Selective Reprogramming of Mobile Sensor Networks through Social Community Detection

Bence Pásztor[1], Luca Mottola[2], Cecilia Mascolo[1], Gian Pietro Picco[3], Stephen Ellwood[4] and David Macdonald[4]

[1] Computer Laboratory, University of Cambridge, UK
[2]Swedish Institute of Computer Science, Sweden
[3]Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, Italy
[4] Wildlife Conservation Research Unit, University of Oxford, UK

Abstract. We target application domains where the behavior of animals or humans is monitored using wireless sensor network (WSN) devices. The code on these devices is updated frequently, as scientists acquire in-field data and refine their hypotheses. Wireless reprogramming is therefore fundamental to avoid the (expensive) re-collection of the devices. Moreover, the code carried by the monitored individuals often depends on their characteristics, e.g., the behavior or preferred habitat. We propose a *selective* reprogramming approach that simplifies and automates the process of delivering a code update to a target subset of nodes. Target selection is expressed through constraints injected in the WSN, triggering automatic dissemination of code updates whenever verified. Update dissemination relies on a novel protocol exploiting the *social* behavior of the monitored individuals. We evaluate our approach through simulation, using real-world animal and human traces. The results shows that our protocol is able to capture the social network structure in a way comparable to existing offline algorithms with global knowledge while allowing runtime adaptation to community structure changes, and that existing dissemination approaches based on gossip generate up to 3 times more network overhead than our socially-aware dissemination.

1 Introduction

Wireless sensor networks (WSNs) are increasingly being used to monitor mobile entities in domains ranging from wildlife monitoring [16, 20] to human health-care [22]. In these contexts, WSN nodes are physically attached to animals or people being monitored. Therefore, unlike traditional WSN architectures where all nodes perform a single system-wide task, in these mobile WSNs the code running on a node is often specific to the monitored individual, and may change over time according to the individual's behavior or context. As an example, WSN devices attached to wildlife species (e.g., zebras [16], turtles [13], or badgers [10]) are currently used to study various aspects of their behavior. In the early stages of the deployment, all nodes monitor the same quantities for domain experts to get an initial insight, which can then be used to re-task some of the nodes to further study certain quantities. For instance, the devices carried by badgers that stay close to their burrows may be used to study the environment around the burrows themselves and explain why this subset of animals are following specific

paths in the forest instead of others, and how their movements depend on the climate. However, re-capturing the animals to manually re-program the nodes would be very costly, if at all possible.

Techniques for run-time reprogramming of WSNs do exist [33]. However, they fail to tackle two fundamental challenges of the application domain we target:

- The area where monitored individuals dwell is likely to extend beyond the communication range of current sensor devices. Thus, the network is most often characterized by *intermittent connectivity* among the mobile WSN nodes [23]. This prevents re-using well-established solutions for static networks [25].
- The few solutions addressing mobile WSNs disseminate code updates to the entire network, and are therefore ill-suited for a *selective* dissemination of code updates to a target subset. Indeed, the updates would reach more nodes than necessary, wasting resources and reducing lifetime.

On the other hand, animals and humans are *social* beings, with recognizable patterns of movement and community interaction, that can be exploited as a vehicle for delivering code to the intended targets. The core contribution of this paper is a novel approach to selective reprogramming in highly-disconnected, mobile WSNs that, based on the individual's interactions *detects communities at runtime, and exploits their existence and relationships towards efficient update dissemination*. For instance, a single WSN node attached to a badger known to roam often between two communities (i.e., a so-called "central" badger, with a socially-bridging role) can be enough to disseminate code from one community of badgers to the other. In our approach, communities are discerned entirely at run-time. This sets us apart from the few existing dissemination approaches based on social communities, that rely on offline centralized protocols [5] or are otherwise unable to adapt to all changes to the social community structure [7,14,31].

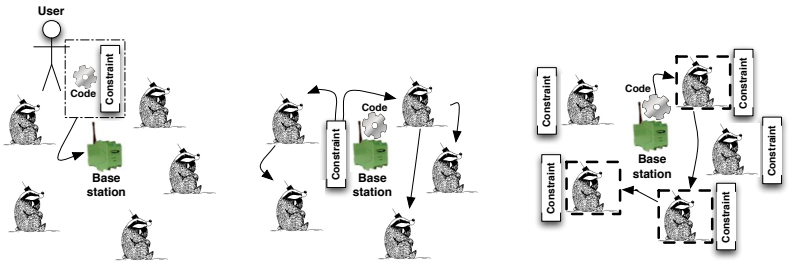
An overview of our approach is provided in Sec. 2, where we introduce a sample scenario showing how a user can target a set of nodes of interest. In Sec. 3, we give details of how the protocol is able *automatically* select these nodes, and deliver the code efficiently.

In Sec. 4, we evaluate the effectiveness of our solution through simulations using animal and human traces collected in real-world experiments. We review related approaches in Sec. 5, and provide brief concluding remarks and directions for future work in Sec. 6.

2 Reference Scenario and System Overview

We illustrate the overview of our approach hand-in-hand with a reference scenario that provides the main application focus for the entire paper. Although the scenario is drawn from the wildlife domain, our techniques are applicable to other mobile WSN scenarios, as we show in Sec. 4 by applying them to human interaction traces. Next, we describe how users specify persistent, network-wide constraints identifying the subset of nodes targeted by reprogramming.

Reference scenario. Fig. 1 depicts the phases of our reprogramming approach in a reference scenario concerned with badger monitoring. As shown in Fig. 1(a), reprogramming entails generating a bundle containing *i)* the code update to be installed on a target



(a) (b) (c)

Fig. 1. Sample scenario showing: (a) code and constraint injection at base station, (b) constraint dissemination to all nodes, and (c) delivery of code to selected nodes (in dashed squares).

subset of the WSN, and *ii)* the *constraint* that identifies these target nodes by means of logical expressions involving their properties. For instance, the constraint may single out only the nodes attached to badgers that spend most of their time close to a cold burrow. The constraints are encoded in periodic beacons for transmission. The bundle is then injected at the base station, or at any other node.

The two constituents of the bundle have a different fate, as show in Fig. 1(b). The constraint is spread to all WSN nodes. Upon reception, a node matches the constraint against its local state, and re-evaluates it periodically. The code update, on the other hand, remains at the base station until at least one node matches the constraint. When this happens, our socially-aware protocol (described in Sec. 3) disseminates the code update only to the target nodes matching the constraint, as shown in Fig. 1(c).

It is important to note that reprogramming can be requested even when no node matching the characteristics specified by the constraint currently exists. In the mobile setting with intermittent connectivity we target, it would be difficult (if not impossible) for users to know and await the moment when the target subset is not empty. Our solution enables users to rely on the system to detect the presence of target nodes *automatically*, by self-adapting to changes in the state of nodes. For instance, one might define constraints to target nodes roaming around different burrows, and inject the code before any node satisfies the constraint. The code will stay at the base station until such behaviour is detected, and will be delivered automatically.

Specifiying constraints. The constraints identifying the target subset are expressed through dedicated constructs. We characterize the state of nodes based on *attributes*. These are name-value pairs describing properties of a node, e.g., the current location or the gender of the individual it is attached to. The construct **`attribute`**`(NAME)` declares an attribute, registered by the run-time layer that takes care of updating the associated value. For instance, in the case of a `LOCATION` attribute, the run-time periodically queries the attached GPS device, and stores the value time-series in memory.

Selecting badgers that stay around cold burrows can be specified as

```
constraint(n_occurrence(LOCATION == burrow) > loc_threshold &&
           avg(TEMPERATURE) < temp_threshold)
```

where `LOCATION` and `TEMPERATURE` are attribute names, and `burrow` is an encoding of a burrow's location in some coordinate system. The built-in functions **`avg`** and **`n_occurrence`** are made available by the underlying run-time support: the latter returns the number of occurrences in an attribute's time series that match the boolean

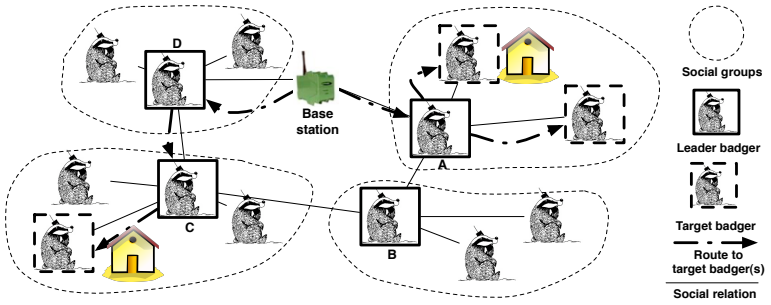


Fig. 2. Example of social communities and their leaders. The target subset includes nodes often visiting a specific area.

condition given as argument. We provide several built-in functions (e.g., **`avg`**, **`max`** and **`min`**) covering a range of common constraints.

A constraint essentially specifies a boolean function that establishes the membership of a node in a given subset (`constraint(TRUE)` targets the entire WSN). This addressing scheme is well-suited to our scenarios where the target subset changes based on the *state* of nodes—that we capture through attributes, and could hardly be captured through node identifiers. Similar approaches exist in the literature [25, 34]. However, their supporting communication layer targets only static WSNs, while we bring the expressive power of attribute-based node selection into mobile WSNs, as discussed next.

3 Socially-aware Dissemination of Code Updates

Once the appropriate constraint is stored at the base station, the problem is to efficiently disseminate the code update to the corresponding target nodes. In principle, this could be done using direct transmissions, however in our scenario, we cannot ensure that all the nodes come in range of the base station due to the limited power and radio range of the devices.

In our dissemination protocol, code updates are relayed opportunistically from one animal to the other upon contact. However, unlike existing approaches that propagate updates to the entire network, we limit dissemination as much as possible to the target nodes. This substantially reduces the network overhead, as evaluated quantitatively in Sec. 4. To achieve this goal, we use a characteristic common to many mobile WSN scenarios, namely, the fact that the monitored individuals exhibit *social behavior*. The implicit structure of social interactions, once elicited, provides an effective tool for steering efficient routing decisions. In the rest of this section we describe the aspects of social interaction that are relevant to our goals, along with the way we exploit them in our dissemination protocol. The social foundation of our protocol hold for many animal species [2] [30], including humans [14].

3.1 Overview

Social foundation. A social network is a logical structure of entities tied by some social relation, e.g., friendship. These networks are characterized by strong clustering [7, 17].

Members of a cluster, or *community*, are usually closer to each other socially, than to the rest of the network – i.e., they interact more and spend more time together. Communities tend to be stable over time, although they occasionally vary. An example is animals sharing the same burrow or foraging in the same areas: when cubs grow up, at some point they separate and move to a different area. Moreover, not all members of a community behave the same way, some animals/people are more active or popular than others.

We use the highly mobile and more socially central members to aid the dissemination, since they are more likely to meet other individuals. We call these nodes *leaders*.

Protocol operation. We assume that the base station, where the bundle containing the code update and constraint reside (Sec. 2), is placed in an area where one or more animals dwell. Animals identified as *leaders* are used to carry code updates to communities where at least one member is in the target subset as shown in Fig. 2.

Our protocol *dynamically* identifies communities and leaders in a fully decentralized way, as discussed next. As illustrated in Fig. 2, communities and leaders determine a logical topology where links represent spatio-temporal relations between two individuals, essentially denoting that they are frequently co-located. We exploit these links to disseminate the code updates according to the forwarding rules described in Sec. 3.4.

3.2 Identifying Communities

Social foundation. Members of the same social community are co-located according to a regular pattern and for long periods of time. For instance, at night, badgers roam independently. During the day, however, they tend to congregate around in burrows, where they sleep. Animals using the same burrow tend to spend considerable time together and are therefore often associated to the same community. Our definition of community is a set of nodes spending a certain percentage of their time together.

Protocol operation. To identify communities, we need to quantify the extent of co-location between nodes. To do so, nodes send periodic beacon messages to discover neighbors. Upon receiving a beacon, a node increments by the beacon interval the *contact time* relative to the sending neighbor. This quantity is divided by the time since the first detection of the same node, yielding a *contact ratio* measuring how frequently the two nodes are co-located. Higher ratios indicate more frequent co-location. As time elapses, the contact ratio becomes an accurate indicator of the amount of interaction between two animals. This metric is better at capturing dynamic changes in the community structure than the often-used total-contact duration [14], since it captures not only the order of encounters, but also is able to decay if two nodes become separated.

To create and maintain communities, all nodes send periodic beacons and evaluate each other's contact ratios, which are embedded within these beacons. Two nodes are considered part of the same community when their contact ratios cross a given threshold. For instance, the aforementioned behavior of badgers, sharing the same burrow for about half of the day, can be modeled by setting a 50% threshold. This indeed corresponds to nodes that are in contact for about half of the time. Thresholds are expected to be defined by domain experts, e.g., based on the species under study. If the ratio crosses the threshold and neither node is yet part of a community, the node with the

smaller identifier creates a unique community identifier and includes it in subsequent beacons; the other node joins the new community upon receiving the beacon. If either node is already part of a community, the other joins the same one. If they belong to different communities, the node in the community with fewer members joins the larger one. To enable these decisions, beacons also carry the community size. Our mechanism captures the time evolution of social relations among individuals as nodes can join and leave communities.

An important observation is that the dissemination protocol uses one layer of clustering. More precisely, a node is either a member of a community or not, we do not consider nodes belonging to multiple communities. One can argue that this applies to animals [2], but not for humans. While similar approaches have been adopted for human networks [14, 31], human social structures are more complex. If the target application heavily involves membership in multiple communities, our protocol would need to be properly extended to cater for it.

3.3 Identifying Leaders

Social foundation. The behavior of members of the same community may differ [29]. Moreover, this behavior can change over time. For instance, during mating season, adult male badgers travel further from their burrow than other community members, looking for females to mate. Therefore, they are more likely to meet badgers from other communities.

Protocol operation. To accurately and dynamically identify leaders within a community, every node keeps track of two quantities:

- Its *total neighbor count* N, i.e., the number of all distinct nodes it has ever met.
- Its *change-degree of connectivity* C, i.e., the number of neighbors it acquires or looses within a time window.

The two metrics account for different aspects, and leaders should score high in both. For instance, a node with high neighbor count can probably reach many members of its community. The same node, however, may have a low value of change-degree of connectivity, e.g., if it does not move often. This node is not well-suited as a leader. The relative weight of the two metrics must be tuned by domain experts based on the species under study. This is achieved by defining a single *leader score* as $L = \alpha N + (1 - \alpha)C$, and properly setting the weight α. In this paper, unless otherwise noted, we use $\alpha = 0.5$. In principle, other metrics could be used, e.g., ego-centrality and betweenness [7]. However, our priority was to disseminate updates as quickly as possible, therefore we focused on identifying the most mobile nodes. Further, an improvement on the neighbour count metric is to use a sliding time window, and consider the neighbour count in this window only. Though we did not use this method in this paper, it is our intention in the future.

Nodes that do not belong to any community or are not associated with a leader (e.g., at start-up or when the community threshold is not reached) are considered leaders of a fictitious community of size one. When a real community with more than one member is created, the node with the highest score L becomes its leader. The identifier

of community leader and its score are embedded within beacons, and broadcast to the 1-hop neighbours of the leader, while nodes who are in direct contact with the leader beacon a score $L = 0$. This ensures that each node in a community is logically one hop away from a leader, since the node with the local maximum score is always chosen. If a node in a community finds its score to be higher than that of the current leader, it takes over the leadership. The same processing applies when a node joins a community.

Leaders do not need to be unique in a community. Although an unlikely situation, it may happen that the leader identifier and score are too slow to disseminate for this information to stabilize. Nonetheless, the presence of multiple leaders with similar scores is not problematic in the dissemination process, described next.

3.4 Code Dissemination

The process of disseminating code updates is logically divided in two steps. First, the opportunistic routes leading to nodes in the target set are determined. Then, the actual code is disseminated along these routes. In practice, however, the latter step is pipelined with the former to reduce latency.

Route establishment. The routes are determined by the constraint selecting the target subset. Constraints, encoded in a compact form, are disseminated to all nodes in the network by piggybacking them on beacons. Upon receiving a constraint, a node evaluates whether it belongs to the target subset. If so, it informs its community leader whenever in range.

LeaderID	Target	NextHop	Distance
A	Yes	Base	2
B	No	C	2
C	Yes	C	1
D	Yes	-	-

Fig. 3. Routing table of node D in Fig. 2.

Leaders use this information to build routing tables like the one in Fig. 3, based on the network shown in Fig. 2. Besides a leader's own entry, the table is populated by exchanging entries with other leaders whenever they meet, through the periodic beacons. The `Target` field indicates whether at least one member in a leader's community is targeted by the constraint. The `NextHop` field identifies the leader that forwarded a given entry. The `Distance` field is the hop-count measure of how "far" a leader is. Multiple constraints can be disseminated in parallel, distinguished by a unique identifier carried by beacons and used to index multiple routing tables at each node.

Update dissemination.

Update dissemination is governed by the following rules:

- a non-leader can only update its own leader;
- a leader can only update other leaders and the members of its own community.

These rules ensure an efficient dissemination, as shown in Sec. 4, as well as consistent delivery. All leaders (including nodes without a community) receive the update. All other nodes in the target set (i.e., the community members) receive the update from their leader.

Updates follow the routes stored in the leaders' routing tables. Consider for instance Fig. 3. When node D receives an update to be disseminated, it determines through the `Target` field that some of its community members are selected by the constraint, along

with members of C's and A's communities. To deliver the update to the selected community members, D waits until it becomes co-located with a sufficient number of community members that require the update (i.e. it receives beacons from these members). These can then receive it simultaneously through broadcast, reducing the communication overhead.

This makes sense for species where the probability of colocation is reasonably high. However, this policy may be revised and the leader could decide to broadcast more often, for example when a given percentage of the required members are present. To reach A and C, D looks at the `NextHop` field in its routing table: the code update is forwarded the next time D meets with C or the base station, respectively.

As constraints are piggybacked on beacons, they propagate faster than code, which is often larger. The routing tables are therefore usually built before the code arrives. If not, the code is buffered until at least one positive value appears in the `Target` field.

Short-lived vs. persistent updates. Constraints and code updates are associated to a version number and a time-to-live (TTL). The version number avoids duplicate delivery. Constraints are re-evaluated periodically and the corresponding entries in the routing table are retained until the TTL expires. When a node matching the constraint is detected, our protocol automatically starts the code dissemination following the mechanisms described. Along with this *short-lived* updates, which disappear from the network after a given time, we also easily support *persistent* updates by setting an infinite TTL. In this case, our scheme caters for a powerful way to make the system self-adapt.

3.5 Implementation Highlights

Our current prototype is based on the Contiki [9] OS, targeting TMote Sky nodes. The system is composed of three core components. A `Communication` component is responsible for building and maintaining routing information. Specifically, it maintains the neighbor table, calculates the contact ratio for every neighbor, and maintains information on the leaders. In addition, the module is also responsible for the reliable delivery of the code updates. To do so, we use a simple broadcast mechanism based on a RTS/CTS mechanism and acknowledgments sent back by the target nodes. A `Constraint Evaluator` module parses received constraints and checks them against the current values of node attributes. This determines whether the local node is included in the target subset. Finally, a `Reprogramming` module dynamically links received code updates (typically of size 2-10 Kb) using the hooks available in Contiki.

4 Evaluation

We first compare the effectiveness of our distributed community detection protocol against a centralized algorithm based on global knowledge of the social graph. The two schemes have similar performance in terms of communities detected, yet our distributed solution is able to detect dynamic changes in the community structure. Next, we assess how community knowledge improves code dissemination. Based on this, our protocol reduces network traffic by a factor of 66% compared to a gossip protocol.

General settings. We use the Reality Mining traces [12] and mobility traces from a badger-monitoring deployment [10]. We used a one month subset of both. The former include proximity information gathered using 43 mobile phones carried by people moving on a university campus. The latter are collected from the movements of 32 badgers equipped with RFID collars and 28 RFID readers deployed in a forest. These data include time-stamped detection of animals by readers at specific places. Therefore, there would be no explicit information on the connectivity *between* the RFID tags carried by the animals. We convert these traces into connectivity information by considering the nodes within wireless range when the animals are detected by the same RFID reader within a 5-minute time-window. Further, we assume animals stay at the burrow between the time they enter and exit - even though the RFID is unable to detect them underground.

The traces present a different radio model from the traditional WSNs, however here we are more interested in the social model governing the movements of the nodes, rather than modeling the radio, and the these traces are ideal for the former.

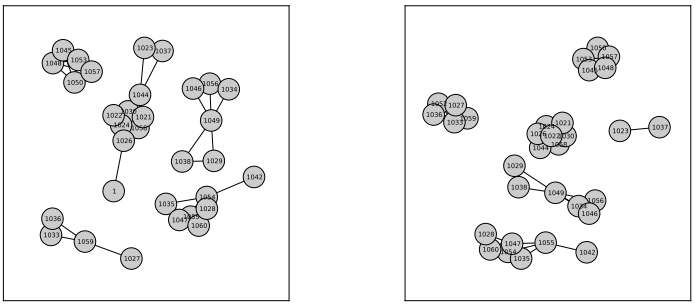
We use the Cooja simulator [27], along with a plug-in we implemented to replace the propagation model in the simulator based on the aforementioned mobility traces. In the community detection protocol, we set the community threshold to 50%. We chose this threshold based on the trace set: badgers sleep during the day in their burrows, therefore they are co-located for at least half a day every day. The threshold is also a good representation of human contacts: if two people spend more than half of their time together, they are more than likely to belong to the same social group. An investigation of the effect of the choice of the threshold is reported later in this section.

4.1 Community Detection

On the badger trace set we compare the performance of our community detection protocol against a well-known algorithm based on modularity optimization [1]. This runs in a centralized fashion and requires global topology knowledge. The communities identified by this algorithm largely reflect the findings obtained through direct observation by the zoologists involved in the study.

Modularity optimization algorithm. Given a specific partitioning of a graph, modularity measures the density of links inside every partition with respect to links between partitions. Higher values correspond to configurations with dense connections inside partitions and sparse connections between different ones. When applied to the study of social networks, partitions are naturally mapped to communities.

The algorithm we consider explores different community configurations to optimize modularity. Initially, every node is in its own community. For every pair of nodes, the algorithm examines the modularity gain obtained by moving either of the two nodes in the other's community. The communities are then changed to maximize this gain. This process repeats for every pair of nodes until no further improvements are achieved. Next, the algorithm creates a new graph with nodes which are the communities found earlier, and the link weights are the sum of the weights of links between the original nodes in the two communities. The algorithm then re-applies the first step on the new graph. The process continues until no further improvements are possible.



(a) Communities after five days. (b) Communities after twenty days.

Fig. 4. Communities found using the contact ratio as metric for link weight.

The input to the algorithm is a social graph where there is a link between two nodes if they meet at least once during the simulation time, and the link weights are the ones calculated by our protocol.

Results. We consider different points in time in the badger trace set. Our solution uses the contact ratio to detect dynamic changes in the community structure, therefore, we run the centralized algorithm using this figure as link weight. In this case, both schemes identify the *same* communities after one, five, and twenty days of traces. The communities found after day five are shown on Fig. 4(a). Nevertheless, our distributed solution runs *inside* the network. The centralized algorithm, on the other hand, may run only at the fringes of the system because of significant computational demands. In addition, it would require periodic topology discovery to provide global information as input. This is hardly possible in a mobile scenario with intermittent connectivity.

Even if the conditions to run the centralized algorithm were satisfied, however, the distributed nature of our scheme brings a unique advantage: that of immediately recognizing changes in the community structure. For instance, in the badger scenario the community structure does not change much after day five. This might appear as the long-term behavior. However, by day twenty we see a new community emerging, as shown in Fig. 4(b). Our scheme immediately detects this change, as it is running right on the WSN devices whose behavior caused the formation of an additional community. The centralized approach would identify the new community with significant latency and high overhead, due to the need of periodically collecting global topology information.

4.2 Code Dissemination

We study the performance of our selective code dissemination protocol against state-of-the-art solutions. We compare our approach against:

– the *GCP* [4] gossip protocol for code propagation in mobile sensor networks. This protocol is agnostic of selective dissemination and distributes the update to every

node. To do so, it uses a token-based mechanism to limit the number of transmissions per node, forwarding a code update to any node in range provided the sender still has tokens to spend.
- a constraint-based gossip protocol we implemented. Like ours, this uses the constraints to identify the nodes requiring a code update. The difference with ours is the lack of community knowledge. A node forwards a code update to a nearby device only if *i)* the neighbor belongs to the target subset, or *ii)* the neighbor met a node in the target subset within a specified period (set to half a day). The latter is required to reach nodes in the target subset that may never be in contact with a sender.

Using version numbers, neither protocols transmit a code update if the intended receiver is already equipped with it.

Settings and metrics. A code update consists of a variable number of packets. Each packet is 128 bytes long. We inject the code update at a random node 5 days after start-up. This delay is necessary for the communities to stabilize. We define the target subsets as a given percentage of nodes out of the total. Based on this value, each simulation run considers a different subset to avoid biases due to the subset chosen. GCP is equipped with 15 tokens per node, after we experimentally verified that this value provides a good trade-off between network traffic and overall delivery. For all protocols, we used a one-minute beacon interval for neighbor discovery.

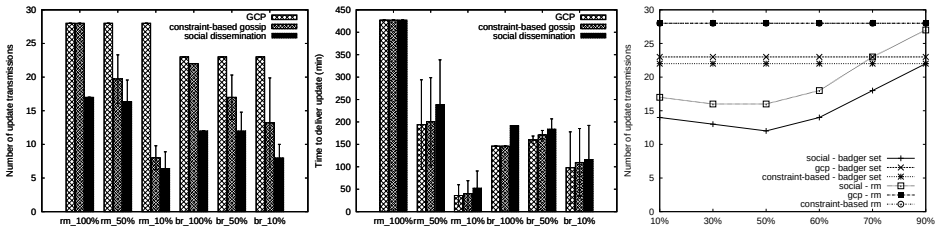
Based on this setting, we measure the following quantities:

- The code update *delivery*, defined as the fraction of nodes in the target subsets that receive the code update. This essentially measures to what extent the dissemination protocol achieves its goal.
- The number of code update *transmissions*, namely the number of bulk data transfers performed during a simulation. This indicates the cost—at the network level—to reach the protocol goal.
- The *latency* required to reach the nodes in the target subset, which provides a complementary measure of cost from a user perspective.

We considered message transmissions as opposed to radio-on-time to evaluate the energy cost of our protocol. Our protocol does not assume that the radio is always on, and is independent of any underlying MAC protocol duty cycling the radio, as long as it provides the ability to discover neighbors and to perform bulk-transfers. There are already efficient MAC protocols for WSN such as [3, 11], and it is also easy to see how the social cluster information could be used for duty cycling the nodes - this is however subject of a future work. Further, we do not consider beacons, as all three protocols send them at the same rate. All protocol messages are embedded in beacons, therefore they do not pose additional overhead (the beacons of GCP are, however, 21 bytes lighter).

We run 20 repetitions for each setting. The following results are averages over these repetitions, while the error bars represent the standard deviation around the average.

Results. Hereafter, we show results obtained with code updates of 10 packets. We verified that changing this figure within the range of 5-20 does not influence our results. This is because the bulk transfer of a code image takes little time compared to node mobility, and always completes before the two nodes disconnect.



(a) Overhead (b) Latency (c) Threshold

Fig. 5. Overhead, latency and the effect of clustering threshold of our protocol compared to GCP and constraint-based gossip.

In all simulations, the three protocols always deliver the code update to all nodes in the target subset. To do so, however, they incur in drastically different costs at the network level. Figure 5(a) shows the number of code update transmissions against varying target subsets. On average, our community-based protocol improves by a factor of 3.1 and 1.8 over GCP and constraint-based gossip, respectively. However, the gains are smaller as the cardinality of the target subset decreases. This is because the leader nodes that carry code around are a fixed cost that we must pay to reach every part of the system. The impact of this cost is greater as the target subset is smaller. As expected, GCP exhibits the same performance regardless of the target subset. Indeed, it stops only when all nodes are reached, even if the ones in the target subsets already received the code update. constraint-based gossip improves on this behavior, as it may stop earlier if there are no more nodes in the target subset requiring the code update.

To achieve this performance, the community-based protocol trades-off transmissions for latency. The latter is shown in Figure 5(b). Nevertheless, the increased latency in our protocol is limited given the absolute values at stake. On average, we have an increase of a factor of only 1.3 in delay compared to GCP, while the worst case is an increase of a factor of 2.6. GCP shows the best performance in this metric, as it has no restrictions on when to forward a code update. Therefore, it takes advantage of every opportunity, at the cost of redundant transmissions. In our protocol, instead, the leader node knows which nodes in its community need the update, therefore it can wait until it is collocated with these nodes. Once they are all in range, the leader node can update them in one go using broadcast transmissions.

In presence of intermittent connectivity, it may take a long time for some nodes to receive the updates. In the case of targeting 50% of the Reality Mining trace set, this results in a large variation in latency, but some variation is also observed in other cases. This is a characteristic of the network, and affects all three protocols.

We also investigate the behavior of leader nodes, as they play a critical role in our solution. Particularly, we study whether their use may lead to an uneven degradation of available energy among the nodes, e.g., because leaders need to handle more network traffic. To do so, we examine the average number of code updates that leader nodes deliver in our solution, compared to the number of nodes in GCP and constraint-based

gossip that deliver an update at least once. We found that the average number of update transmissions a leader sends is 2.3 in the reality mining and 1.2 for the badger trace set. Both GCP and the constraint-based approaches send 3 and 1.3 update transmissions on average per node, for the reality mining and badger trace set, respectively. We conclude that the leader nodes are *not* depleting their resources more quickly compared to other solutions. Particularly, the leaders we identify largely correspond to nodes that—because of the patterns of colocation—would deliver the code updates anyways. On average, 87% of leader nodes deliver code updates also in GCP and constraint-based gossip. However, our community-detection mechanism identifies *apriori* such nodes. By doing so, we can make them wait for a good opportunity, e.g., when they are in contact with members of their community, to save on unnecessary transmissions.

To further study the effect of leader selection, we also compared our results from targeting the entire network to the performance of the same protocol with random leader selection. This scheme selects leaders randomly from the members of each group. While our overhead is 56% of that of GCP when targeting the entire network, averaged over the two trace set, the random leader selection uses 84%. It is still better than GCP, since the protocol can take advantage of the colocation of the community members, though uses more than necessary transmissions to deliver the code to the communities.

We have also analyzed the effect of the threshold on which communities are separated, which for this analysis has been 50%. In Fig. 5(c) we plot the number of updates sent by all three protocols on both the reality mining and the badger datasets, targeting the entire network. As it can be seen, the threshold choice does affect our results: a different threshold means different community structures and a different number of leaders, thus leads to different overhead. Note, however, that even in the case of bad choices of thresholds, the performance falls back to that of the gossip-based protocols.

5 Related Work

Social routing. A few recent approaches leverage social-inspired metrics for routing. SimBet [7] achieves efficient data dissemination by exploiting "betweenness", a measure of how an individual may socially connect other entities not necessarily known to each other. Bubblerap [15] and Island Hopping [31] use a centralized algorithm to detect communities, based on global knowledge. In Bubblerap's distributed extension, it detects communities at run-time only if their cardinality grows over time. Thus, every node is bound to the first community it is mapped to, missing the dynamic evolution of social interactions.

In contrast to these approaches, our solution detects communities at run-time and in a fully decentralized fashion. Moreover, we are able to adapt to dynamic changes in the community structure and in the mapping of entities to communities. These features are pivotal to leverage communities for routing in the scenarios we target.

Delay tolerant routing approaches use notions of previous encounters and mobility patterns to decide on best message carriers [19,28,32]. This approach was also extended to mobile sensor networks [28]: while the use of mobility and connectivity to identify good carriers is shared in our approach, with respect to dissemination we go one step

further and use community knowledge to improve on the number of messages needed to spread the updates.

WSN reprogramming. To the best of our knowledge, our work is the first to provide a solution for *selective* code dissemination in *mobile* sensor networks. However, the literature includes a wealth of approaches for system-wide reprogramming in static networks [33]. For instance, Trickle [18] disseminates code updates using a "polite gossip" technique to suppress redundant transmissions. The rate of control traffic is adjusted at every device based on the state of neighbor nodes. As neighborhoods keep changing in the scenarios we target, a similar solution would be very inefficient.

Solutions for selective code dissemination in static networks also exist. For instance, Figaro [26] allows selecting subsets of nodes based on node attributes. It employs a tree-based routing scheme for code dissemination, which is difficult to apply in a mobile, disconnected scenario like ours. In TinyCubus [24], code is disseminated to all nodes with a given role, e.g., all cluster-heads. At the network level, TinyCubus assumes a priori knowledge of the system topology, as it requires to specify an upper bound on the number of hops separating nodes with the same role. Such scheme is hardly applicable in presence of dynamic topologies and intermittent connectivity.

In a mobile setting, Impala [21] leverages gossip dissemination to distribute code updates to every device. Version numbers are used to cater for eventual delivery. GCP [4] also targets system-wide reprogramming in mobile sensor networks, using a polite gossip technique similar to Trickle. However, GCP limits network traffic using a token-based scheme whereby nodes can transmit only if they possess enough tokens. ReMo [8] focuses on both static and mobile networks, using physical-layer metrics such as the Link Quality Indicator (LQI) [6] to establish routes for code dissemination. Although these solutions target scenarios similar to ours, they still do not tackle the problem of selective code dissemination. Therefore, being unaware of the selection criteria specified by our users, their use would correspond to significant energy waste.

6 Conclusion

We presented a system for selective reprogramming in mobile WSNs, based on social community detection. Our solution allows users to target a subset of the WSN nodes using constraints on node attributes. A dedicated protocol exploits the social interactions among the monitored entities to disseminate code updates efficiently. We evaluated our framework through real mobility traces. The results showed that, although experiencing a small latency overhead, our protocol saves up to 66% of the transmissions even when reprogramming targets the entire system. These performance gains increase when targeting a subset of the nodes, by virtue of our routing strategy that builds routes to the target nodes based on the social communities. Our future work includes deploying the system on animals in the context of a wildlife monitoring project.

7 Acknowledgments

The work described in this paper was partially supported by ESF MiNEMA, EPSRC grants EP/E012914 and EP/C544773, the Autonomous Province of Trento under the

call for proposals “Major Projects 2006” (project ACube), CONET under EU contract FP7-2007-2-224053 and Swedish Foundation for Strategic Research (SSF).

References

1. V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *J.STAT.MECH.*, page P10008, 2008.
2. J. L. Brown and G. H. Orians. Spacing patterns in mobile animals. *Annual Review of Ecology and Systematics,*, 1, 1970.
3. M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *SenSys '06: Proc. of the 4th Int. Conf. on Embedded Networked Sensor Systems*, pages 307–320, New York, NY, USA, 2006. ACM.
4. Y. Busnel, M. Bertier, E. Fleury, and A.-M. Kermarrec. GCP: Gossip-based Code Propagation for Large-scale Mobile Wireless Sensor Networks. In *Proc. of the Int. Conf. on Autonomic Computing and Communication Systems*, 2007.
5. S.-Y. Chan, P. Hui, and K. Xu. Community Detection of Time-Varying Mobile Social Networks. In *Proc. of the First Int. Conf. on Complex Sciences: Theory and Applications (Complex 2009)*, 2009.
6. Chipcon Tech. CC2420 Datasheet. `focus.ti.com/docs/prod/folders/print/cc2420.html`.
7. E. M. Daly and M. Haahr. Social Network Analysis for Routing in Disconnected Delay-tolerant MANETs. In *Proc. of the Int. Symp. on Mobile Ad-Hoc Networking and Computing (MobiHoc)*, 2007.
8. P. De, Y. Liu, and S. K. Das. ReMo: An Energy Efficient Reprogramming Protocol for Mobile Sensor Networks. In *Proc. of the Int. Conf. on Pervasive Computing and Communications (PERCOM)*, 2008.
9. A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proc. of* 1^{st} *Wkshp. on Embedded Networked Sensors*, 2004.
10. V. Dyo, S. A. Ellwood, D. W. Macdonald, A. Markham, C. Mascolo, B. Pasztor, N. Trigoni, and R. Wohlers. Poster Abstract: Wildlife and Environmental Monitoring using RFID and WSN Technology. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems (SenSys)*, 2009.
11. V. Dyo and C. Mascolo. Efficient node discovery in mobile wireless sensor networks. In *DCOSS '08: Proc. of the 4th IEEE Int. Conf. on Distributed Computing in Sensor Systems*, pages 478–485, Berlin, Heidelberg, 2008. Springer-Verlag.
12. N. Eagle and A. S. Pentland. Reality mining: sensing complex social systems. *Personal Ubiquitous Comput.*, 10(4), 2006.
13. A. Gorlick. Turtles to test wireless network, July 2007.
14. P. Hui, J. Crowcroft, and E. Yoneki. Bubble rap: Social-based Forwarding in Delay Tolerant Networks. In *Proc. of the Int. Symp. on Mobile Ad-Hoc Networking and Computing (MobiHoc)*, 2008.
15. P. Hui, E. Yoneki, S. Y. Chan, and J. Crowcroft. Distributed community detection in delay tolerant networks. In *Proc. of Int. Wrkshp. on Mobility in the Evolving Internet Architecture (MobiArch)*, 2007.
16. P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *Proc. of the Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, 2002.

17. J. Krause, D. Croft, and R. James. Social Network Theory in the Behavioural Sciences: Potential Applications. *Behavioral Ecology and Sociobiology*, 62(1), 2007.
18. P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: a Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *Proc. of the Symp. on Networked Systems Design and Implementation (NSDI)*, 2004.
19. A. Lindgren, A. Doria, and O. Schelén. Probabilistic Routing in Intermittently Connected Networks. In *Proc. of the Int. Wkshp. on Service Assurance with Partial and Intermittent Resources (SAPIR)*, 2004.
20. A. Lindgren, C. Mascolo, M. Lonegan, and B. McConnell. Seal2Seal: A Delay-Tolerant Protocol for Contact Logging in Wildlife Monitoring Sensor Networks. In *Proc. of Int. Conf. on Mobile Ad-hoc and Sensor Systems (MASS)*, 2008.
21. T. Liu and M. Martonosi. Impala: A middleware system for managing autonomic, parallel sensor systems. In *Proc. of the SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2003.
22. K. Lorincz, B.-R. Chen, G. Werner Challen, A. Roy Chowdhury, S. Patel, P. Bonato, and M. Welsh. Mercury: A Wearable Sensor Network Platform for High-fidelity motion Analysis. In *Proc. of the Int. Conf. on Embedded Networked Sensor Systems (SenSys)*, 2009.
23. M. Lukac, L. Girod, and D. Estrin. Disruption Tolerant Shell. In *Proc. of the SIGCOMM Wkshp. on Challenged Networks (CHANTS)*, 2006.
24. P. J. Marrón, A. Lachenmann, D. Minder, J. Hahner, R. Sauter, and K. Rothermel. TinyCubus: a flexible and adaptive framework sensor networks. In *Proc. of the European Wkshp. on Wireless Sensor Networks (EWSN)*, 2005.
25. L. Mottola and G. P. Picco. Logical Neighborhoods: A Programming Abstraction for Wireless Sensor Networks. In *Proc. of the Int. Conf. on Distr. Computing in Sensor Systems (DCOSS)*, 2006.
26. L. Mottola, G. P. Picco, and A. Amjad. Fine-Grained Software Reconfiguration in Wireless Sensor Networks. In *Proc. of European Conf. on Wireless Sensor Networks (EWSN)*, 2008.
27. F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level Simulation in COOJA. In *Proc. of the European Conf. on Wireless Sensor Networks (EWSN)*, 2007.
28. B. Pasztor, M. Musolesi, and C. Mascolo. Opportunistic mobile sensor data collection with scar. In *Proc. of the 4th IEEE Int. Conf. on Mobile Ad-hoc and Sensor Systems (MASS'07)*, Pisa, Italy, October 2007. IEEE Press.
29. G. Ramos-Fernández, J. Mateos, O. Miramontes, G. Cocho, H. Larralde, and B. Ayala-Orozco. Lévy Walk Patterns in the Foraging Movements of Spider Monkeys (Ateles geoffroyi). *Behavioral Ecology and Sociobiology*, 55(3), 2004.
30. G. C. Sanderson. The Study of Mammal Movements: A Review. *The Journal of Wildlife Management*, 30(1), 1966.
31. N. Sarafijanovic-Djukic, M. Pidrkowski, and M. Grossglauser. Island Hopping: Efficient Mobility-Assisted Forwarding in Partitioned Networks. In *Proc. of the Int. Conf. on Sensor and Ad Hoc Communications and Networks (SECON)*, 2006.
32. T. Small and Z. J. Haas. The shared wireless infostation model: a new ad hoc networking paradigm (or where there is a whale, there is a way). In *Proc. of the 4th ACM Int. Symp. on Mobile ad hoc networking & computing(MobiHoc)*, pages 233–244, New York, NY, USA, 2003. ACM.
33. Q. Wang, Y. Zhu, and L. Cheng. Reprogramming wireless sensor networks: challenges and approaches. *IEEE Network*, 20(3), 2006.
34. M. Welsh and G. Mainland. Programming Sensor Networks Using Abstract Regions. In *Proc. of the Symp. on Networked Systems Design and Implementation (NSDI)*, 2004.