exist to reduce ε_{TS} to a few μs per hop [82]. In practice, this means that we cannot detect "instantaneous" violations that exist only for a few hundred milliseconds. However, these violations are not an issue in the scenarios we target where, as we already mentioned, transient violations of much longer duration are commonly accepted, and properly specified in DICE using the `tolerate` clause.

6.5 Implementation

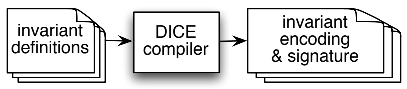
We describe the DICE tool-chain and report about the memory footprint of the run-time described in Section 6.4. Our prototype targets TinyOS [49], and relies on CTP [44] for the tree-based forwarding necessary to TREE. Nevertheless, the techniques we described thus far do not depend on either.

Tool-chain. In our example scenario, invariant I4 instructs DICE to check that the fan is active under certain conditions. However, the control logic actually operating the fan runs on the application code of the actuator node, external to DICE. Attributes are the *trait d'union* between DICE and the application, enabling the former to become aware about the status of the latter that is relevant to invariant monitoring.

Figure 6.6 outlines the tool-chain supporting this design. The DICE compiler generates one nesC interface for each attribute declaration, thus providing the connection between the source of attribute values and the DICE run-time. The latter accesses the interface through a compiler-generated component that periodically polls data from it or, for `on event` attributes, awaits the signaling of the corresponding events. The attribute interfaces, the components providing these interfaces, the DICE run-time, and the TinyOS libraries are input to the nesC compiler, which yields the binary image to upload on the nodes.



(a) Attributes and application.



(b) Invariants.

Figure 6.6: The DICE tool chain.

Invariant specifications do not require integration into a binary image. Thus, in principle they can be changed freely *without* reprogramming the WSN nodes. As shown in Figure 6.6(b), the compiler generates an invariant-specific encoding and an *invariant signature*. The former is essentially the postfix traversal of the invariant, with names replaced by numerical identifiers to reduce space. The latter indicates the minimal set of attributes necessary to evaluate the invariant, as explained in Section 6.3. This information is included in the local view and maintained by the run-time.

Memory overhead. The DICE system has a modular design, which allows the user to select which protocol to employ based on system and application constraints.

Table 6.1 shows the code memory used by DICE components. In terms of data memory, FLAT occupies ∼1.4 KB, while TREE occupies ∼4.2 KB. The larger data memory footprint required by TREE is caused by the buffers internal to CTP. These values are independent of the number and nature of the invariant monitored, each contributing an additional 10 B in the local view, 12 B in the network cache, and 14 B in the history buffer. The invariant specification itself is very compact thanks to the aforementioned encoding; for

Component	TREE	FLAT
local view manager	9.8 KB	
dissemination manager	7.4 KB	2.9 KB
load balancing	3.2 KB	n/a
history buffer	2.2 KB	
evaluation manager	6.6 KB	
Total	**28.7 KB**	**25.5 KB**

Table 6.1: Code memory usage of DICE components.

instance, the properties I4 and I5 in Section 6.2 occupy only 194 B.

6.6 Evaluation

In this section, we evaluate the performance and correctness of our DICE prototype. Section 6.6.1 focuses on TOSSIM simulations. These allow us to analyze and compare FLAT and TREE by relying on global knowledge of the network state. Moreover, using a simulator simplifies the assignment of value distributions to network nodes, and enables us to easily play with different scenarios. Finally, it allows us to analyze relatively large networks, up to 225 nodes, at the price of a less accurate representation of wireless transmission. Therefore, Section 6.6.2 analyzes data from a real deployment consisting of Telos motes placed in indoor and outdoor areas of our lab. This serves as a validation of our simulation results, albeit on a smaller scale, and as a real-world testing of our prototype.

Metrics. We focus on detection latency and communication overhead. As for the former, we define the *global detection latency* as the time difference between the instant when a state change triggering a violation *occurs* somewhere in the network, and the instant at which the violation is *first detected* anywhere in the network. From the application point of view, this indicates how fast DICE can detect a violation. However, other metrics are useful to evaluate the "internal" performance of detection. We define the *node detection latency* as the time difference between the instant when a state change triggering a violation occurs somewhere in the network, and the time at which a *given* node locally detects the violation. The average value of this metric is an indication of the speed at which the information necessary to evaluate invariants propagates through the network. Global and node detection latency are clearly related: the former coincides with the node detection latency of the *first* node recognizing the violation, easily computed as the minimum among all node detection latencies. The node detection latency can be computed only for those nodes that actually detect the violation—all in FLAT, typically only a fraction in TREE. Therefore, for the latter protocol we also evaluate the *detection count*, i.e., the number of nodes that detect the violation, as a measure of the redundancy of detection in TREE.

To investigate the overhead imposed on a node, we report on the *average number of local view changes per node* and the *average number of packets sent per node*, measured over the time span between the generation of the state change triggering a violation and the time at which the *last* node in the network detects it. These can be regarded as an indirect measure of the computational overhead and a direct measure of the communication overhead, respectively.

Correctness. In DICE, every state change triggers a brief period during which invariants may be incorrectly reported as violated or being complied with. This is the time required for the update to propagate throughout the network and is assessed as part of the aforementioned latency metrics.

Instead, our focus is on missed violations and false alarms. These can be generated by an incomplete propagation due to message loss. However, in Flat the periodic rebroadcast guarantees eventual delivery. In Tree, update propagation relies on CTP, which achieves a message delivery ratio up to 99.9% [44]. Therefore, we do not analyze this aspect further.

The more interesting case is instead one where an incorrect report about the invariant is caused by the reordering of updates inside the network, as discussed in Section 6.4.4, caused by an uneven propagation speed and a high churn in the attribute values. These conditions may reorder updates and cause indifferently a missed violation or a false alarm. For simplicity, in our evaluation we use scenarios that generate only the former.

6.6.1 Simulation Experiments

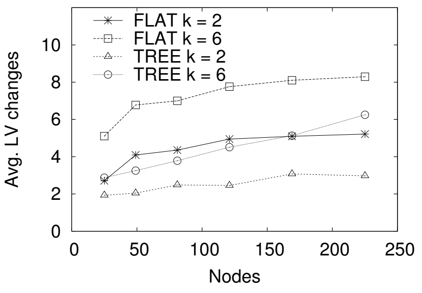
We analyze the behavior of DICE using synthetic distributions of attributes, and evaluate separately the performance of the update dissemination and the likelihood of missed violations.

As discussed in Section 6.3, the mechanics of detection depend on the nature of the invariant being monitored. However, the performance of Type I and Type II invariants is comparable, as they rely on the same underlying dissemination mechanism; they differ only in that the former can leverage "collective silence", minimizing communication overhead to a great extent. For this reason, in the following we consider only Type II invariants.

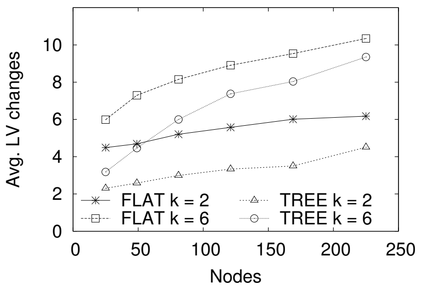
Nodes are arranged in a square grid[2], with an inter-node space of 10 m. We analyze network configurations ranging from 25 nodes to 225 nodes. Unless otherwise noted, in Tree the root is placed in one of the grid corners. Network connectivity is simulated using the LinkLayerModel tool in the TinyOS distribution.

We configured Flat and Tree as follows. The polite gossiping employed by the former uses $\tau_l = 200$ ms and $\tau_h = 4$ s as lower and upper bounds of the transmission period, and $\gamma = 5$ as the number of neighbors triggering a message suppression. In Tree, we use the default CTP parameters of the TinyOS distribution. Moreover, as mentioned in Section 6.4.3, Tree buffers incoming updates for a small interval, which we set to 200 ms (i.e., same as τ_l) to ensure that the minimum time an update is buffered at a node is the same for Flat and Tree. Unless otherwise noted, the results of each experiment are averaged over 50 repetitions.

[2]We also ran simulations with randomly-generated topologies. However, the many sources of randomness (topology, value distribution, timers, etc.) make them much less insightful.



(a) Gradient distribution.



(b) Random distribution.

Figure 6.8: Average local view changes per node.

Performance of Update Dissemination

Our protocols are influenced by the complexity of the invariant and specifically by its signature, determining the number of attribute values that must be present in the nodes' local views. To capture this aspect we use five invariants, collectively defined as:

$$\forall n_1, \ldots, n_k : \sum_{i=1}^{k} x@n_i < T, \qquad k \in \{2, 3, 4, 5, 6\}$$

As illustrated in Section 6.3, each invariant requires the local view to contain the k largest values of the x attribute. To avoid cluttering our charts, in the remainder we show only the results for the extremes, i.e., for $k \in \{2, 6\}$.

We consider two value distributions for x. The first is a 3-dimensional gradient (Figure 6.7). This distribution simulates a physical phenomenon (e.g., a heating source) where the values sensed in the range $[1, 10]$ are proportional to the inverse of the square of the distance from a source, placed at the center of the grid. As the grid and distribution are perfect, we obtain a set of concentric rings of nodes with the same value for x. In the second distribution, instead, each node assumes a random value in the range $[1, 100]$. Albeit somewhat more artificial, this distribution is interesting in that violations do not follow a pattern and can happen anywhere in the network.

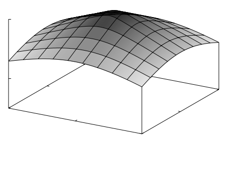


Figure 6.7: A gradient distribution for x.

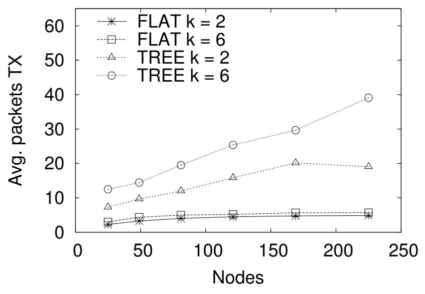
The simulations are executed as follows. In the initial state *i)* all nodes hold a value $x = 0$; *ii)* their local view reflects this global state; *iii)* the overlay used by TREE protocol is completely built. This initial, stable state is then perturbed with either of the aforementioned gradient and random distribution. The simulation ends when all nodes converge again to the same local view containing the new global maximum.

Unless otherwise noted, all simulations in this section are performed by using the setup above.

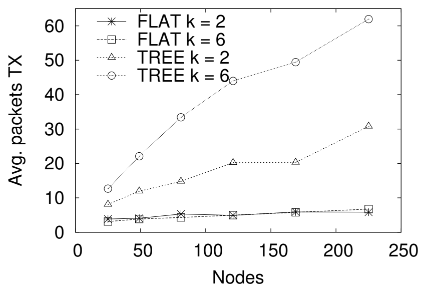
Local view changes and packets sent. As shown in Figure 6.8, this metric is slightly higher in FLAT w.r.t. TREE. Indeed, in the latter local view changes are aggregated as they travel upstream: at a parent node, a local view change from one child (representing the state of an entire sub-tree rooted at it) may be superseded by a local view from another child, and result in a single local view change propagated upstream. Instead, in Flat local view changes propagate in an unstructured fashion: a local view change may still "quench" another at a given node, but because view changes propagate along arbitrary paths, the same view change may reach other nodes along a different path and trigger other view changes.

Looking at local view changes alone, Tree would appear as the most convenient approach. However, tables turn when the average number of packets actually transmitted is considered, as shown in Figure 6.9. The higher value for Tree is determined by the messages necessary to maintain the network caches up-to-date and consistent—therefore, in essence, by the structure induced by Tree. In contrast, Flat structure-less dissemination of updates does not bear this overhead.

Impact of update rate. However, this holds only in the case where the frequency at which the attribute values change at each node is relatively low. This is evident in the experiment in Figure 6.10, where we simulate a 10 × 10 grid monitoring the invariant $\forall m, n : x@m + x@n < T$. Nodes start from a random value of attribute x. Periodically, a number of randomly-selected nodes change their value to obtain the rate on the x-axis, over a simulated time of 2 hours. Intuitively, if the update rate is low, one or more "sweeps" of the entire network are sufficient in Flat to inform all nodes about the new local view. If instead there are many concurrent changes, in Flat they propagate in an unstructured fashion, each potentially causing a new update and therefore a new dissemination competing against the others. This ultimately generates a traffic that grows linearly with the update rate, precisely due to Flat's inability to aggregate effectively the updates. In contrast, the structure provided by Tree, detrimental at low update rates, becomes an asset when the update rate increases: local view updates can be effectively aggregated in-network and over subtrees, therefore greatly limiting the increase of traffic.



(a) Gradient distribution.



(b) Random distribution.

Figure 6.9: Average number of sent packets per node.

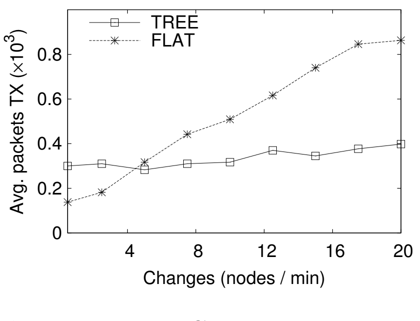
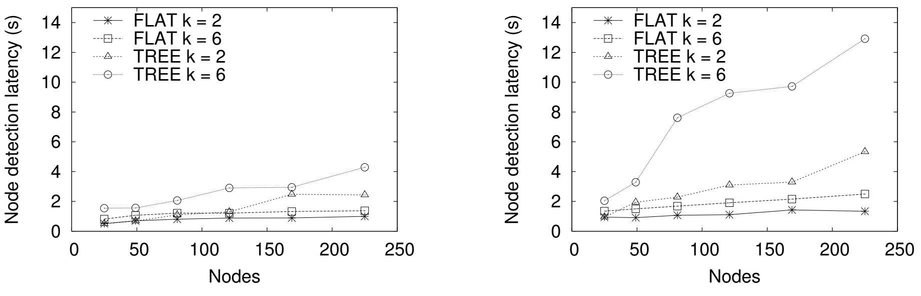


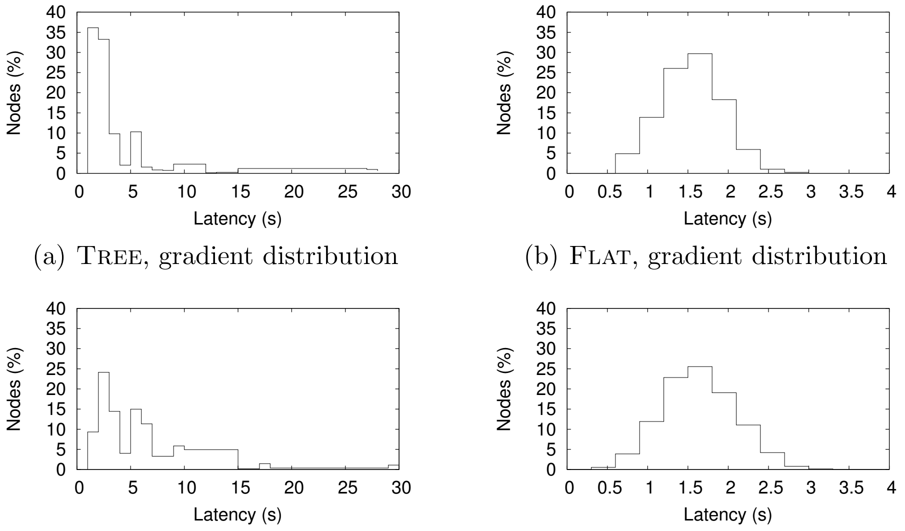
Figure 6.10: Communication overhead vs. update rate.

Detection latency. Figure 6.11 focuses on the average node detection latency. A first observation is that the relative performance of Flat and Tree is greatly affected by the distribution of attribute values: the behavior of Tree with the random distribution is significantly worse than Flat—an order of magnitude in the case of $k = 6$. Indeed, in the case of the gradient distribution, the attribute maximum are located at the center of



(a) Gradient distribution. (b) Random distribution.

Figure 6.11: Average node detection latency.



(a) Tree, gradient distribution (b) Flat, gradient distribution

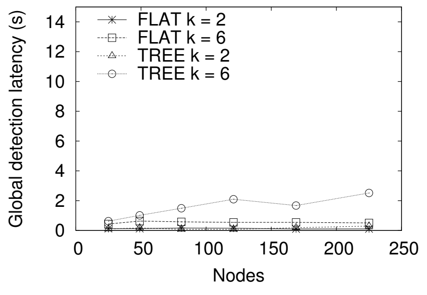
(c) Tree, random distribution (d) Flat, random distribution

Figure 6.12: Detection latency for 225 nodes, $k = 2$. Values on the x-axis have different scales.

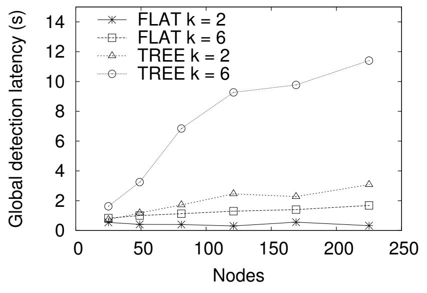
the grid; a violation can be easily detected by neighbors in either protocol. Instead, in the random distribution the maximum can be anywhere. The significantly higher latency of Tree is explained by the fact that, in this protocol, *i)* the likelihood of detection is higher in nodes that are closer to the root, and *ii)* the root can be far away from the nodes contributing to the detection. Instead, in Flat the propagation of maximum can be thought of as a "bubble", whose expansion (i.e., caused by the propagation of local view updates) is not restricted by an overlay as in Tree. As a consequence, the violation is detected as soon as the "bubbles" corresponding to the attribute maximum intersect at some node. This not only yields a smaller average node detection latency w.r.t. Tree, but also a different distribution of the detection latencies at each node, as shown in Figure 6.12 for both gradient and random distribution. In the case of Flat, the detection latency at each node resembles a Gaussian distribution, with relatively short tails. Instead, in Tree the distribution of latency is much more irregular: the nodes that are closer to the maximum (placed in the middle of the grid) detect the violation with a latency comparable to the slowest Flat nodes, but others may detect with a latency two orders of magnitude larger. Moreover, the difference when moving from a gradient to a random distribution is more marked in Tree than in Flat, especially w.r.t. the tails of the latency distribution.

In addition to the distribution of attribute values, which in practice is often unknown, the other parameter affecting the relative performance of the two protocols is the complexity of the monitored invariant, which in our experiments is represented by the value of k. As k increases, a node requires data from an increasing number of sources to perform detection. Therefore, the latency increases with k for both Flat and Tree, as shown in Figure 6.11. However, the impact of this parameter is significantly larger in the latter.

Finally, Figure 6.13 shows the global detection latency (i.e., the time to the first detection), while Figure 6.14 shows the ratio between the average node detection and the global detection. Under challenging conditions, such as invariants with high complexity or scenarios with a random distribution, it is generally more difficult to detect violations. This causes the global latency to be higher and closer to the node latency, as reflected by the low values, and constant ratio between the two, in Figure 6.14(a) for $k = 6$ and Figure 6.14(b) for all cases. Otherwise, if the invariant is simple or the changes more localized, as in Figure 6.14(a) for $k = 2$, the global latency is much smaller than the node latency, with the structure of the overlay inducing bigger differences in Tree.



(a) Gradient distribution.



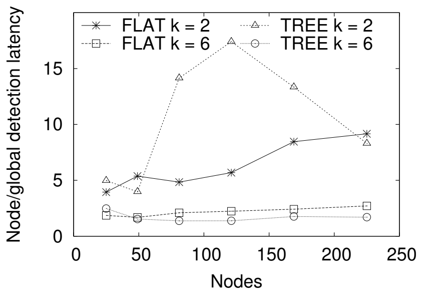
(b) Random distribution.

Figure 6.13: Global detection latency.

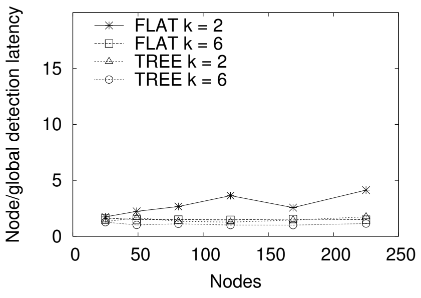
Detection count. Similar arguments explain the trends of detection count (i.e., the fraction of nodes that detect violation). As already mentioned in Section 6.6, this metric is meaningful only for Tree, as Flat guarantees eventual detection at all nodes. As seen in Figure 6.15, the higher the complexity of a invariant, the lower the detection count. The likelihood of a node performing detection increases closer to the root, and as k increases the phenomenon is exacerbated. A similar reasoning holds w.r.t. the attribute value distribution: the more scattered the values triggering a violation, the fewer the nodes that possess all the required information to enable detection, which explains the lower values for the random distribution.

Assessing the Risk of Missed Violations

In this work we aim at striking a balance between the desire to detect all violations, and the reality of distributed computing, aggravated by the resource-constrained scenario defined by WSNs. In Section 6.4.4 we discussed the compromises we make w.r.t. the potential of



(a) Gradient distribution.



(b) Random distribution.

Figure 6.14: Ratio between average node detection and global detection latency.

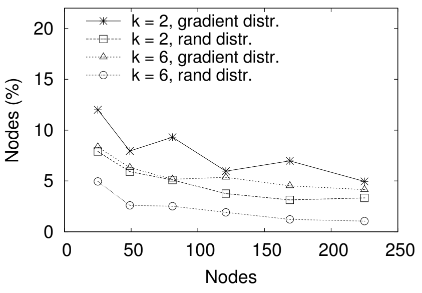
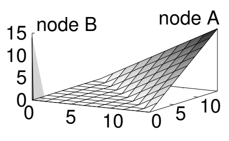
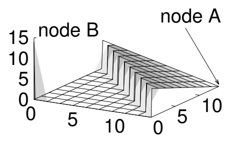


Figure 6.15: Nodes detecting violations in TREE.



(a) Initial distribution.



(b) Final distribution.

Figure 6.16: Distributions to assess the impact of history size.

missing a violation; here, we evaluate their usefulness. Firstly, we analyze the effectiveness of the history buffer in preventing missed violations, by reproducing a scenario similar to Figure 6.5(a). Secondly, we assess how DICE behaves in those situations where history buffers cannot help, similar to Figure 6.5(b). Finally, we evaluate the impact of node failures.

History buffers. We assess the extent to which history buffers help avoiding missed violations by monitoring $\forall m, n : x@m + x@n < 28$ in a 225-node grid. Nodes are initially assigned the distribution of x values in Figure 6.16(a), which violates the invariant due to the two nodes A and B, in opposite corners, with $x = 14$. After 500 ms (i.e., while the updates caused by the initial distribution are still propagating) this distribution is changed into the final one in Figure 6.16(b), which satisfies the invariant. Note that in both distributions a fraction of the nodes maintains the value of their attribute at $x = 0$. The change of distribution in the other nodes, instead, triggers a flood of updates. These updates are inserted in the history of all nodes, including A in the upper-right corner, on which we focus our experiment. A is also the root used in the simulations carried out for TREE. Our goal is to see how A can evaluate the invariant, and detect the violation, by combining its history with an older state of node B. Indeed, both A and B hold a value $x = 14$; however, the rapid change in distribution after only 500 ms occurs before B's value propagates all the way to A. The latter node can detect the violation only if its entry with $x = 14$ is still available in the history when B's $x = 14$ reaches A. Indeed, the high number of concurrent updates near A may fill A's history, flushing old entries.

In our experiments, the history size ranges from 25 to 50. For each size, we report the percentage of violations detected by A over 350 runs. To determine the extent to which concurrent updates render the history ineffective in preventing missed violations, we use different sizes for the upper-right triangle of Figure 6.16(a), ranging from 28 to 120 nodes.

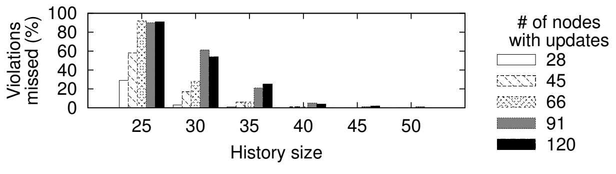


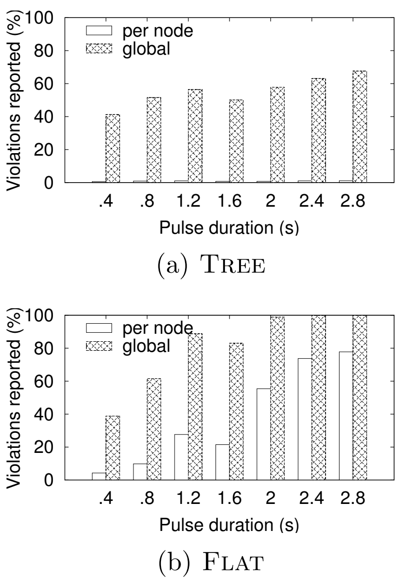
Figure 6.17: Impact of history size in FLAT.

Interestingly, even with a small history size of 25 elements, TREE manages to capture the violations triggered by all updates. The explanation lies in the shape of the tree topology built by TREE: the average number of children is 1, and the maximum is 5. Thus, each node receives data from a small number of neighbors, and therefore the risks of filling up the history buffer are lower. This is not valid for FLAT, where a node can overhear packets from several neighbors. However, Figure 6.17 shows that a history of 40 elements is already sufficient to bring the likelihood of missed violations below 4% in all cases, and a relatively small history of 50 elements guarantees detection even with a massive amount of updates, where more than half (120 out 225) of the nodes change their distribution. This confirms that the history buffer is effective in preventing missed violations.

Update reordering. The previous experiments demonstrates the ability of DICE to reconstruct a violation based on old values contained in the history. However, as mentioned in Section 6.4.4, this may not be enough when updates are propagated along non-FIFO, multi-hop paths; this may cause a more recent update to supersede an old one, as shown in Figure 6.5(b). Situations like these are more likely to happen if the violation persists for a short time; a long duration implies a re-propagation of updates, and therefore increased chances that they are received in the right order.

We reproduce these scenarios as follows. We simulate a 225-node grid monitoring $\forall m, n : x@m + x@n < 10$. The value distribution for x is such that all nodes are placed on a plateau at $x = 1$, except for two opposite vertexes at $x = 3$. As in Figure 6.5(b), we change simultaneously the value of x on both these vertexes, obtaining short "pulses" of $x = 6$ that cause the violation of the invariant. Between pulses, we set $x = 3$ for 3 s, during which the updates should propagate throughout the network. This choice yields a worst-case scenario where the number of hops in between the two vertexes is the largest possible, which increases the probability of losing updates due to their reordering along multi-hop paths. For TREE, the root was set to one of the corner nodes other than the two chosen vertexes. The input to experiments is the duration of the pulses. The output is the percentage of violations detected globally and per node. These are reported over 50 runs of 5-minute experiments, for a total of more than 3000 pulses simulated.

Figure 6.18 shows the results, distinguishing between the number of violations detected globally and the average of those detected by each node. The significantly better performance of FLAT is determined by its fully decentralized nature: because updates disseminate along multiple, arbitrary paths, although *some* nodes may miss some violations, others can catch them. Even when the pulse duration is 400 ms (i.e., twice the lowest polite gossiping period τ_l) 40% of the violations are detected by at least one node. With a pulse duration of 2 s, more likely to occur in real-world applications, Flat detects 100% of the violations. The worse performance of Tree is caused by its structure, which not only forces updates to propagate through predefined paths, but also yields larger detection latencies, as already pointed out. Latencies are also the cause for the lower average detection rate per node. Moreover, compared to the gradient scenario in Section 6.6.1, here the maximum are not necessarily close to each other, therefore latency is higher.



(a) Tree

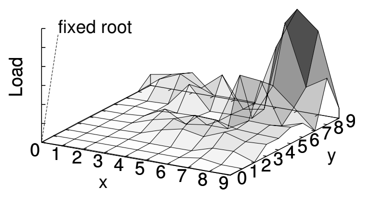
(b) Flat

Figure 6.18: Detection of fast violation pulses.

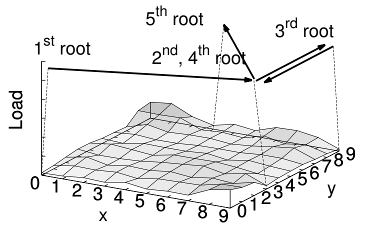
Node failures. The overlay exploited by Tree makes it more fragile in comparison with Flat's structure-less, intrinsically more resilient strategy. To quantify this aspect, we run experiments comparing the two protocols in the presence of node failures. On a 10×10 grid, we monitor the invariant $\forall m, n : x@m + x@n < T$. All nodes change their x to a random value every 2 minutes, over a simulated time of 2 hours. In Tree, the root was set in one of the grid corners. Every 2 minutes we crash two random nodes at the worst possible moment, i.e., right in between an attribute change and its reporting. Crashes are not recovered: the system reaches a point where the network is partitioned and no detection is possible. In this challenging scenario, Tree is able to detect violations only in 56% of the induced crashes. Instead, Flat is able to detect a violation in 84% of the induced crashes, thanks to the inherent resilience of its dissemination.

Load Balancing in Tree

In Section 6.4.3 we described a load balancing scheme for Tree to mitigate the effect that, in this protocol, nodes farther from the root bear a bigger fraction of the communication overhead. To evaluate our scheme, we simulate a network of 100 nodes deployed in a 10×10 grid. We monitor the invariant $\forall m, n : x@m + x@n < T$ for 5 simulated days. We randomly change the value of attribute x at each node every 2 minutes. To quantify the load, we count the number of bytes transmitted and received at every node, which approximates the corresponding energy consumption. Figure 6.19(a) shows the case where no load balancing is performed, and the root remains fixed at coordinates (0,0). The chart confirms the intuition that nodes far from the root bear a much higher load w.r.t. the others. Figure 6.19(b) shows, in the same scenario, the effect of changing the root at the end of each simulated day based on the scheme described in Section 6.4.3. The relative standard deviation of the load is 175% in Figure 6.19(a), and drops to 43% in Figure 6.19(b).



(a) Root is fixed at (0,0).



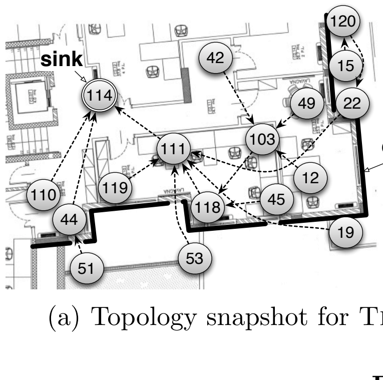
(b) Root is initially at (0,0), and moves according to our load balancing scheme.

Figure 6.19: Load distribution in TREE.

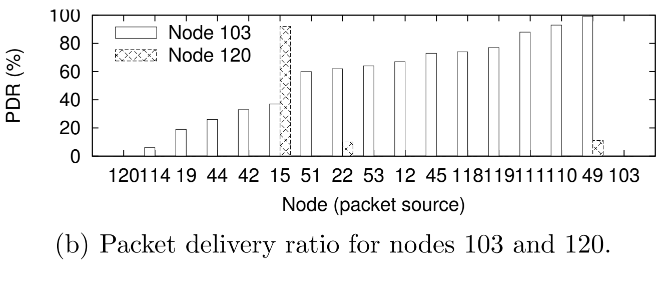
In principle, an even more uniform load could be achieved by increasing the frequency with which the metric w is computed, and therefore a new root is elected. However, this comes at two costs. First, the overhead of informing the next root of its new role. In the simulations of Figure 6.19, this averages to 1.7 packets per node for each hand-off. Second, the transfer of responsibility to the new root is not instantaneous. In our simulations, the time elapsed since the old root relinquished its role until all nodes join the tree set up by the new root is on average 1.2 s, and depends on the distance between the two roots. For instance, the transfer from (0,0) to (9,3) takes longer than the one from (9,3) to (9,9). While the tree is being reconfigured, packets containing local view updates may "wander" in the network; we counted an average of 0.44 per node during each root transfer. However, these are not lost: the underlying CTP protocol buffers each received packet at each hop, enabling their correct re-routing towards the new root as soon as its tree is set up.

6.6.2 Testbed Experiments

To analyze the traffic patterns and investigate the system behavior over time against the dynamics of real-world sensed data, we run a number of tests using 17 Telos motes deployed in a lab environment as shown in Figure 6.20(a). Every node periodically reports statistics to a node connected to a computer. We monitor the invariant $\forall m, n : temp@m - temp@n < 10°C$ where *temp* is the temperature, sampled every 2 s using the on-board SHT11 sensor. This rate is overkill for a slowly-changing phenomena such as temperature: we intentionally use it to stress our system. To avoid short-term oscillations in temperature values due to sensor inaccuracies, we fed DICE with a moving



(a) Topology snapshot for TREE.

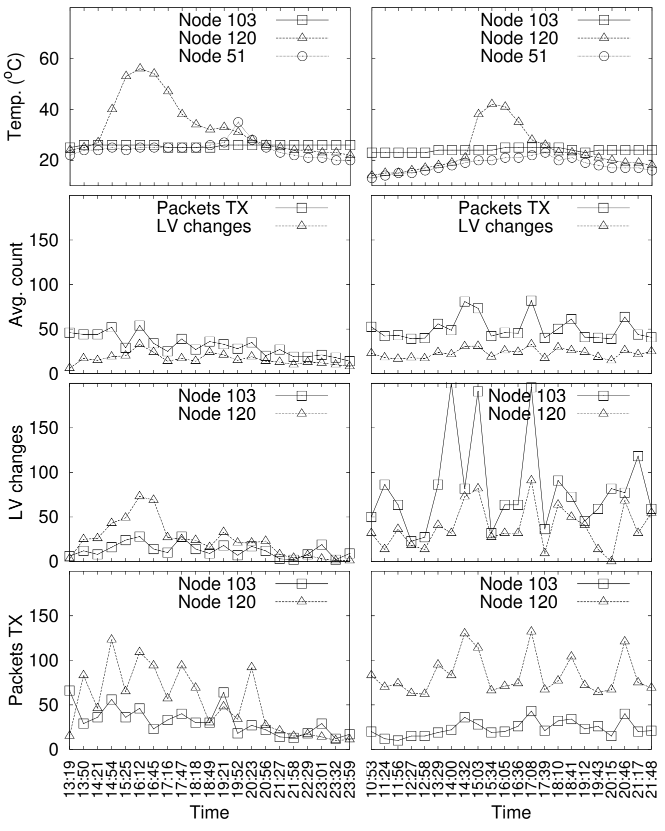


(b) Packet delivery ratio for nodes 103 and 120.

Figure 6.20: Laboratory testbed.

average over the three most recent readings of *temp*. We configured the protocols as mentioned in Section 6.6.1, and let the system run for about 11 hours.

Figure 6.21 illustrates the results we gathered. The charts in the top row show the temperature values at three nodes representative of different placements, yielding different temperature changes. As shown in Figure 6.20(a), node 103 is placed indoors; its reported temperature value is relatively constant. Node 120 is instead exposed to direct sunlight; its readings are greatly influenced by the time of the day. This and similarly-placed nodes are more likely to experience a sudden value change, large enough (w.r.t. nodes in other areas) to trigger violations. Finally, node 51 is almost always in shade, therefore usually reports the lowest temperature.



(a) Tree (b) Flat

Figure 6.21: Real-world results.

The charts in the two center rows of Figure 6.21 illustrate the system performance over time, plotting the local view changes and packets sent per node, aggregated over periods of 30 minutes. On average, we observe about one local view change every 3.75 minutes for Tree and 1.5 minutes for Flat. Therefore, as in our simulations, the number of local view changes is higher in Flat than in Tree. However, unlike in simulation, in this case the number of packets is higher than local view changes. The reason is that temperature changes very slowly, therefore the communication overhead is dominated by the keep-alive messages in both protocols.

A finer-grained perspective on the system dynamics is illustrated by the charts at the bottom of Figure 6.21, where we plot the metrics above at two nodes. Node 103 is in the center of the testbed, whereas node 120 is at its fringe. The snapshot of the routing topology[3] for Tree in Figure 6.20(a) shows that node 103 aggregates and reports data only from indoor nodes, with a relatively constant temperature curve. Additionally, the path from node 120 towards the sink does not include node 103. Thus, one would expect this node to have a lower number of local view changes than node 120, which is instead outdoor and experiencing a temperature increase. Interestingly, this is the case for Tree but not for Flat. The reason is that in Flat, as the packet delivery ratio in

[3]The topology is relatively stable, with an average of only 5.70 parent changes per node in the 11-hour experiment.

Figure 6.20(b) indicates, node 103 has more good neighbors and therefore more update sources, hence the higher number of local view updates. Instead, node 120 can receive updates from few sources, hence its lower number of local view changes.

Finally, the effect of polite gossiping in Flat can be observed by looking at the local view changes and packets transmitted for nodes 103 and 120. In the case of node 103, not every update corresponds to a packet transmitted. As this node has a larger number of neighbors, it is more likely to suppress its broadcast, unlike node 120. For the latter, in the absence of neighbors communicating redundant data, we can see that the number of packets transmitted is considerably higher than the number of local view updates.

6.7 DICE on Mobile Nodes

Originally, the DICE system and its underlying protocols were designed for scenarios in which nodes are fixed. Applications scenarios involving mobile nodes raised our interest and, consequently, we investigated opportunities of adapting DICE to this new, challenging environment.

In this respect, previous experiments [22, 23] show that CTP, the protocol empowering Tree, yields a high message loss when nodes are mobile nodes. Intuitively, the fundamental cause is that CTP imposes a rigid structure, as every node can only forward data to its parent. In a mobile environment, a moving parent means a broken link. Consequently, we dismissed from the start the idea of using Tree on mobile nodes.

Hereafter, we argue that we can take leverage on Flat to address scenarios in which nodes are mobile. In Section 6.7.1, we claim that the architecture in Figure 6.1 is sufficient to handle scenarios with limited mobility. If instead the application scenario is highly dynamic, Flat can take leverage from one of the group monitoring algorithms described in Chapter 5. We describe this integration in Section 6.7.2. In Section 6.7.3, we provide a framework which allows one to choose an approach according to the network mobility. Finally, in Section 6.7.4 we show results from early experiments on mobile WSNs.

6.7.1 Slowly Mobile Scenarios

Flat is not hampered by the need of maintaining a routing structure. In Trickle, the protocol underlying Flat, data is broadcast to any neighboring node which relays the data further. From this aspect, the identity of neighbors is irrelevant and thus they can change at any time. In this section, we show that a pure Flat solution, unchanged w.r.t. Section 6.4.1, can handle mobility.

Consider the example in Figure 6.22. Here, node C, holding the network-wide maximum, relocates to another position in the network. That is, C moves from the neighborhood of B and F (step 1), to the neighborhood of A and D (step 2). In Flat, B and F rely on C's periodic broadcasts to identify that C is alive; once the latter relocates and no more broadcasts are received, nodes B and F deem C disappeared. Thus, in step 2, nodes B and F remove C's entry, e.g., $\langle x, 9, C, t_1 \rangle$, from their local view and start disseminating an eviction $\langle x, C, t_2 \rangle$ with $t_2 > t_1$.

The eviction leads the network into a transient state where nodes have an inaccurate view on the global maximum, i.e., one which no longer contains $\langle x, 9, C, t_1 \rangle$. However, Trickle guarantees that the eviction *eventually* reaches node C and, once this happens, C will re-endorse itself as the network-wide maximum by broadcasting a new local view entry $\langle x, 9, C, t_3 \rangle$ where $t_3 > t_2$.

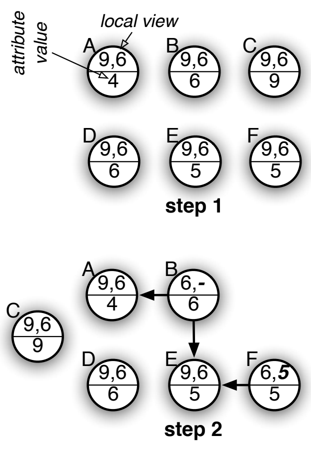


Figure 6.22: Example of node relocation in FLAT.

According to the protocol in Section 6.4.2, the Trickle timers are reset to the lower bound τ_l every time the local view is updated. Hence, in the scenario depicted in Figure 6.22, a network-wide traffic peak occurs upon C's movement. For sporadic node movements, this peak can be acceptable. However, all node movements trigger other evictions and, along with these, resets of the Trickle timers and more traffic peaks. When node movements are frequent, the solution described in this section is inefficient. To address frequent movements, nodes B and F must detect that C is still part of the network in step 2 (Figure 6.22) and thus avoid the eviction of C and the inherent traffic peaks. In the next section, we show that this is possible through an integration of DICE with one of the group monitoring algorithms presented in Chapter 5.

6.7.2 Using Group Monitoring in Dynamic Scenarios

From the previous section, we conclude that the (local) failure detection mechanism in FLAT described in Section 6.4.2 misleads the system into believing that mobile nodes have departed. In this section, we describe an architecture that integrates DICE with the work on the group membership problem tackled in Chapter 5. In this integration, the group membership algorithms play the role of global failure detectors and prevent traffic bursts in FLAT when nodes are mobile. Our focus is on the difference w.r.t. to the original DICE architecture illustrated in Figure 6.1.

Revisited run-time architecture. Figure 6.23 shows an updated DICE system, constrained to use FLAT in the dissemination manager and featuring a new component, the *group manager*. The latter is essentially an implementation of any of the group monitoring

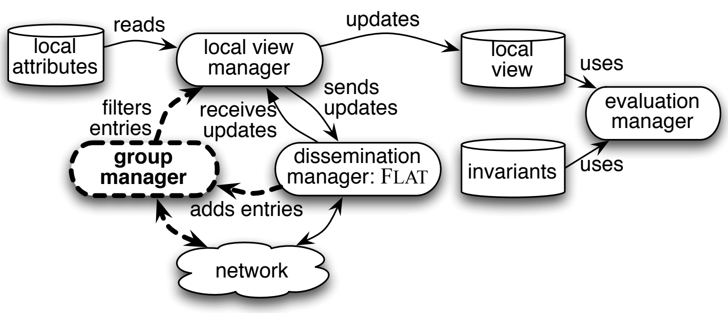


Figure 6.23: Updated DICE architecture integrating FLAT and group monitoring.

protocols described in Chapter 5.

The group manager lies at the same level of the dissemination manager. It exchanges messages with other nodes to maintain consistent information on the group membership, i.e., on the other nodes that form a transitively connected set. The local view manager uses group information to filter entries in the local view as follows:

- When the group manager detects the departure of a node, it sends an eviction notification to the local view manager. The latter drops all entries from the local view that belong to the departed node. The updated view is sent to the network through the dissemination manager. Notice however that neither eviction entries nor their propagation are needed; using the group manager, all nodes detect departures from the group without any information from the dissemination manager.

  The eviction notification from the group manager updates the local view only if the departed node contributes to the local view. Otherwise, no update occurs, and hence FLAT remains silent.

- When the local view manager receives an update from the dissemination manager, it first runs the update through the group manager. This allows the dissemination manager to determine whether the update originates on a departed node, case in which the update is dropped.

According to the second rule above, if the broadcast of a discovery beacon belonging to a new neighbor (used by the group monitoring algorithms) follows the broadcast of a local view update of the same neighbor (handled by the dissemination manager), any values belonging to the new neighbor are ignored until the group manager is updated. Although the group view will eventually be updated, we avoid latency penalties by updating the information on group membership with new discoveries made by the dissemination manager, as illustrated in Figure 6.23.

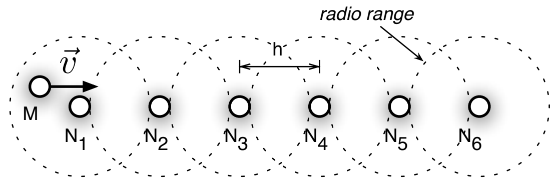
Limits of group monitoring. The group manager prevents incorrect evictions from the local view. This is achieveable only as long as the group membership information is accurate. Henceforth, we characterize the scenarios in which node mobility affects the ability of the group monitoring algorithms to provide correct information.

First, we characterize all group monitoring in Chapter 5 protocols using three high-level parameters, as follows:

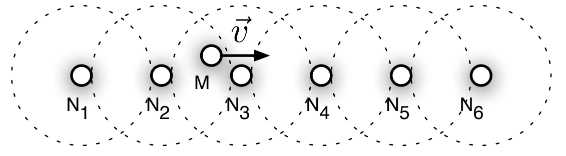
- *The neighbor discovery time b.* This is the time a node takes to detect that a new node entered its radio range. In practice, it corresponds to the vector clock broadcast period b for CLOCKS, while for LINKS and DIST, it is the beacon period b of the underlying discovery protocol.

- *The departure discovery time $\mathcal{T}$.* This is the maximum amount of time to declare a neighbor missing. In CLOCKS, $\mathcal{T}$ is dictated by a timestamp difference, while in LINKS and DIST, $\mathcal{T}$ is a characteristic of the underlying neighbor discovery protocol.

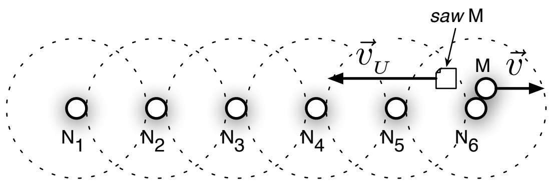
- *The propagation delay per hop t_{hop}.* This corresponds to the vector clock broadcast period in CLOCKS and to a small buffering timeout in LINKS and DIST.



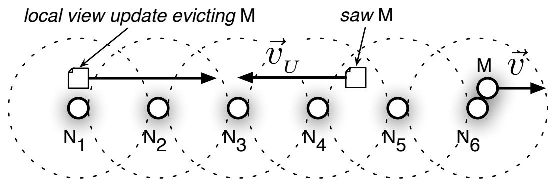
(a) M is first seen by N_1.



(b) M exits N_1's range. No other nodes detected its presence so far. N_1 starts the timeout $\mathcal{T}$.



(c) M is subsequently seen by N_6 which broadcasts an updated traveling at velocity $\vec{v}_u$.



(d) $\mathcal{T}$ expires at N_1 before the arrival of N_6's update. N_1 evicts M from the local view.

Figure 6.24: Scenario depicting the limitations of the group monitoring algorithms.

We use these parameters in the example scenario sketched in Figure 6.24. Here, a node M holding an attribute maximum travels at constant velocity $\vec{v}$ and passes through the range of a number of nodes N_i uniformly located on a line. This is the worst possible arrangement, as under this arrangement we have the largest network diameter and the smallest node redundancy. We assume that the distance h between adjacent nodes is equal to the (idealized) radio range.

In Figure 6.24(a), M is first detected by node N_1. From here, M continues its motion and exits N_1's range. Hence N_1 starts the timer to count $\mathcal{T}$. Meanwhile, as depicted in Figure 6.24(b), M moves in the vicinity of N_2 and N_3; M does not broadcast any beacon yet and thus it is not detected by N_2 and N_3. The next beacon of M occurs in Figure 6.24(c), when M enters N_6's range. The latter sends a notification traveling at velocity $\vec{v}_u$. Figure 6.24(d) illustrates the pitfall of this scenario: $\mathcal{T}$ can expire on N_1 before N_6's update reaches N_1. If this is the case, N_1 broadcasts an update that evicts M from the local view.

We claim that this is possible only if M travels at a considerable velocity. To support our claim, we generalize as follows. We assume that M is detected for the first time by node N_1, and the second time by a node N_k situated k hops from N_1 (in Figure 6.24, $k = 6$). As N_k detects M after b elapsed since N_1's discovery, we infer the following relation between M's velocity and the hop count k

$$k = \left\lfloor \frac{b \times v}{h} \right\rfloor$$

After the second discovery, N_k broadcasts an update that travels at velocity $\vec{v}_u = \frac{h}{t_{\text{hop}}}$ (Figure 6.24(c)). The condition under which N_1 does not evict M from the group is that N_k's update reaches N_1 before the expiration of $\mathcal{T}$. Thus

$$k \times t_{\text{hop}} < \mathcal{T}$$

From the previous two equations, we obtain the velocity threshold v_L at which M's movement does not cause an incorrect eviction. This is

$$v_L \triangleq \frac{\mathcal{T} \times h}{b \times t_{\text{hop}}}$$

Theoretically, for velocities smaller than v_L, the group membership is reported correctly at all times. Thus, there are no peaks in Flat traffic and properties are accurately monitored. In practice, the limit v_L is too high to be a challenge, even for reasonable configurations of the group monitoring algorithms. For instance, in Chapter 5 we tested Clocks with $b = t_{\text{hop}} = 5$ s and $\mathcal{T} = 30$ s. Under this configuration, assuming $h = 20$ m, the limit $v_L = 24$ m/s. Intuitively, the protocol is oblivious of a car travelling at 86 km/h.

The above reasoning is valid only in the absence of message loss. If the beacon that allows N_k to discover M (e.g., N_6 in Figure 6.24(c)) is lost, then the next discovery occurs when M is in the range of N_{2k}. If beacon loss is an isolated event, then M is wrongfully evicted and a Flat traffic peak occurs. If instead beacon loss is not an isolated event, the velocity threshold v_L decreases proportionally to the packet delivery ratio. For instance, if every other beacon is lost, the velocity threshold halves, i.e., $v_L = \frac{\mathcal{T} \times h}{2 \times b \times t_{\text{hop}}}$. We however do not foresee packet loss to be an issue: [7] shows that motes are able to exchange packets even when they travel at high velocity.

6.7.3 Choosing a Solution

We argued that unmodified Flat is sufficient to cope with mobile nodes at the expense of an increased packet overhead. We additionally showed an architecture in which group monitoring algorithms can be used to mitigate the traffic overhead. In this section, we compare the unmodified Flat against solutions using two of the group monitoring protocols described in , i.e., Clocks and Dist. We purposely do not include Links in this comparison due to its poor performance resulting from the evaluation in Chapter 5.

Moreover, in this section we focus on the traffic overhead of each approach. possible comparison dimension is the duration of the time window during which DICE is inconsistent. In this respect note that, when using Flat in conjunction with a group membership protocol, the network is always consistent — assuming the velocity of nodes is lower than the threshold v_L determined in the previous section. Instead, when using a pure Flat solution, the duration of the time window is given by the time took by an incorrect eviction entry to travel back and forth from the node generating the eviction and the evicted node (e.g., between nodes B or F and node C in Figure 6.22); this time is at most $2 \times d \times \tau_l$, where d is the network diameter and τ_l is the lower bound of the Trickle timers that dictate the propagation velocity of local view updates. Therefore, *i)* if consistency (and implicitly accuracy) is the greatest concern, a mixed solution Flat + Clocks or Flat + Dist is always preferable, and *ii)* to help choosing in the case when overhead has the bigger weight in the choice, we develop the framework that follows.

Assumptions. To evaluate overhead, we develop an analytical framework at the core of which lies a network of n nodes with a maximum diameter d. In our model, we assume that a node holding an attribute maximum (not necessarily the same every time) moves periodically from one edge to another edge of the network. We consider that the period between consecutive movements is a random variable with uniform distribution. This behavior corresponds to a homogeneous Poisson process; we assume that the rate of the process is λ. Moreover, we consider that the movement completes before the value of moving node is evicted from the local view by the node's former neighbors. We also assume that each packet has a fixed overhead of 13 B, as per the minimum frame size in [2].

For simplicity, we consider all nodes are static except those holding an attribute maximum. If for the unmodified FLAT and CLOCKS this assumption bears no impact, this is the best case scenario for DIST, where a reduced mobility implies a reduced overhead. Next, we show that, despite this simplifying assumption, the integration DICE + DIST is the most expensive among all three options.

Using simple FLAT. As discussed in Section 6.7.1, when using DICE without the integration with group monitoring, every movement of a maximum holder is followed by two phases:

1. The former neighbors of the moving node deem it missing. They insert an eviction entry in the local view, which travels across the network to reach the moving node in its new position.
2. The moving node detects the inaccurate evictions and disseminates a correcting update.

In the first phase, the eviction travels across the network between opposite edges. Thus, according to Corollary 1 in Annex B, approximately $\lg \sqrt[d+1]{(d+1)!}$ packets are sent during this time. Because the movement is periodic, the second phase is similar to a periodic update of the maximum in a static network. Thus, we can apply Corollary 2 to estimate the number of broadcast packets, that is, $\lg \frac{\lambda^{-1}+\tau_l}{2\tau_l}$. If the size of a local view update is $|LV|$, then the bit rate of FLAT is r_{FLAT}, where

$$\mathsf{r}_{\text{FLAT}} = \lambda \times (13 + |LV|) \times \left(\overbrace{\lg \sqrt[d+1]{(d+1)!}}^{\text{phase 1}} + \overbrace{\lg \frac{\lambda^{-1} + \tau_l}{2\tau_l}}^{\text{phase 2}} \right) \tag{6.2}$$

Using Clocks with DICE. CLOCKS entails a periodic broadcast of a vector clock, irrespective of node mobility. We assume that the broadcast period is b, and the size of a vector clock element is $|vc|$. Moreover, although CLOCKS prevents any local view update, FLAT entails a periodic traffic given by the upper Trickle bound τ_h. Following this reasoning, we obtain the bit rate $\mathsf{r}_{\text{CLOCKS}}$ of the mixed FLAT + CLOCKS solution as

$$\mathsf{r}_{\text{CLOCKS}} = \overbrace{\frac{13 + n \times |vc|}{b}}^{\text{vector clock}} + \overbrace{\frac{13 + |LV|}{\tau_h}}^{\text{local view}} \tag{6.3}$$

Using Dist with DICE. The neighbor discovery protocol under DIST entails a lightweight periodic broadcast which we assume to occur with period b. For every movement, there

Description	Notation	Value
Network size	n	25
Network diameter	d	5
Overhead per packet		13 B
FLAT		
Local view size	$\|LV\|$	14 B
Lower Trickle bound	τ_l	0.2 s
Upper Trickle bound	τ_h	30 s
CLOCKS		
Vector clock broadcast period	b	5 s
Vector clock entry size	$\|vc\|$	3 B
DIST		
Neighbor discovery broadcast period	b	5 s
Digest broadcast period	$\mathcal{D}$	30 s
Distance vector, digest entry size	$\|dv\|$	3 B

Table 6.2: Values used in the numerical comparison.

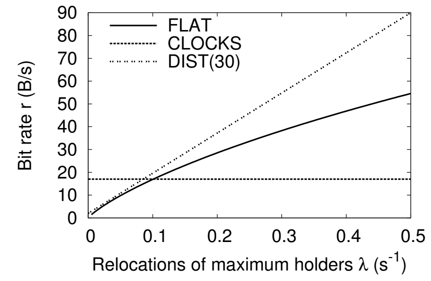


Figure 6.25: Impact of node mobility.

will be two updates of the distance vector, one triggered by the former neighbors of the moving nodes, the other by the new neighbors. Let $|dv|$ denote the size of a distance vector element. We additionally consider that a digest is broadcast every $\mathcal{D}$, that the average neighborhood size is $^n/_d$, and that each digest entry has size $|dv|$. Considering that the Trickle timers reach and remain stable at their upper bound τ_h, the bit rate of DIST is r_{DIST}, where

$$\mathsf{r}_{\text{DIST}} = \overbrace{\frac{13}{b}}^{\text{beacon}} + \overbrace{\lambda \times 2 \cdot (13 + n \times |dv|)}^{\text{distance vector}} + \overbrace{\frac{13d + |dv| \cdot n}{\mathcal{D} \cdot d}}^{\text{digest}} + \overbrace{\frac{13 + |LV|}{\tau_h}}^{\text{local view}} \tag{6.4}$$

Numerical comparison. Equations 6.2, 6.3 and 6.4 characterize the packet overhead in the presence of mobility. Hereafter, we use numerical values to compare the three solutions; in this respect, we re-use values that were employed in the previous evaluations of our protocols throughout this thesis. We summarize them in Table 6.2. The results are illustrated in Figure 6.25 depicting the variance of bit rate of all three solutions with the mobility of the nodes holding an attribute maximum.

Interestingly, the unmodified FLAT solution outperforms a mixed FLAT + DIST solution even in the case when node mobility is low. Intuitively, more traffic is needed to maintain an accurate view on the group membership than the traffic required by the unmodified FLAT to recover from errors induced by mobility. Furthermore, recall the assumption that only the nodes holding an attribute maximum are mobile. General node mobility only leads to increased traffic demands on DIST; comparatively, the pure FLAT solution is affected mainly by the mobility of nodes holding a maximum and not by the general node mobility. Consequently, we consider the complexity of the mixed FLAT + DIST solution unfeasible w.r.t. a pure FLAT solution. Instead, when comparing the unmodified FLAT against the mix FLAT + CLOCKS, we observe that, as expected, FLAT has lower traffic overhead in the case of low mobility, while the mixture FLAT + CLOCKS is preferable in the case of highly dynamic scenarios.

6.7.4 Mobile DICE in Action

The previous analysis identified the integration between CLOCKS and FLAT to be the most efficient solution for monitoring invariants on mobile nodes. In this section, we further investigate this integration, more specifically, we analyze the behavior of the integration on real motes. Our goal is to understand the relation between the number of group changes indicated by CLOCKS and the packet overhead in DICE. In this respect, we devised two scenarios, one involving real-world mobility and the second controlled node departures and joins. In both scenarios, all nodes monitor $\forall m, n : temp@m + temp@n < 60°\text{C}$ where *temp* is the moving average of the temperature computed using 3 samples collected in the past 6 s. We configured the Trickle timers in FLAT with $\tau_l = 200$ ms and $\tau_h = 30$ s. We configured CLOCKS so that the vector clock exchange occurs with period $b = 5$ s and neighbors are declared missing after $\mathcal{T} = 30$ s.

Mobile motes. In the first scenario, we deployed 15 nodes in our laboratory, as depicted by Figure 6.26. 8 of these were carried by the people in the lab at all times throughout the day; the remaining 7 were static and ensured connectivity between rooms. The result is a group of mobile nodes formed during office hours. Some of these move, as people often interacted one with another, and, at times, some members occasionally leave for breaks, meetings or other duties, and later re-join the group. At the end of the office hours, people were asked to leave on the desk the mote they carried; from this moment, the network underwent a period during which nodes were static. Albeit mobility in this scenario may look limited at first, it however mimics the ACube scenario describing the moving group in Chapter 1. Indeed, churn inside group — the main factor affecting the behavior of DICE — is perceived in a similar way by the network, irrespective of whether the inertial system (i.e., the group) is moving or remaining still.

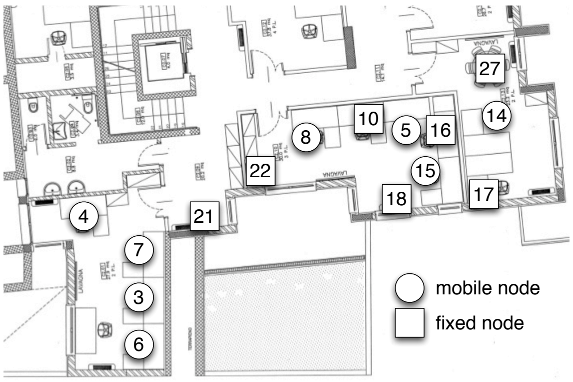


Figure 6.26: Deployment of mobile nodes.

The results, illustrated in Figure 6.27, were collected using two sniffing nodes deployed in the two main rooms of our laboratory. The first chart shows the number of variations in the group size as reported by CLOCKS. At 9:00 we observe an initial peak of 15 group changes corresponding to the system bootstrap. Around 10:30, the WiFi adapter of one of the laptops severely interfered with the WSN, effectively causing an intermittent connection between nodes 3, 4, 6 and 7 and the rest of the network. We also observe a small number of group changes around 22:30 and 2:30 when a number of nodes were reported missing. These reports can only be incorrect and caused by packet loss as the building is locked down and no person remains in the building past 22:00.

The second chart in Figure 6.27 shows the packet overhead. As expected, the number

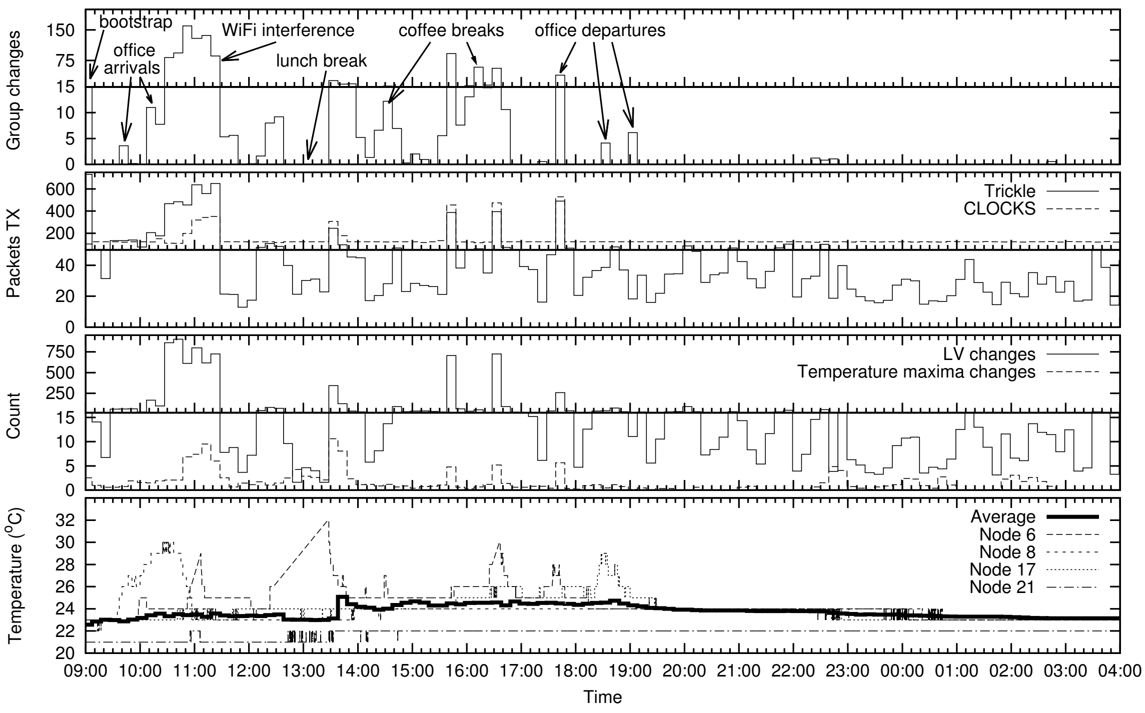


Figure 6.27: The behavior of DICE with Clocks on mobile nodes. The results show averages over 10 min steps. Note the double scale on the y axis which highlights the peaks for group changes, packets transmitted, local view and temperature changes.

of packets exchanged (containing vector clocks) by Clocks remains roughly constant at 120 throughout the experiment. Otherwise, we observe that while some peaks in the Trickle / Flat traffic corresponding to group changes do exist, these are not as heavy as compared to the behavior of the network during the night, essentially the time when nodes remain fix. This is however as expected; recall from Section 6.7.2 that the mobility of nodes changes the local view (depicted by the third chart) only when one of the nodes holding an attribute maximum leaves or joins the group.

During the night, when node mobility is no longer present, changes in the attribute values remained the only source for local view changes and, inherently, packet overhead. Indeed, changes of the temperature value result in a several peaks in the count of the transmitted packets and local view changes past 22:00. These persist throughout the night as in the fourth chart in Figure 6.27 we observe a constant temperature decrease after 20:00.

Our experiment indicates that DICE experiences the highest amount of traffic overhead in the presence of group joins or departures; this is the case, for instance, of the coffee breaks around 16:00. Otherwise, if no changes in the composition of the group occur, the traffic overhead is comparable to the overhead of a fixed network; in this respect consider, for example, the difference between the traffic at 15:00 and the traffic late at night. Moreover, according to the difference between the behavior during the night and the behavior during the day, changes in the group composition have a higher impact w.r.t.

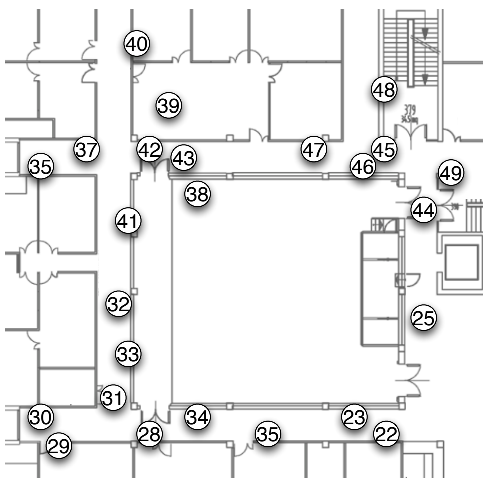


Figure 6.28: Testbed for controlled node departures.

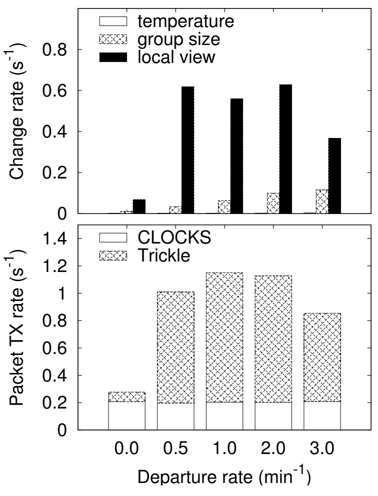


Figure 6.29: Results of controlled departures experiments.

to the changes of attribute values.

Controlled departures. The previous experiment highlighted the big impact of changes in the group composition of the network overhead. In the second experiment, we focus on this aspect, specifically, we design a scenario where we can exhibit control on the group change rate and analyze the resulting traffic. To this end, we used 25 nodes of the testbed deployed in an office building as per Figure 6.28. In our scenario, nodes join or leave the network periodically, however, the size of the network remains constant throughout time. To this end, we simulate departures by periodically turning off the radio on a number of randomly selected nodes. At every such simulated departure, we re-enable the radio of the nodes whose departure was previously simulated. In result, we achieve departure rates of $\{0, 1/2, 1, 2, 3\}$ per minute[4]. We repeat this procedure for approximately 1 h.

The results of our experiment are illustrated by the two charts in Figure 6.29. The chart on the top shows the average change rate for the sensed temperature, group and local view during the entire experiment. We observe that, as temperature changes are practically nonexistent, the main cause for local view updates is node departures / joins. Moreover, as expected, from the chart on the bottom, we notice that the fraction of transmitted packets due to CLOCKS is constant and that the number of Trickle / FLAT packets is similar to the number of local view changes.

Interesting however is that the number of local view changes is not proportional to the number of group changes. This is nonetheless as expected: a local view update is triggered only when a maximum joins or leaves the group; as in our experiments, the departing nodes are randomly selected, they do not necessarily hold a maximum and thus their departure does not reflect into a local view change. Therefore, the difference between various group change rates is not as pronounced as compared to the case in which no departure occurs. This reasoning also holds for the packet overhead, depicted by the

[4]To achieve the rate of 1/2 departures/min, we turn off the radio of 2 nodes every 4 min. For the remainder rates, we turn off the radio of 2, 4, respectively 6 nodes every 2 min.

chart on the bottom.

6.8 Related Work

The problem we tackle with DICE is reminiscent of predicate detection in distributed systems. In this field, seminal work investigates the detection of stable predicates [20], whose truth value changes only once in the system lifetime. Our work focuses instead on *unstable predicates* [41], that is, predicates whose truth value may change repeatedly. A particular class of unstable predicates are linear inequalities in the form $\sum_i x_i > K$, similar to Type II invariants. This class has been studied previously in [115]. In some respect, the algorithm in this latter work is analogous to ours, e.g., it identifies the largest values of x_i to decide whether the predicate is satisfied. However, the techniques employed by [115]—and by most literature on distributed predicate detection, including post-mortem analysis of WSNs [108]—are based on logical time, expensive and hardly applicable at run-time in WSNs.

In-network aggregation, the technique we employ to reduce traffic and collect network state, is widely referenced in the WSN literature. Various protocols, network overlays and summarization techniques have been proposed in this respect [25, 78, 93]. However, the general goal of these works focuses on data acquisition and traffic optimization, overlooking issues that are key in monitoring invariants (e.g., the consistency of the gathered state).

Moreover, declarative approaches have been proposed as programming [91] or debugging [13] abstractions for WSNs. TinyDB [79] is an example of the former, where an SQL-based, database-like abstraction simplifies querying data from a WSN. In contrast with TinyDB and similar programming-oriented approaches, which focus on providing construct to develop the core application functionality, DICE focuses instead on the complementary problem of ensuring that the latter behaves as intended. In this respect, the work closest to ours is the one on passive distributed assertions (PDA) [104]. As in our system, programmers specify the correct behavior of programs by using predicates that can be seen as a combination of Type I and Type II invariants. Nevertheless, the monitoring process occurs in a centralized manner, by relying on a *fixed* monitoring station *outside* the WSN, and based on global knowledge of the system state. The latter is acquired by using a secondary WSN deployed alongside the real one that sniffs and delivers packets to the monitoring station, which in many cases limits practical applicability. Moreover, to the best of our knowledge [91], we are among the first to propose a programming abstraction that explicitly deals with node mobility.

6.9 Discussion and Outlook

We presented DICE, a system for WSN-based distributed monitoring of global invariants in physical processes. DICE provides a declarative language to specify invariants and a run-time support enabling efficient monitoring of their violations. The run-time can be configured to use either the structure-less Flat protocol or the Tree protocol, which instead relies on an overlay.

FLAT provides increased fault tolerance by allowing *any* node to detect a violation, at the expense of increased overhead in scenarios with high rates of changes in the monitored application state. Nevertheless, the nature of FLAT permits the operation of DICE on top of mobile networks. In this respect, a pure FLAT solution efficiently addresses networks in which node mobility is reduced. If however, mobility is complex, it is advisable to use FLAT in conjunction with one of the group membership algorithms described in Chapter 5, the latter playing the role of a global failure detector.

Instead, TREE provides improved performance for fixed scenarios in which the change rate of application state is high by structuring and optimizing the dissemination of relevant state changes, but this very structure makes the approach more complex, as it requires additional mechanisms to preserve structure in the presence of failures, and limits the ability to detect violation only to a subset of the nodes.

In perspective, we see this work as a building block for more complex functionality. We are already working to enable the specification of invariants involving aggregation over subsets of nodes, e.g., the sensor reading of each node must not deviate too much from the global average. On the other hand, our approach is naturally complemented by language constructs and run-time support for *enforcing* invariants. The combination of the two—monitoring and enforcement—would provide developers with a full-fledged tool-set for building sophisticated, fully decentralized, and yet reliable cyber-physical applications.

Part III

Conclusions

Chapter 7

Conclusions and Outlook

Mobile WSNs are emerging as a solution addressing specific application demands such as the study of the social behavior of humans and wildlife, shepherding, tracking of assets, and environmental monitoring. Additionally, mote mobility can be employed as an elegant means serving non-functional requirements, e.g., to increase the lifetime of fixed WSNs. Nevertheless, mobility raises peculiar challenges, as interactions are transient, disconnections frequent and lifetime constraints more stringent w.r.t. the case of fixed nodes.

The work in this thesis is application-driven; in our three motivating scenarios — monitoring of the social behavior of wildlife, assisted living, support of business processes — we identified three fundamental problems that received little attention in the state-of-art. These are *1)* neighbor discovery, or the problem of identifying the nodes in the radio range, *2)* group membership, or the problem of identifying the transitively connected nodes, and *3)* the problem of monitoring distributed invariants in mobile networks, as well as in networks where nodes are fixed. Each problem entails a solution that lies at a different software layer, ranging from the MAC to the application layer.

RUTh, our *neighbor discovery protocol* executes at the MAC layer. We used an analytical model of the radio transceiver to formulate an optimization problem from two perspectives. First, we dealt with the detection latency: our goal was to guarantee a minimal energy consumption and that all neighbors are detected within a given interval. Dually, we tackled the problem from the perspective of lifetime, where we provide guarantees on the node lifetime while minimizing the time in which neighbors are detected. The defining feature of our protocol is its probabilistic mode; in this mode, RUTh trades-off the number of detected contacts for a decreased latency, respectively an increased lifetime.

We designed, implemented and compared three independent solutions to the problem of *group membership* using mainstream routing algorithms in computer science. These solutions cover the design space. At one extreme lies CLOCKS, a protocol in which nodes proactively exchange vector clocks to determine the group composition. At the other extreme lies LINKS, in which we use link state information to provide nodes with a consistent view on the group membership; in this solution, data is exchanged as a reaction to changes in the one-hop neighborhood. Finally, we presented DIST, a hybrid solution in which nodes reactively exchange distance vectors to infer the group composition and proactively exchange state summaries to recover from possible faults.

At the application layer, we provided DICE, a full-fledged solution to the problem of monitoring *global invariants*. The first constituent of this solution is a simple language in which domain experts specify invariants describing the correct behavior of the monitored processes. The invariants are compiled and dynamically loaded in a network that may be either fixed or mobile. To address fixed networks, we developed a solution that exploits in-network aggregation to achieve a minimal overhead and guarantees that at least one node detects invariant violations. We addressed slowly-mobile networks through a protocol that provides all nodes with an eventually consistent view on the violation state of the monitored invariants.

Interestingly, dynamic networks require the integration of all contributions in this thesis. In particular, node mobility misleads DICE into believing that relocated nodes are departed. From this perspective, group membership information can be integrated as a global failure detector in the DICE run-time. In this respect, we identified that the group monitoring protocol based on vector clocks is the most promising solution as it entails a fixed amount of traffic irrespective of node mobility. Furthermore, the integration can in principle reach lower layers. Specifically, we envision a solution in which a mobile node undergoes two general states, as follows. In the first state, the node is isolated and the operation of the higher level solutions (e.g., group monitoring, DICE) is suspended; the node employs RUTh to achieve a maximal energy-efficiency. In the second state, the node is in contact and thus the operation of the higher level solutions is resumed.

In hindsight, in addition to the extensive evaluation of our contributions in this thesis, we are looking forward to study the behavior of RUTh, group membership and DICE in real-world contexts. In particular, we have already developed the full RUTh technical machinery necessary for domain experts to properly configure the behavior of nodes in their applications. We showed that our protocol conforms to expectations in our laboratory and in controlled deployments. Nonetheless, preliminary experimentation also showed that our protocol is limited by the vagaries of wireless communication. If domain experts are to employ RUTh in their applications, they must be provided with the principled tools (e.g., models and empirical results) required to understand the scenarios that hamper our protocol.

A similar case occurs for our group monitoring algorithms. We determined that among the three, the link state solution performs the worst. Unfortunately, the choice between the other two solutions is not so obvious. Further investigation is required to break the tie between the vector clock and distance vector solutions. To this end, we require more accurate models of the group dynamics (e.g., for groups of people and animals) w.r.t. the solutions we resorted to in the absence of a ground truth. Another perspective from which these protocols can be analyzed and we have not yet considered is the underlying duty-cycle scheme. Intuition tells us that LPL hinders the reactive solutions, as these require immediate dissemination of state once neighborhoods change. This is a topic that we did not include in our evaluation and therefore additional measurements are needed for a deeper understanding of group membership problem in the context of WSNs.

A push for an integrated architecture containing DICE, group monitoring and RUTh is an important contribution per se. In this thesis, we approached the problem only from the perspective of mobility and provided a model that helps understanding which of DICE and an integrated DICE + CLOCKS is more suitable to address mobility. Despite this, a number of questions remained open. For instance, we can imagine a study of the interplay between detection latency and energy efficiency along the lines of RUTh, which provides an optimal configuration of the parameters in the framework. Moreover, the invariant specification language can be enriched with invariants involving aggregates over groups of nodes or with constructs that close the control loop, i.e., by enforcing invariants.

Part IV

Addendum

Bibliography

[1] The MultiHopLQI protocol. `http://www.tinyos.net/tinyos-2.x/tos/lib/net/lqi`. Accessed on 2/2010.

[2] IEEE Standard 802.15.4-2006: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), September 2006.

[3] Allred, J., Hasan, A. B., Panichsakul, S., Pisano, W., Gray, P., Huang, J., Han, R., Lawrence, D., and Mohseni, K. SensorFlock: an airborne wireless sensor network of micro-air vehicles. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2007), SenSys, ACM, pp. 117–129.

[4] Alsalih, W., Hassanein, H., and Akl, S. Placement of multiple mobile data collectors in wireless sensor networks. *Ad Hoc Netw. 8* (June 2010), 378–390.

[5] Ammari, H. M., and Das, S. K. Mission-oriented k-coverage in mobile wireless sensor networks. In *Proc. of the Int. Conf. on Distributed Computing and Networking* (2010), ICDCN, Springer-Verlag, pp. 92–103.

[6] Anastasi, G., Borgia, E., Conti, M., and Gregori, E. A Hybrid Adaptive Protocol for Reliable Data Delivery in WSNs with Multiple Mobile Sinks. *The Computer Journal 54*, 2 (2011), 213–229.

[7] Anastasi, G., Conti, M., Gregori, E., Spagoni, C., and Valente, G. Motes Sensor Networks in Dynamic Scenarios: an Experimental Study for Pervasive Applications in Urban Environments. *Int. Journal of Ubiquitous Computing and Intelligence 1* (2006).

[8] Birman, K. P. ISIS: A System for Fault-Tolerant Distributed Computing. Tech. rep., Cornell University, Ithaca, NY, USA, 1986.

[9] Brookmeyer, R., Johnson, E., Ziegler-Graham, K., and Arrighi, H. M. Forecasting the global burden of Alzheimer's disease. *Alzheimer's & Dementia : The Journal of the Alzheimer's Association 3*, 3 (July 2007), 186–191.

[10] Burrell, J., Brooke, T., and Beckwith, R. Vineyard Computing: Sensor Networks in Agricultural Production. *IEEE Pervasive Computing 3* (January 2004), 38–45.

[11] Butler, Z. J., Corke, P. I., Peterson, R. A., and Rus, D. Virtual Fences for Controlling Cows. In *Proc. of the Int. Conf. on Robotics and Automation* (2004), ICRA, IEEE, pp. 4429–4436.

[12] Cagnacci, F., Boitani, L., Powell, R. A., and Boyce, M. S. Animal ecology meets GPS-based radiotelemetry: a perfect storm of opportunities and challenges. *Philosophical Trans. of The Royal Society Biological Sciences 365*, 1550 (2010), 2157–2162.

[13] Cao, Q., Abdelzaher, T., Stankovic, J., Whitehouse, K., and Luo, L. Declarative tracepoints: a programmable and application independent debugging system for wireless sensor networks. In *Proc. of the Int. Conf. on Embedded Network Sensor Systems* (2008), SenSys, ACM, pp. 85–98.

[14] Cattani, M. Group Monitoring in Wireless Sensor Networks. Master's thesis, Università degli Studi di Trento. Faculty of Mathematical, Phisical and Natural Sciences, 2010.

[15] Cattani, M., Guna, S., and Picco, G. P. Group monitoring in mobile wireless sensor networks. In *Proc. of the Int. Conf. on Distributed Computing in Sensor Systems* (2011), DCOSS, IEEE, pp. 1–8.

[16] Ceriotti, M., Corra, M., D'Orazio, L., Doriguzzi, R., Facchin, D., Guna, S. T., Jesi, G. P., Cigno, R. L., Mottola, L., Murphy, A. L., Pescalli, M., Picco, G. P., Pregnolato, D., and Torghele, C. Is there light at the ends of the tunnel? Wireless sensor networks for adaptive lighting in road tunnels. In *Proc. of the Int. Conf. on Information Processing in Sensor Networks* (2011), IPSN, IEEE, pp. 187–198.

[17] Ceriotti, M., Mottola, L., Picco, G. P., Murphy, A. L., Guna, S., Corra, M., Pozzi, M., Zonta, D., and Zanon, P. Monitoring heritage buildings with wireless sensor networks: The torre aquila deployment. In *Proc. of the Int. Conf. on Information Processing in Sensor Networks* (2009), IPSN, IEEE, pp. 277–288.

[18] Chakrabarti, A., Sabharwal, A., and Aazhang, B. Using predictable observer mobility for power efficient design of sensor networks. In *Proc. of the Int. Conf. on Information Processing in Sensor Networks* (2003), IPSN, Springer-Verlag, pp. 129–145.

[19] Chan, H., Perrig, A., and Song, D. Random key predistribution schemes for sensor networks. In *Proc. of the Symp. on Security and Privacy* (2003), SP, IEEE, pp. 197–213.

[20] Chandy, K. M., and Lamport, L. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst. 3*, 1 (1985), 63–75.

[21] Chini, G. Wireless Sensor Networks for Assisted Living. An advanced monitoring system for fall detection using WSN. Master's thesis, Università degli Studi di Trento. Faculty of Mathematical, Phisical and Natural Sciences, 2010.

[22] Chipara, O., Brooks, C., Bhattacharya, S., Lu, C., Chamberlain, R. D., Roman, G.-C., and Bailey, T. C. Reliable real-time clinical monitoring using sensor network technology. In *Proc. of the AMIA Annual Symp.* (2009), AMIA, American Medical Informatics Association, pp. 103–107.

[23] Chipara, O., Lu, C., Bailey, T. C., and Roman, G.-C. Reliable clinical monitoring using wireless sensor networks: experiences in a step-down hospital unit. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2010), SenSys, ACM, pp. 155–168.

[24] Chipcon Tech. CC2420 Datasheet. `focus.ti.com/docs/prod/folders/print/cc2420.html`. Accessed on 9/2010.

[25] Considine, J., Li, F., Kollios, G., and Byers, J. Approximate Aggregation Techniques for Sensor Databases. In *Proc. of the Int. Conf. on Data Engineering* (2004), ICDE, IEEE, pp. 449–460.

[26] Costa, P., Mottola, L., Murphy, A. L., and Picco, G. P. Programming wireless sensor networks with the TeenyLime middleware. In *Proc. of the Int. Conf. on Middleware* (2007), Middleware, Springer-Verlag, pp. 429–449.

[27] Ş. Gună, Mottola, L., and Picco, G. DICE: Monitoring Global Invariants of Physical Processes using Wireless Sensor Networks. *(under review)* (2011).

[28] Di Francesco, M., Das, S. K., and Anastasi, G. Data Collection in Wireless Sensor Networks with Mobile Elements: A Survey. *ACM Trans. Sen. Netw. 8* (2011), 7:1–7:31.

[29] Doriguzzi Corin, R., Russello, G., and Salvadori, E. TinyKey: A light-weight architecture for Wireless Sensor Networks securing real-world applications. In *Proc. of the Int. Comf. on Wireless On-Demand Network Systems and Services* (2011), WONS, IEEE, pp. 68–75.

[30] Dunkels, A., Gronvall, B., and Voigt, T. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proc. of the Int. Conf. on Local Computer Networks* (2004), LCN, IEEE, pp. 455–462.

[31] Dunkels, A., Österlind, F., and He, Z. An Adaptive Communication Architecture for Wireless Sensor Networks. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2007), SenSys, ACM, pp. 335–349.

[32] Dutta, P., and Culler, D. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proc. of the Int. Conf. on Embedded Network Sensor Systems* (2008), SenSys, ACM, pp. 71–84.

[33] Dyo, V., Ellwood, S. A., Macdonald, D. W., Markham, A., Mascolo, C., Pásztor, B., Scellato, S., Trigoni, N., Wohlers, R., and Yousef, K. Evolution and sustainability of a wildlife monitoring sensor network. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2010), SenSys, ACM, pp. 127–140.

[34] Dyo, V., and Mascolo, C. Efficient node discovery in mobile wireless sensor networks. In *Proc. of the Int. Conf. on Distributed Computing in Sensor Systems* (2008), DCOSS, Springer-Verlag, pp. 478–485.

[35] Eisenman, S. B., Miluzzo, E., Lane, N. D., Peterson, R. A., Ahn, G.-S., and Campbell, A. T. The BikeNet mobile sensing system for cyclist experience mapping. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2007), SenSys, ACM, pp. 87–101.

[36] Faulkner, M., Olson, M., Chandy, R., Krause, J., Chandy, K. M., and Krause, A. The next big one: Detecting earthquakes and other rare events from community-based sensors. In *Proc. of the Int. Conf. on Information Processing in Sensor Networks* (2011), IPSN, IEEE, pp. 13–24.

[37] Frank, C., and Römer, K. Algorithms for generic role assignment in wireless sensor networks. In *Proc. of the Int. Conf. on Embedded networked sensor systems* (2005), SenSys, ACM, pp. 230–242.

[38] Gage, D. W. Command control for many-robot systems. *Control 10*, June (1992), 28–34.

[39] Galassi, M., et al. *GNU Scientific Library Reference Manual*, 3rd ed. Network Theory Ltd., February 2003.

[40] Gandham, S. R., Dawande, M., Prakash, R., and Venkatesan, S. Energy efficient schemes for wireless sensor networks with multiple mobile base stations. In *Global Telecommunications Conf* (2003), GLOBECOM, IEEE, pp. 377–381.

[41] Garg, V. K., and Waldecker, B. Detection of weak unstable predicates in distributed programs. *IEEE Trans. Parallel Distrib. Syst. 5* (March 1994), 299–307.

[42] Ghasemzadeh, H., Loseu, V., and Jafari, R. Collaborative signal processing for action recognition in body sensor networks: a distributed classification algorithm using motion transcripts. In *Proc. of the Int. Conf. on Information Processing in Sensor Networks* (2010), IPSN, ACM, pp. 244–255.

[43] Ghosh, A., and Das, S. Coverage and connectivity issues in wireless sensor networks: A survey. *Pervasive and Mobile Computing 4*, 3 (June 2008), 303–334.

[44] Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., and Levis, P. Collection tree protocol. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2009), SenSys, ACM, pp. 1–14.

[45] Grossglauser, M., and Tse, D. N. C. Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Trans. Netw. 10* (August 2002), 477–486.

[46] Guha, S., Plarre, K., Lissner, D., Mitra, S., Krishna, B., Dutta, P., and Kumar, S. AutoWitness: locating and tracking stolen property while tolerating GPS and radio outages. In *Proc. of the Int Conf. on Embedded Networked Sensor Systems* (2010), SenSys, ACM, pp. 29–42.

[47] Gupta, P., and Kumar, P. R. The capacity of wireless networks. *IEEE Trans. on Information Theory 46*, 2 (Mar. 2000), 388–404.

[48] Hedrick, C. Routing Information Protocol (RIP). RFC 1058, 1988.

[49] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., and Pister, K. System architecture directions for networked sensors. *SIGPLAN Not. 35* (November 2000), 93–104.

[50] Hoppough, S. Shelf life. *Forbes* (April 24 2006).

[51] Hu, Y.-C., Perrig, A., and Johnson, D. B. Ariadne: a secure on-demand routing protocol for ad hoc networks. *Wirel. Netw. 11* (January 2005), 21–38.

[52] Huang, J.-H., Amjad, S., and Mishra, S. CenWits: a sensor-based loosely coupled search and rescue system using witnesses. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2005), SenSys, ACM, pp. 180–191.

[53] Hull, B., Bychkovsky, V., Zhang, Y., Chen, K., Goraczko, M., Miu, A., Shih, E., Balakrishnan, H., and Madden, S. CarTel: a distributed mobile sensor computing system. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2006), SenSys, ACM, pp. 125–138.

[54] Jain, S., Shah, R. C., Brunette, W., Borriello, G., and Roy, S. Exploiting mobility for energy efficient data collection in wireless sensor networks. *Mob. Netw. Appl. 11* (June 2006), 327–339.

[55] Jhumka, A., and Mottola, L. On Consistent Neighborhood Views in Wireless Sensor Networks. In *Proc. of the Int. Symp. on Reliable Distributed Systems* (2009), SRDS, IEEE, pp. 199–208.

[56] Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L. S., and Rubenstein, D. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. *SIGARCH Comput. Archit. News 30*, 5 (2002), 96–107.

[57] Kandhalu, A., Lakshmanan, K., and Ragunathan, R. U-Connect: A low-latency energy-efficient asynchronous neighbor discovery protocol. In *Proc. of the Int. Conf. on Information Processing in Sensor Networks* (2010), IPSN, ACM, pp. 350–361.

[58] Kansal, A., Somasundara, A. A., Jea, D. D., Srivastava, M. B., and Estrin, D. Intelligent fluid infrastructure for embedded networks. In *Proc. of the Int. Conf. on Mobile systems, applications, and services* (2004), MobiSys, ACM, pp. 111–124.

[59] Karlof, C., Sastry, N., and Wagner, D. TinySec: a link layer security architecture for wireless sensor networks. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2004), SenSys, ACM, pp. 162–175.

[60] Kim, D. H., Kim, Y., Estrin, D., and Srivastava, M. B. SensLoc: sensing everyday places and paths using less energy. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2010), SenSys, ACM, pp. 43–56.

[61] Kim, H. S., Abdelzaher, T. F., and Kwon, W. H. Minimum-energy asynchronous dissemination to mobile sinks in wireless sensor networks. In *Proc. of the 1st Int. Conf. on Embedded Networked Sensor Systems* (2003), SenSys, ACM, pp. 193–204.

[62] Klingbeil, L., and Wark, T. A Wireless Sensor Network for Real-Time Indoor Localisation and Motion Monitoring. In *Proc. of the Int. Conf. on Information processing in sensor networks* (2008), IPSN, IEEE, pp. 39–50.

[63] Klugman, J., et al. *Human Development Report 2009. Overcoming barriers: Human mobility and development.* Palgrave Macmillan, 2009.

[64] Kothari, N., Gummadi, R., Millstein, T., and Govindan, R. Reliable and efficient programming abstractions for wireless sensor networks. In *Proc. of the Conf. on Programming language design and implementation* (2007), PLDI, ACM, pp. 200–210.

[65] Lachenmann, A., Marrón, P. J., Minder, D., and Rothermel, K. Meeting lifetime goals with energy levels. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2007), SenSys, ACM, pp. 131–144.

[66] Lai, T.-t. T., Chen, Y.-h. T., Huang, P., and Chu, H.-h. PipeProbe: a mobile sensor droplet for mapping hidden pipeline. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2010), SenSys, ACM, pp. 113–126.

[67] Laibowitz, M., Gips, J., Aylward, R., Pentland, A., and Paradiso, J. A. A sensor network for social dynamics. In *Proc. of the Int. Conf. on Information processing in sensor networks* (2006), IPSN, ACM, pp. 483–491.

[68] Levis, P., Patel, N., Culler, D., and Shenker, S. Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proc. of the Symp. on Networked Systems Design and Implementation* (2004), NSDI, USENIX, pp. 15–28.

[69] Lin, K., and Levis, P. Data Discovery and Dissemination with DIP. In *Proc. of the Int. Conf. on Information processing in sensor networks* (2008), IPSN, IEEE.

[70] Lipták, B. *Process Control.* Butterworth-Heinemann, 1995.

[71] Liu, T., and Martonosi, M. Impala: a middleware system for managing autonomic, parallel sensor systems. In *Proc. of the Symp. on Principles and practice of parallel programming* (2003), PPoPP, ACM, pp. 107–118.

[72] Lorincz, K., Chen, B.-r., Challen, G. W., Chowdhury, A. R., Patel, S., Bonato, P., and Welsh, M. Mercury: a wearable sensor network platform for high-fidelity motion analysis. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2009), SenSys, ACM, pp. 183–196.

[73] Lorincz, K., Malan, D. J., Fulford-Jones, T. R. F., Nawoj, A., Clavel, A., Shnayder, V., Mainland, G., Welsh, M., and Moulton, S. Sensor Networks for Emergency Response: Challenges and Opportunities. *IEEE Pervasive Computing 3*, 4 (Oct. 2004), 16–23.

[74] Lu, H., Yang, J., Liu, Z., Lane, N. D., Choudhury, T., and Campbell, A. T. The Jigsaw continuous sensing engine for mobile phone applications. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2010), SenSys, ACM, pp. 71–84.

[75] Luo, J., and Hubaux, J. Joint mobility and routing for lifetime elongation in wireless sensor networks. In *Proceedings of the Conf. on Computer Communications* (2005), Infocom, IEEE, pp. 1735–1746.

[76] Luo, J., Panchard, J., Piórkowski, M., Grossglauser, M., and pierre Hubaux, J. Mobiroute: Routing towards a mobile sink for improving lifetime in sensor networks. In *Proc. of the Int. Conf. on Distributed Computing in Sensor Systems* (2006), DCOSS, Springer Verlag, pp. 480–497.

[77] Lusseau, D., and Newman, M. E. J. Identifying the role that individual animals play in their social network. *Proc. R. Soc. London B (Suppl.) 271* (2004).

[78] Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. TAG: a Tiny AGgregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev. 36* (December 2002), 131–146.

[79] Madden, S. R., Franklin, M. J., Hellerstein, J. M., and Hong, W. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst. 30* (March 2005), 122–173.

[80] Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., and Anderson, J. Wireless sensor networks for habitat monitoring. In *Proc. of the Int. workshop on Wireless sensor networks and applications* (2002), WSNA, ACM, pp. 88–97.

[81] Malinowski, M., Moskwa, M., Feldmeier, M., Laibowitz, M., and Paradiso, J. A. CargoNet: a low-cost micropower sensor node exploiting quasi-passive wakeup for adaptive asychronous monitoring of exceptional events. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2007), SenSys, ACM, pp. 145–159.

[82] Maróti, M., Kusy, B., Simon, G., and Lédeczi, A. The flooding time synchronization protocol. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2004), SenSys, ACM, pp. 39–49.

[83] Martinez, K., Hart, J. K., and Ong, R. Environmental sensor networks. *Computer 37* (August 2004), 50–56.

[84] Mattern, F. Virtual time and global states of distributed systems. In *Proc. of the Workshop on Parallel and Distributed Algorithms* (1989), Elsevier, pp. 215–226.

[85] McGlynn, M. J., and Borbash, S. A. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *Proc. of the Int. Symp. on Mobile ad hoc networking & computing* (2001), MobiHoc, ACM, pp. 137–145.

[86] Mills, D. Network Time Protocol (NTP). RFC 958, 1985.

[87] Miluzzo, E., Lane, N. D., Fodor, K., Peterson, R., Lu, H., Musolesi, M., Eisenman, S. B., Zheng, X., and Campbell, A. T. Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application. In *Proc. of the Int. Conf. on Embedded Network Sensor Systems* (2008), SenSys, ACM.

[88] Mohan, P., Padmanabhan, V. N., and Ramjee, R. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proc. of the Int. Conf. on Embedded Network Sensor Systems* (2008), SenSys, ACM, pp. 323–336.

[89] Molteni, D. Reti di sensori wireless per lo studio dei comportamenti sociali di animali selvatici (Wireless Sensor Networks for Studying Wildlife Social Behavior). Master's thesis, Università degli Studi di Trento. Engineering Faculty, 2010.

[90] Moss, D., and Levis, P. BoX-MACs: Exploiting Physical and Link Layer Boundaries in Low-Power Networking. Tech. Rep. SING-08-00, Stanford Information Networks Group, 2008.

[91] Mottola, L., and Picco, G. P. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Comput. Surv. 43* (April 2011), 19:1–19:51.

[92] Moy, J. Open Shortest Path First (OSPF). RFC 2328, 1998.

[93] Nath, S., Gibbons, P. B., Seshan, S., and Anderson, Z. Synopsis diffusion for robust aggregation in sensor networks. *ACM Trans. Sen. Netw. 4* (April 2008), 7:1–7:40.

[94] Österlind, F., Dunkels, A., Eriksson, J., Finne, N., and Voigt, T. Cross-level sensor network simulation with COOJA. In *Proc. of the Int. Conf. on Local Computer Networks* (2006), LCN, IEEE, pp. 641–648.

[95] Peterson, M. N., Lopez, R. R., Frank, P. A., Peterson, M. J., and Silvy, N. J. Evaluating capture methods for urban white-tailed deer. *Wildlife Society Bulletin 31*, 4 (2003).

[96] Polastre, J., Hill, J., and Culler, D. Versatile low power media access for wireless sensor networks. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2004), SenSys, ACM, pp. 95–107.

[97] Polastre, J., Szewczyk, R., and Culler, D. Telos: Enabling ultra-low power wireless research. In *Proc. of the Int. Symp. on Information Processing in Sensor Networks* (2005), IPSN, IEEE.

[98] Purohit, A., Sun, Z., Mokaya, F., and Zhang, P. SensorFly: Controlled-mobile sensing platform for indoor emergency response applications. In *Proc. of the Int. Conf. on Information Processing in Sensor Networks* (2011), IPSN, IEEE, pp. 223–234.

[99] Rana, R. K., Chou, C. T., Kanhere, S. S., Bulusu, N., and Hu, W. Ear-phone: an end-to-end participatory urban noise mapping system. In *Proc. of the Int. Conf. on Information Processing in Sensor Networks* (2010), IPSN '10, ACM, pp. 105–116.

[100] Razvan Musaloiu, E., Liang, C.-J. M., and Terzis, A. Koala: Ultra-Low Power Data Retrieval in Wireless Sensor. In *Proc. of the Int. Conf. on Information Processing in Sensor Networks* (2008), IPSN, IEEE, pp. 421–432.

[101] Rhee, I., Shin, M., Hong, S., Lee, K., Kim, S., and Chong, S. CRAWDAD trace ncsu/mobilitymodels/gps/kaist (v. 2009-07-23). From http://crawdad.cs.dartmouth.edu/ncsu/mobilitymodels/GPS/KAIST.

[102] Roman, G.-C., Huang, Q., and Hazemi, A. Consistent group membership in ad hoc networks. In *Proc. of the Int. Conf. on Software Engineering* (2001), ICSE, IEEE, pp. 381–388.

[103] Römer, K. Time synchronization in ad hoc networks. In *Proc. of the Int. Symp. on Mobile ad hoc networking & computing* (2001), MobiHoc, ACM, pp. 173–182.

[104] Römer, K., and Ma, J. PDA: Passive distributed assertions for sensor networks. In *Proc. of the Int. Conf. on Information Processing in Sensor Networks* (2009), IPSN, IEEE, pp. 337–348.

[105] Royer, E. M., and Toh, C.-K. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications 6*, 2 (Apr. 1999), 46–55.

[106] Schiele, B., Schmidt, A., Michahelles, F., and Matter, P. Applying wearable sensors to avalanche rescue: First experiences with a novel avalanche beacon. *Computers & Graphics 27*, 6 (2003), 839–847.

[107] Sentilla. TMote Sky Datasheet. `http://www.sentilla.com/moteiv-transition.html`. Accessed on 1/2011.

[108] Sookoor, T., Hnat, T., Hooimeijer, P., Weimer, W., and Whitehouse, K. Macrodebugging: global views of distributed program execution. In *Proc. of the 7th Conf. on Embedded Networked Sensor Systems (SenSys)* (2009).

[109] Sorber, J. *System support for perpetual mobile tracking.* PhD thesis, University of Massachusetts - Amherst, 2010.

[110] Sridhar, N. Decentralized Local Failure Detection in Dynamic Distributed Systems. In *Proc. of the Symp. on Reliable Distributed Systems* (2006), SRDS, IEEE, pp. 143–154.

[111] Stankovic, J. A., Lee, I., Mok, A., and Rajkumar, R. Opportunities and obligations for physical computing systems. *Computer 38* (November 2005), 23–31.

[112] Sugihara, R., and Gupta, R. K. Improving the Data Delivery Latency in Sensor Networks with Controlled Mobility. In *Proc. of the Int. Conf. on Distributed Computing in Sensor Systems* (2008), DCOSS, Springer-Verlag, pp. 386–399.

[113] Thiagarajan, A., Biagioni, J., Gerlich, T., and Eriksson, J. Cooperative transit tracking using smart-phones. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2010), SenSys, ACM, pp. 85–98.

[114] Tolle, G., Polastre, J., Szewczyk, R., Culler, D., Turner, N., Tu, K., Burgess, S., Dawson, T., Buonadonna, P., Gay, D., and Hong, W. A macroscope in the redwoods. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2005), SenSys, ACM, pp. 51–63.

[115] Tomlinson, A. I., and Garg, V. K. Monitoring functions on global states of distributed programs. *J. Parallel Distrib. Comput. 41* (March 1997), 173–189.

[116] Tseng, Y.-C., Hsu, C.-S., and Hsieh, T.-Y. Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks. *Comput. Netw. 43*, 3 (2003), 317–337.

[117] Čapkun, S., Hubaux, J.-P., and Buttyán, L. Mobility helps security in ad hoc networks. In *Proc. of the Int. Symp. on Mobile ad hoc networking & computing* (2003), MobiHoc, ACM, pp. 46–56.

[118] Vitenberg, R., Keidar, I., Chockler, G. V., and Dolev, D. Group communication specifications: A comprehensive study. *ACM Computing Surveys 33*, 4 (1999).

[119] Wan, C.-Y., Eisenman, S. B., Campbell, A. T., and Crowcroft, J. Siphon: overload traffic management using multi-radio virtual sinks in sensor networks. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2005), SenSys, ACM, pp. 116–129.

[120] Wang, Z. M., Basagni, S., Melachrinoudis, E., and Petrioli, C. Exploiting Sink Mobility for Maximizing Sensor Networks Lifetime. In *Proc. of the Int. Conf. on System Sciences* (2005), IEEE, p. 287.

[121] Wark, T., Crossman, C., Hu, W., Guo, Y., Valencia, P., Sikka, P., Corke, P., Lee, C., Henshall, J., Prayaga, K., O'Grady, J., Reed, M., and Fisher, A. The design and evaluation of a mobile sensor/actuator network for autonomous animal control. In *Proc. of the Int. Conf. on Information processing in sensor networks* (2007), IPSN, ACM, pp. 206–215.

[122] Welsh, M., and Mainland, G. Programming sensor networks using abstract regions. In *Proc. of the Symp. on Networked Systems Design and Implementation* (2004), NSDI, USENIX, pp. 29 – 42.

[123] Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J., and Welsh, M. Fidelity and yield in a volcano monitoring sensor network. In *Proc. of the Symp. on Operating Systems Design and Implementation* (2006), OSDI, USENIX, pp. 381–396.

[124] Whitehouse, K., Sharp, C., Brewer, E., and Culler, D. Hood: a neighborhood abstraction for sensor networks. In *Proc. of the Int. Conf. on Mobile systems, applications, and services* (2004), MobiSys, ACM, pp. 99–110.

[125] Wood, A., Virone, G., Doan, T., Cao, Q., Selavo, L., Wu, Y., Fang, L., He, Z., Lin, S., and Stankovic, J. Alarm-net: Wireless sensor networks for assisted-living and residential monitoring. Tech. rep., 2006.

[126] Yap, K.-K., Srinivasan, V., and Motani, M. MAX: human-centric search of the physical world. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2005), SenSys, ACM, pp. 166–179.

[127] Ye, F., Luo, H., Cheng, J., Lu, S., and Zhang, L. A two-tier data dissemination model for large-scale wireless sensor networks. In *Proc. of the Int. Conf. on Mobile computing and networking* (2002), MobiCom, ACM, pp. 148–159.

[128] Young, A. D., Ling, M. J., and Arvind, D. K. Distributed estimation of linear acceleration for improved accuracy in wireless inertial motion capture. In *Proc. of the Int. Conf. on Information Processing in Sensor Networks* (2010), IPSN, ACM, pp. 256–267.

[129] Zhang, P., Sadler, C. M., Lyon, S. A., and Martonosi, M. Hardware design experiences in ZebraNet. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems* (2004), SenSys, ACM, pp. 227–238.

[130] Zhang, Z. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges. *IEEE Communications Surveys & Tutorials 8*, 1 (Mar. 2006), 24–37.

[131] Zheng, R., Hou, J. C., and Sha, L. Asynchronous wakeup for ad hoc networks. In *Proc. of the Int. Symp. on Mobile ad hoc networking & computing* (2003), MobiHoc, ACM, pp. 35–45.

[132] Zhu, Y., and Sivakumar, R. Challenges: communication through silence in wireless sensor networks. In *Proc. of the Int. Conf. on Mobile computing and networking* (2005), MobiCom, ACM, pp. 140–147.

Appendix A

The Discovery Probability in RUTh

In this appendix, we compute the probability $P\left[\bigvee_{i=0}^{n} \mathcal{C}(i)\right]$ that at least one contact is detected during a time period spanning n consecutive epochs. This appendix serves as an analytical support for Section 4.4.2. Recall that the main challenge lies in computing the probability of a x-discovery chain, as illustrated by Figure A.1.

In what follows, we make the simplifying assumption that the contact occurs at time $\gamma = 0$. We then determine the analytical expression of the discovery probability as follows:

1. show that the length of a discovery chain is finite (Lemma 1);
2. use the previous result to express the discovery probability at the end of epoch n as a recursive function (Theorem 1);
3. the coefficients in the above function are basic probabilities (e.g. $P\left[A_n \leftarrow B_n\right]$). We determine their associated probability density function (pdf) and then use the relation $P\left[X \in D\right] = \int_D f(x)\, \mathrm{d}x$, where f is the pdf of X, to switch to probabilities. To embed this analytical process in our solver, we rely on the numerical integration tools in [39].

First, we show that the length of a discovery chain is finite. We state the following

Lemma 1. *If* $\textsc{active} < {}^{T}\!/_{2} + \lambda$, *the maximum length of a discovery chain is* $2m + 3$ *where* $m \in \mathbb{N}$ *and*

$$m = \left\lfloor \frac{\textsc{active} - 2\lambda}{T + 2\lambda - 2\textsc{active}} \right\rfloor$$

Proof. If $\phi < \lambda$, the displacement of B's schedule w.r.t. A's own schedule prevents B's active interval to span two consecutive epochs of A. Thus, in this case, only 1-discovery chains are possible. Hereafter, we assume $\phi \geq \lambda$.

First, we define a set of helper intervals I_m that begin with the end of A's last receive check in a random epoch k, and end with the beginning of A's beacon in epoch $k+m+1$. Figure A.1 illustrates I_1 and I_2. As I_1 begins in epoch k and ends in epoch $k+2$, it is obvious that this interval must be larger than T, the epoch duration. Similarly, I_2 must be larger than $2T$. Generalizing, we obtain $I_m \geq mT$. This constraint captures the minimal

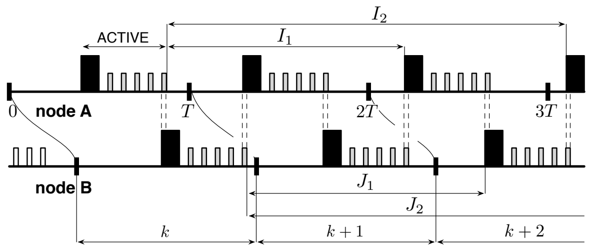


Figure A.1: Schedule for a 6-discovery chain. Also shown are the helper intervals I_m and J_m.

span of a chain: it ensures that the gap between activity in k and activity in $k+1+m$ is large enough to cover the intermediate m epochs.

To understand the connection between I_m and the chain length, we first observe that a 2-discovery chain (e.g., $A_k \leftarrow B_k \wedge B_k \leftarrow A_{k+1}$ in Figure A.1) is always possible. In fact, it expands in only two consecutive epochs and, therefore, it is not subject to the previous constraint. Moreover, each increment of m adds two discoveries to the chain: I_1 adds $A_{k+1} \leftarrow B_{k+1} \wedge B_{k+1} \leftarrow A_{k+2}$, and so on. Therefore, I_m determines, through the aforementioned constraint, the minimal span of a discovery chain of length $(2+2m)$.

We now determine instead the maximal span of I_m. Let us consider the conditions under which a 4-discovery chain can occur, associated to I_1 in Figure A.1. Depending on the schedule, various values are possible for I_1. However, it is easy to see (e.g., in Figure A.1) that its *maximum* value occurs when the active intervals overlap only for the minimum amount of time λ necessary to enable discovery, i.e., $\max(I_1) = 3\textsc{active} - 4\lambda$. Similarly, for a 6-discovery chain, we have $\max(I_2) = 5\textsc{active} - 6\lambda$. Therefore, we generalize to a discovery chain of length $2+2m$, for which

$$\max(I_m) = (2m+1)\,\textsc{active} - 2\,(m+1)\,\lambda$$

This result, combined with $I_m \geq mT$, yields the upper bound

$$m \leq \left\lfloor \frac{\textsc{active} - 2\lambda}{T + 2\lambda - 2\textsc{active}} \right\rfloor \tag{A.1}$$

where we use the floor operator because $m \in \mathbb{N}$.

Thus far, we considered only the intervals I_m, defined w.r.t. A's schedule. Similar considerations, however, hold for B's schedule. As shown in Figure A.1, we can define a set of intervals J_m, beginning with the end of B's last receive check in epoch k. Observe that B's behavior is symmetric to A's and therefore $\max(J_m) = \max(I_m)$. Nevertheless, the intervals J_m are shifted to the right w.r.t. the intervals I_m because of the phase ϕ. Therefore, much like a 2-discovery chain is always possible for A, a 3-discovery chain is always possible for B. Therefore, $(3+2m)$ is the maximal length of a discovery chain for B, where m is given by Equation A.1. □

This result allows us to compute the discovery probability during epoch n by considering only a limited number of past epochs. Before enunciating our main result, we introduce the following notation for a discovery chain. If $X \in \{A, B\}$, we define ${}^{i}\mathcal{D}_{X_n}$ as the i-discovery chain that ends with the discovery that occurs during the transmission of

X's n-th beacon. For instance, Figure A.1 depicts the chain ${}^{6}\mathcal{D}_{A_{k+2}}$. 1-discovery chains are equivalent to a single discovery event, e.g., $A_n \leftarrow B_n \equiv {}^{1}\mathcal{D}_{B_n}$.

Theorem 1. *If* ACTIVE $< T/2$, *then the discovery probability* $P\left[\bigvee_{i=0}^{n} \mathcal{C}(i)\right]$ *is a recursive function defined as:*

$$\begin{aligned} f(n) \triangleq\ & g(n) + P\left[B_n \leftarrow A_n\right] \cdot (1 - f(n-1)) \\ & + \textstyle\sum_{i=0}^{m} (1 - f(n-i-1)) \cdot P\left[{}^{2i+1}\mathcal{D}_{B_n}\right] \\ & + \textstyle\sum_{i=1}^{m} (g(n-i) - 1) \cdot P\left[{}^{2i}\mathcal{D}_{B_n}\right] \end{aligned}$$

where $g(n)$ *is a helper function defined as:*

$$\begin{aligned} g(n) \triangleq\ & f(n-1) + P\left[A_n \leftarrow B_{n-1}\right] \cdot (1 - g(n-1)) \\ & + \textstyle\sum_{i=1}^{m} P\left[{}^{2i}\mathcal{D}_{A_n}\right] \cdot (f(n-i-1) - 1) \\ & + \textstyle\sum_{i=0}^{m} P\left[{}^{2i+1}\mathcal{D}_{A_n}\right] \cdot (1 - g(n-i)) \end{aligned}$$

and the value of m *is obtained from Lemma 1.*

Proof. Despite their complex expressions, the functions above are derived through the simple repeated application of the fundamental relation $P[A \vee B] = P[A] + P[B] - P[A \wedge B]$. For instance, we start by expanding $P\left[\bigvee_{i=0}^{n} \mathcal{C}(i)\right]$ into

$$P[\mathcal{C}(n)] + P\left[\bigvee_{i=0}^{n-1} \mathcal{C}(i)\right] - P\left[\mathcal{C}(n) \wedge \bigvee_{i=0}^{n-1} \mathcal{C}(i)\right]$$

Here, the second term is $P\left[\bigvee_{i=0}^{n-1} \mathcal{C}(i)\right] = f(n-1)$, the first term in $g(n)$. Moreover, since we measure time by using A's schedule as reference, and $\phi > 0$, A's epoch n can only overlap with B's epochs $n-1$ or n. Therefore $\mathcal{C}(n)$, the event representing contact, can be expressed as the combination of the individual events in these epochs:

$$\mathcal{C}(n) \equiv A_n \leftarrow B_n \vee B_n \leftarrow A_n \vee A_n \leftarrow B_{n-1} \vee B_{n-1} \leftarrow A_n$$

Note how terms of $\mathcal{C}(n)$ appear in various positions in $f(n)$ and $g(n)$. The remaining terms of these functions, that is, chains longer than 1, derive from conjunctions such as $\mathcal{C}(n) \wedge \bigvee_{i=0}^{n-1} \mathcal{C}(i)$.

From now on, the recursive expansion of the union adds new terms to the conjunction, resulting in ever-increasing chains. Recursion ends when the maximum chain length dictated by Lemma 1 is reached. Next, we provide the details of these calculations.

Premises. It is obvious that if either one of $A_n \leftarrow B_n$ or $B_n \leftarrow A_n$ holds, then a contact occurs in slot n. In addition, as we count epochs as seen by A, a contact can possibly be triggered by B's activity in its epoch $n-1$. That is, if one of $A_n \leftarrow B_{n-1}$ or $B_{n-1} \leftarrow A_n$ holds, then a contact occurs in slot n. Notice that because the phase $\phi \geq 0$, then there is no overlap between A's n-th epoch and B's $n+1$ epoch.

Let $\mathcal{D}_n \triangleq A_n \leftarrow B_n \vee B_n \leftarrow A_n$. With this notation, we summarize the previous discussion as:

$$\mathcal{C}(n) \triangleq \mathcal{D}_n \vee A_n \leftarrow B_{n-1} \vee B_{n-1} \leftarrow A_n$$

Now, let $f(n) \triangleq P\left[\bigvee_{i=0}^{n} \mathcal{C}(i)\right]$. We expand $\mathcal{C}(n)$ in the union and then we apply the relation $P[X \vee Y] = P[X] + P[Y] - P[X \wedge Y]$ to obtain:

$$\begin{aligned} f(n) &= P\left[\mathcal{D}_n \vee A_n \leftarrow B_{n-1} \vee B_{n-1} \leftarrow A_n \vee \bigvee_{i=0}^{n-1} \mathcal{C}(i)\right] \\ &= \mathsf{p}_1 + P[A_n \leftarrow B_n] - \mathsf{p}_2 \end{aligned} \tag{A.2}$$

where $P[X] = \mathsf{p}_1$, $P[Y] = P[A_n \leftarrow B_n]$ and $P[X \wedge Y] = \mathsf{p}_2$:

$$\begin{aligned} \mathsf{p}_1 &\triangleq P\left[B_n \leftarrow A_n \vee A_n \leftarrow B_{n-1} \vee B_{n-1} \leftarrow A_n \vee \bigvee_{i=0}^{n-1} \mathcal{C}(i)\right] \\ \mathsf{p}_2 &\triangleq P\left[\left(B_{n-1} \leftarrow A_n \vee \bigvee_{i=0}^{n-1} \mathcal{C}(i)\right) \wedge A_n \leftarrow B_n\right] \end{aligned} \tag{A.3}$$

Notice that p_2 replaces the term $P[X \wedge Y]$ in the expansion of the union $P[X \vee Y]$. Also, there was a shortcut we took in the expression of p_2 above: we dropped the terms $B_n \leftarrow A_n$ and $A_n \leftarrow B_{n-1}$ that would normally appear in p_2, as these are incompatible with $A_n \leftarrow B_n$:

- $P[A_n \leftarrow B_n \wedge B_n \leftarrow A_n] = 0$, as only one of these events is possible at the same time.
- $P[A_n \leftarrow B_n \wedge A_n \leftarrow B_{n-1}] = 0$, as during an epoch a node cannot receive two beacons sent by the same node.

Proof roadmap. We reached a state where we have two expressions, p_1 and p_2. To prove Theorem 1, we must simplify them until we find a recursive connection between them. This connection will consists of the function $f(n)$ and the helper function $g(n)$.
Step 1 (finding p_1). We expand p_1 first. We start by expanding the logical disjunction in the expression of p_1 appearing immediately after $B_n \leftarrow A_n$. We obtain:

$$\begin{aligned} \mathsf{p}_1 &= g(n) + P[B_n \leftarrow A_n] - \\ &- P\left[B_n \leftarrow A_n \wedge \left(A_n \leftarrow B_{n-1} \vee B_{n-1} \leftarrow A_n \vee \bigvee_{i=0}^{n-1} \mathcal{C}(i)\right)\right] \end{aligned}$$

where

$$g(n) \triangleq P\left[A_n \leftarrow B_{n-1} \vee B_{n-1} \leftarrow A_n \vee \bigvee_{i=0}^{n-1} \mathcal{C}(i)\right] \tag{A.4}$$

Notice that in the last term of p_1:

- $P[B_n \leftarrow A_n \wedge A_n \leftarrow B_{n-1}] = 0$. This can be proved by contradiction. For instance, assume $A_n \leftarrow B_{n-1}$ stands, that is, in epoch n node A received B's $n-1$-th beacon. If $B_n \leftarrow A_n$ was also true, this would imply that B in epoch n would receive A's n-th beacon. In other words, B's listen in the n-th epoch occurs before A's n-th beacon, which also occurs before B's $n-1$-th epoch. Hence, the contradiction.
- $P[B_n \leftarrow A_n \wedge B_{n-1} \leftarrow A_n] = 0$ because A's n-th beacon can only be received once by node B.
- the indexes in $\bigvee_{i=0}^{n-1} \mathcal{C}(i)$ stop at $n-1$, thus the union $\bigvee_{i=0}^{n-1} \mathcal{C}(i)$ is independent from $B_n \leftarrow A_n$.

Armed with the previous observations, we simplify p_1 to:

$$\mathsf{p}_1 = g(n) + P[B_n \leftarrow A_n] - f(n-1) \cdot P[B_n \leftarrow A_n]$$

Step 2 (finding p_2). At this point, p_1 cannot be reduced further, although much of the difficulty has been passed on $g(n)$. First, we simplify p_2 defined in (A.3), the focus on $g(n)$, defined in (A.4).

We use the chain notation and we factorize the unions in (A.3) to obtain:

$$\begin{aligned} \mathsf{p}_2 &= P\left[\left(B_{n-1} \leftarrow A_n \vee \left(\bigvee_{i=0}^{n-1} \mathcal{C}(i)\right)\right) \wedge A_n \leftarrow B_n\right] \\ &= f(n-1) \cdot P[A_n \leftarrow B_n] + P\left[{}^2\mathcal{D}_{B_n}\right] - \mathsf{p}_3 \end{aligned} \tag{A.5}$$

where $\mathsf{p}_3 \triangleq P\left[\bigvee_{i=0}^{n-1} \mathcal{C}(i) \wedge^2 \mathcal{D}_{B_n}\right]$.

Notice that normally $\mathcal{C}(n-1)$ contains a term $B_{n-1} \leftarrow A_{n-1}$. This is however incompatible with the chain ${}^2\mathcal{D}_{B_n}$ (the same reasoning as per the simplification of p_1 can be applied). Thus we drop this term and further expand the disjunction to obtain:

$$\mathsf{p}_3 = g(n-1) \cdot P\left[{}^2\mathcal{D}_{B_n}\right] + P\left[{}^3\mathcal{D}_{B_n}\right] - \mathsf{p}_4$$

where:

$$\begin{aligned} \mathsf{p}_4 &\triangleq P\left[\left(B_{n-2} \leftarrow A_{n-1} \vee \bigvee_{i=0}^{n-2} \mathcal{C}(i)\right) \wedge^3 \mathcal{D}_{B_n}\right] \\ &= f(n-2) \cdot P\left[{}^3\mathcal{D}_{B_n}\right] + P\left[{}^4\mathcal{D}_{B_n}\right] - \mathsf{p}_5 \end{aligned}$$

Recall that $A_n \leftarrow B_n \equiv^1 \mathcal{D}_{B_n}$ to observe the similarities between the above identity and (A.5). Namely, there is a pattern involving the call to $f(n)$ and the chain probabilities. In p_4, we define p_5 as:

$$\mathsf{p}_5 \triangleq g(n-2) \cdot P\left[{}^4\mathcal{D}_{B_n}\right] + P\left[{}^5\mathcal{D}_{B_n}\right] - \mathsf{p}_6$$

where:

$$\mathsf{p}_6 \triangleq f(n-3) \cdot P\left[{}^5\mathcal{D}_{B_n}\right] + P\left[{}^6\mathcal{D}_{B_n}\right] - \mathsf{p}_7$$

Notice that the pattern keeps repeating. It will do so until we reach a chain that has probability 0 according to Lemma 1.

Step 3 (the helper function $g(n)$). We are currently in the state where we can infer a recursive formula for p_2. Next, we focus on the remaining objective, i.e., finding an expression for $g(n)$. We factorize (A.4) and transform it to obtain:

$$\begin{aligned} g(n) &= P\left[A_n \leftarrow B_{n-1} \vee \left(B_{n-1} \leftarrow A_n \vee \bigvee_{i=0}^{n-1} \mathcal{C}(i)\right)\right] \\ &= \mathsf{p}_9 + P[A_n \leftarrow B_{n-1}] - \mathsf{p}_8 \end{aligned}$$

where $\mathsf{p}_9 \triangleq P\left[B_{n-1} \leftarrow A_n \vee \bigvee_{i=0}^{n-1} \mathcal{C}(i)\right]$ and, because of the incompatibility between $A_n \leftarrow B_{n-1}$ and $B_{n-1} \leftarrow A_n$, $\mathsf{p}_8 \triangleq P\left[\bigvee_{i=0}^{n-1} \mathcal{C}(i) \wedge A_n \leftarrow B_{n-1}\right]$.

In p_8, we expand $\mathcal{C}(n-1)$ and drop the resulting incompatible terms. We obtain:

$$\mathsf{p}_8 = g(n-1) \cdot P[A_n \leftarrow B_{n-1}]$$

We have found a formula for p_8. We still have to work on p_9. It is immediate that:

$$\mathsf{p}_9 = f(n-1) + P[B_{n-1} \leftarrow A_n] - \mathsf{p}_{10}$$

where $\mathsf{p}_{10} \triangleq P\left[\bigvee_{i=0}^{n-1} \mathcal{C}(i) \wedge^1 \mathcal{D}_{A_n}\right]$. Next, we expand $\mathcal{C}(n-1)$ to obtain:

$$\mathsf{p}_{10} = g(n-1) \cdot P\left[{}^1\mathcal{D}_{A_n}\right] + P\left[{}^2\mathcal{D}_{A_n}\right] - \mathsf{p}_{11}$$

We continue recursively:

$$\begin{aligned}
\mathsf{p}_{11} &\triangleq f(n-2) \cdot P\left[{}^2\mathcal{D}_{A_n}\right] + P\left[{}^3\mathcal{D}_{A_n}\right] - \mathsf{p}_{12} \\
\mathsf{p}_{12} &\triangleq g(n-2) \cdot P\left[{}^3\mathcal{D}_{A_n}\right] + P\left[{}^4\mathcal{D}_{A_n}\right] - \mathsf{p}_{13} \\
\mathsf{p}_{13} &\triangleq f(n-3) \cdot P\left[{}^4\mathcal{D}_{A_n}\right] + P\left[{}^5\mathcal{D}_{A_n}\right] - \mathsf{p}_{14} \\
&\dots
\end{aligned}$$

In conclusion. The recursion continues up to the moment when the chains in the equation become long enough for Lemma 1 to dictate that their probability is 0. At that point, by replacing all previous expressions in (A.2), we conclude the proof.

□

To compute $f(n)$, we must compute the probability of basic discovery events, e.g., $P[A_n \leftarrow B_n]$. However:

- From probability theory, if X and Y are two random variables with pdf f_X and f_Y, respectively, the joint pdf $X+Y$ is the convolution of the individual ones:

$$f_{X+Y}(t) = (f_X * f_Y)(t) \triangleq \int_{\mathbb{R}} f_X(\xi) \cdot f_Y(t-\xi)\, \mathrm{d}\xi$$

- The pdf of the difference $X-Y$ can be expressed, using the operator $\circ$ to simplify expressions, as:

$$f_{X-Y}(t) = (f_X \circ f_Y)(t) \triangleq \int_{\mathbb{R}} f_X(\xi) \cdot f_Y(t+\xi)\, \mathrm{d}\xi$$

Armed with this knowledge, we state the following

Lemma 2. *If α_n and β_n are random variables with uniform distribution in the interval $[0, T - \text{ACTIVE})$, and the phase difference has uniform distribution in $[0,T)$, then for $n \geq 1$ the pdf-s of $P[A_n \leftarrow B_n]$ and $P[B_n \leftarrow A_n]$ are:*

$$f_{A_n \leftarrow B_n}(z) = f_{B_n \leftarrow A_n}(z) = (f_\phi * f_{\beta_n} \circ f_{\alpha_n})(z)$$

Proof. Discovery $A_n \leftarrow B_n$ occurs when the beginning of B's active interval falls in A's active interval, after the beacon and before the last possible minimum overlap λ. Formally:

$$\begin{cases} \phi + \beta_n \geq \alpha_n + \tau \\ \phi + \beta_n \leq \alpha_n + \text{ACTIVE} - \lambda \end{cases} \tag{A.6}$$

Let us define $Z \triangleq \phi + \beta_n - \alpha_n$ and f_Z as the associated pdf. Based on the previous considerations, we can derive:

$$f_Z(z) = (f_\phi * f_{\beta_n} \circ f_{\alpha_n})(z)$$

From the definition of Z and Equation (A.6), the event $A_n \leftarrow B_n$ is equivalent to the event $Z \in [\tau; \text{ACTIVE} - \lambda]$. Therefore, they share the same pdf: $f_{A_n \leftarrow B_n} = f_Z$. A similar reasoning holds for the event $B_n \leftarrow A_n$. By imposing constraints similar to Equation (A.6), we obtain that this event is equivalent to $Z \in [\lambda - \text{ACTIVE}; -\tau]$, and therefore $f_{B_n \leftarrow A_n} = f_Z$. □

This lemma allows us to compute the probabilities by integrating the corresponding pdf-s:

$$P[A_n \leftarrow B_n] = \int_{\tau}^{\text{ACTIVE}-\lambda} f_Z(z)\, \mathrm{d}z$$
$$P[B_n \leftarrow A_n] = \int_{\lambda-\text{ACTIVE}}^{-\tau} f_Z(z)\, \mathrm{d}z$$

A last lemma provides the machinery to compute the probability of an entire chain, e.g., $P[{}^{i}\mathcal{D}_{X_n}]$. To this end, we first identify the probability density chain, and then we use numerical integration tools to compute the chain probability. Key to the process is the following

Lemma 3. *The joint density function of chains* ${}^{2k}\mathcal{D}_{A_n}$, ${}^{2k+1}\mathcal{D}_{A_n}$, ${}^{2k}\mathcal{D}_{B_n}$, *and* ${}^{2k+1}\mathcal{D}_{B_n}$ *are the functions* $f_{{}^{2k}\mathcal{D}_{A_n}} : \mathbb{R}^{2k} \to \mathbb{R}$, $f_{{}^{2k+1}\mathcal{D}_{A_n}} : \mathbb{R}^{2k-1} \to \mathbb{R}$, $f_{{}^{2k}\mathcal{D}_{B_n}} : \mathbb{R}^{2k} \to \mathbb{R}$, *respectively* $f_{{}^{2k+1}\mathcal{D}_{B_n}} : \mathbb{R}^{2k-1} \to \mathbb{R}$ *given by the following recursive formulas:*

$$f_{{}^{2k}\mathcal{D}_{A_n}}(\mathbf{X}) = f_{{}^{2k-1}\mathcal{D}_{A_n}}(\mathbf{X_{1:2k-1}}) \cdot f_1\left(\textstyle\sum_{i=1}^{2k}(x_i)\right)$$
$$f_{{}^{2k+1}\mathcal{D}_{A_n}}(\mathbf{X}) = f_{{}^{2k}\mathcal{D}_{A_n}}(\mathbf{X_{1:2k}}) \cdot f_2\left(\textstyle\sum_{i=1}^{2k-1} x_i\right)$$
$$f_{{}^{2k}\mathcal{D}_{B_n}}(\mathbf{X}) = f_{{}^{2k-1}\mathcal{D}_{B_n}}(\mathbf{X_{1:2k+1}}) \cdot f_3\left(\textstyle\sum_{i=1}^{2k}(x_i)\right)$$
$$f_{{}^{2k+1}\mathcal{D}_{B_n}}(\mathbf{X}) = f_{{}^{2k}\mathcal{D}_{B_n}}(\mathbf{X_{1:2k}}) \cdot f_4\left(\textstyle\sum_{i=1}^{2k+1} x_i\right)$$

where f_1, f_2, f_3, *respectively* f_4 *are densities built using the "$*$" and "$\circ$" operators,* x_i *are elements of vector* $\mathbf{X}$, *while* $\mathbf{X_{1:i}}$ *is a vector consisting of the first* i *elements of* $\mathbf{X}$.

Proof. We only give the proof for the chain ${}^{2k}\mathcal{D}_{A_n}$, as for the other chains we have a similar reasoning. For this chain, we exclude the case when $k = 0$.

First, remember that this chain is a shortcut notation for

$${}^{2k}\mathcal{D}_{A_n} \equiv B_{n-1} \leftarrow A_n \wedge A_{n-1} \leftarrow B_{n-1} \wedge \ldots \wedge A_{n-k} \leftarrow B_{n-k}$$

We associate a random variable to each of the elementary events, similarly to Lemma 2, that ultimately decides the probability for that event. For instance, to $B_{n-1} \leftarrow A_n$, we associate

$$\mathsf{X}_1 \triangleq \phi + \beta_{n-1} - \alpha_n$$

The probability of the chain can be expressed the joint probability of the elementary probabilities in the chain. For instance, one such elementary probability is $B_{n-1} \leftarrow A_n$ which, similarly to Lemma 2, can be further represented as:

$$P\left[B_{n-1} \leftarrow A_n\right] = P\left[\mathsf{X}_1 \in [\lambda - \textsc{on}; -\tau]\right]$$

For the rest of elementary events in the chain, we define:

$$\mathsf{X}_2 \triangleq \phi + \beta_{n-1} - \alpha_{n-1}$$
$$\vdots$$
$$\mathsf{X}_{2k} \triangleq \phi + \beta_{n-k} - \alpha_{n-k}$$

Notice that the probability of the chain is the probability that all the above are true, i.e.:

$$P\left[{}^{2k}\mathcal{D}_{A_n}\right] = P\left[\mathsf{X}_1 \in [\lambda - \textsc{on}; -\tau], \mathsf{X}_2 \in [\tau; \textsc{on} - \lambda], \ldots\right]$$

Consequently, we have the identity between the probability distribution functions $f_{{}^{2k}\mathcal{D}_{A_n}} = f_{\mathsf{X}_1,\ldots,\mathsf{X}_{2k}}$, where the latter is the joint distribution of X_1, …, X_{2k}. This can be further exploited as:

$$f_{\mathsf{X}_1,\ldots,\mathsf{X}_{2k}}(\mathbf{X}) = f_{\mathsf{X}_1,\ldots,\mathsf{X}_{2k-1}}(\mathbf{X_{1:2k-1}}) \cdot$$
$$\cdot f_{\mathsf{X}_{2k}|\mathsf{X}_1,\ldots,\mathsf{X}_{2k-1}}(x_{2k}|\,\mathsf{X}_1 = x_1, \ldots, \mathsf{X}_{2k-1} = x_{2k-1})$$

Here, $f_{\mathsf{X}_1,\ldots,\mathsf{X}_{2k-1}}$ is in fact $f_{{}^{2k-1}\mathcal{D}_{A_n}}$. Moreover, if we define $f_1 \triangleq f_{\mathsf{X}_{2k}|\mathsf{X}_1,\ldots,\mathsf{X}_{2k-1}}$, we obtain that $f_{{}^{2k}\mathcal{D}_{A_n}} = f_{{}^{2k-1}\mathcal{D}_{A_n}} \cdot f_1$. The first term, i.e., $f_{{}^{2k-1}\mathcal{D}_{A_n}}$, is part of the recursion step, as part of the hypothesis of this lemma. Thus, we turn our efforts towards finding an expression for f_1.

To this end, we introduce another variable Y defined as $\mathsf{Y} \triangleq \sum_{i=1}^{2k} \mathsf{X}_i$. Notice that the event "X_{2k} takes x_{2k} conditional upon $\mathsf{X}_1 = x_1, \ldots, \mathsf{X}_{2k-1} = x_{2k-1}$" is the equivalent of the event "Y takes $\sum_{i=1}^{2k} x_{2i}$". Thus, the following holds:

$$f_{\mathsf{X}_{2k}|\mathsf{X}_1,\ldots,\mathsf{X}_{2k-1}}(x_{2k}|\,\mathsf{X}_1 = x_1, \ldots) = f_{\mathsf{Y}}\left(\sum_{i=1}^{2k} x_i\right)$$

From probability theory we know that the pdf of the sum consisting of independent variables is the convolution of the variables' pdf. Consequently, given the definition of Y, the following stands:

$$f_{\mathsf{Y}} = f_{\mathbf{X}1} * f_{\mathbf{X}2} * \ldots * f_{\mathbf{X}2k}$$

Here, each of $f_{\mathbf{X}i}$ can be further expended using the "$*$ and "$\circ$" operators, e.g., $f_{\mathbf{X}1} = f_\phi * f_{\beta_{n-1}} \circ f_{\alpha_n}$. By transitivity, the formula above also applies to f_1, concluding thus our proof. $\square$

We can now integrate numerically to compute the probabilities associated to each chain. For instance:

$$P\left[{}^{2k}\mathcal{D}_{A_n}\right] = \int\limits_{-\tau}^{\textsc{on}-\lambda} \int\limits_{\textsc{on}-\lambda}^{\tau} \cdots \int\limits_{\textsc{on}-\lambda}^{\tau} f_{{}^{A}\mathcal{D}_{A_{n-k}}{}^{n}}(\mathbf{X})\, \mathrm{d}\mathbf{X}$$

Appendix B

Traffic Overhead in DICE

This section is an analytical framework supporting statements we make throughout Section 6.7. The main result in this section is Theorem 2. The theorem sets an lower bound for the average number of packets transmitted during a given time frame (counting from when a new maximum appears). We further refine this theorem in two corollaries, one describing the latency with which all the nodes in the network converge on the same view, the other describing the minimum amount of traffic when the attribute maximum is periodically updated.

Assumptions. The framework works on the following simplifying assumptions, necessary to make the problem tractable:

1. Communication is ideal, that is, it occurs without packet loss and without delays in the network stack.
2. The network already converged and the Trickle timers reached the maximum bound τ_h by the time the new maximum appears.
3. All nodes are *at most* d hops away from the node where the maximum appears. We further assume that the network topology is symmetrical w.r.t. to the node where the maximum appears.

Hereafter, we denote with N_0 the node where the maximum appears, and with N_i a node located i hops from N_0. The reference topology is illustrated in Figure B.1.

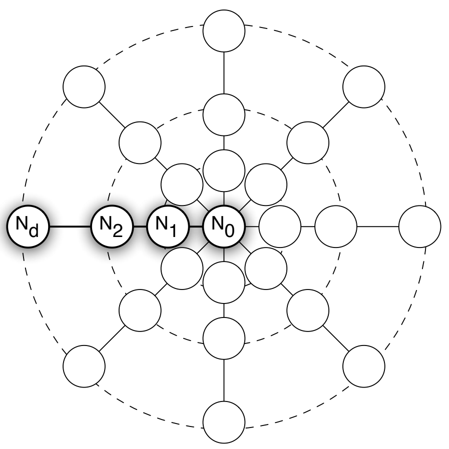


Figure B.1: The reference network topology used throughout this section. N_0 is the node where the maximum appears.

Theorem 2. *The lower bound for the average number of packets sent by all nodes in an interval t since the updated of the maximum is a function* $\mathsf{md}(t)$ *where*

$$\mathsf{md}(t) = \lg \frac{\sqrt[d+1]{\prod_{i=0}^{d} (t - (i-1) \cdot \tau_l)}}{\tau_l}$$

Proof. Recall that in Trickle broadcasts are scheduled in a geometric sequence with scale factor τ_l and common ratio 2. That is, the first broadcast occurs at τ_l, the second at $\tau_l + 2\tau_l$, the third at $\tau_l + 2\tau_l + 2^2\tau_l$, and so on. Therefore, in an interval no longer than t, the source of the update N_0 broadcasts an *integer* number of packets n_0 such that

$$\sum_{i=0}^{n_0-1} 2^i \cdot \tau_l \leq t$$

The packet with number $n_0 + 1$ is sent *after* t. Trickle timers increase exponentially, thus we have

$$t < \sum_{i=0}^{n_0} 2^i \cdot \tau_l = \left(2^{n_0+1} - 1\right) \cdot \tau_l \tag{B.1}$$

From the previous equation, we infer that

$$\lg \frac{t + \tau_l}{2\tau_l} < n_0$$

To compute the average, we need to determine how many packets are sent by each node. In this respect, we generalize the above inequality as follows. The update reaches nodes N_1, N_2, …, N_d with a latency of τ_l, $2\tau_l$, …, respectively $d\tau_l$. Therefore, the interval during which we count packets "shortens" for each node with a period equal to the previous latency. That is, N_1 has a period of $t - \tau_l$ during which it broadcasts packets corresponding to the maximum update, N_2 has a period of $t - 2 \cdot \tau_l$, and so on. We generalize Equation B.1 considering that a node N_j, where $0 \leq j \leq d$, broadcasts n_j packets such that

$$t - j \cdot \tau_l < \sum_{i=0}^{n_j} 2^i \cdot \tau_l = \left(2^{n_j+1} - 1\right) \cdot \tau_l$$

from which we deduce that

$$\lg \frac{t - (j-1) \cdot \tau_l}{2\tau_l} < n_j$$

We now compute the average number of packets $\overline{n}$. First, recall that the network is symmetrical, that is, from the perspective of node N_0, it has the same topology in all directions. Therefore, the traffic is expected to be the same irrespective of the direction, and therefore we can average only on the nodes N_j to obtain the network average $\overline{n}$:

$$\begin{aligned}\overline{n} = \sum_{j=0}^{d} \frac{n_j}{d+1} &> \frac{1}{d+1} \cdot \sum_{j=0}^{d} \lg \frac{t - (j-1) \cdot \tau_l}{2\tau_l} \\ &> \lg \frac{\sqrt[d+1]{\prod_{j=0}^{d} \left(t - (j-1) \cdot \tau_l\right)}}{2\tau_l}\end{aligned}$$

which concludes our proof. $\square$

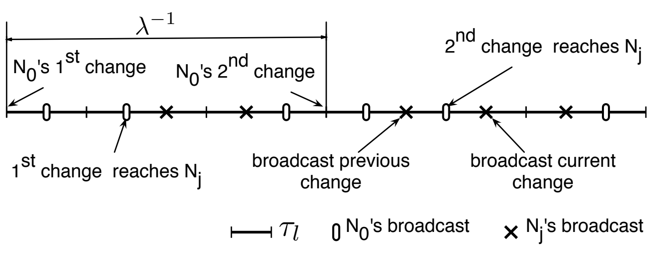


Figure B.2: Sample timeline showing broadcasts by nodes N_0 and $N_j = 3$ consequence of periodic maximum appearances.

An immediate consequence of the above theorem is the following

Corollary 1. *Nodes transmit on average at least* $\lg \sqrt[d+1]{(d+1)!}$ *packets by the time the update of node* N_0 *reaches the network fringe, i.e., node* N_d*.*

Proof. The proof follows immediately if we consider that the latency at which N_d is updated is $d \times \tau_l$ (each traversed hop bears a penalty of τ_l) and compute $\mathsf{md}\,(d \times \tau_l)$ according to Theorem 2. $\square$

We assume that the occurrence of new attribute maximum can be modeled using a Poisson process, which accurately models natural phenomena, but also human behavior. This approach allows us to characterize the network overhead in time, that is, given a maximum update rate, we can infer the number of packets sent per unit of time. In the following, we further assume that a new maximum appears *before* the Trickle timers reached their upper bound τ_h. Note that this is a relatively high occurrence of the new maximum values.

Formally, if λ is the maximum update rate, then a new maximum appears with a period λ^{-1}. Moreover, the upper bound of the Trickle timers is τ_h. Considering that the trickle timers are scheduled in geometric sequence, it takes no longer than $2 \cdot \tau_h - 1$ to reach the upper bound. Given the previous assumption, the inequality $\lambda^{-1} < 2 \cdot \tau_h - 1$ holds. With this, we enunciate the following

Corollary 2. *If the appearance of new attribute maximum can be modeled as a Poisson process with rate* λ*, and if* $\lambda^{-1} < 2 \cdot \tau_h - 1$*, then nodes transmit on average at least* $\lg \frac{\lambda^{-1}+\tau_l}{2\tau_l}$ *packets between two consecutive occurrence of a new maximum.*

Proof. Using the same reasoning as per Theorem 2, the source of the updates, i.e., node N_0, broadcasts n_0 packets such that

$$\lg \frac{\lambda^{-1} + \tau_l}{2\tau_l} < n_0$$

We claim that the above inequality holds for all nodes N_j. To justify our claim, consider the timeline depicted by Figure B.2. Here, we illustrate two maximum changes that occur on node N_0, evenly spaced by the duration λ^{-1} (dictated by our modeling of the phenomenon as a Poisson process). Notice that the first update takes $j \times \tau_l$ to reach node N_j, located j hops from N_0. Moreover, the second update takes an equal amount of time to reach N_j. Consequently, while the second update is propagating towards N_j, the later node is still propagating the first update. In result, the transmission schedule of N_j is the same as the one of node N_0, only that it is shifted in time with $j \times \tau_l$. Thus, N_j sends the same amount of packets as N_0 does. As this is valid for any node N_j, the minimum bound on the average is dictated by the inequality above. $\square$

The minimum packet rates follows immediately by multiplying λ with the lower bound set forth by Corollary 2.