MUSTER: Adaptive Energy-Aware Multi-Sink Routing in Wireless Sensor Networks

Luca Mottola and Gian Pietro Picco

Abstract— **Wireless sensor networks (WSNs) are increasingly proposed for applications characterized by many-to-many communication, where *multiple sources* report their data to *multiple sinks*. Unfortunately, mainstream WSN collection protocols are generally designed to account for a single sink and, dually, WSN multicast protocols optimize communication from a single source.**

In this paper we present MUSTER, a routing protocol expressly designed for many-to-many communication. First, we devise an analytical model to compute, in a centralized manner, the optimal solution to the problem of simultaneously routing from multiple sources to multiple sinks. Next, we illustrate heuristics approximating the optimal solution in a distributed setting, and their implementation in MUSTER. To increase network lifetime, MUSTER minimizes the number of nodes involved in many-to-many routing and balances their forwarding load. We evaluate MUSTER in emulation and in a real WSN testbed. Results indicate that our protocol builds near-optimal routing paths, doubles the WSN lifetime, and overall delivers to the user 2.5 times the amount of raw data w.r.t. mainstream protocols. Moreover, MUSTER is intrinsically amenable to in-network aggregation, pushing the improvements up to a 180% increase in lifetime and a 4-time increase in data yield.

Index Terms— **Wireless sensor networks, multi-sink routing, analytical model, distributed protocol, performance evaluation.**

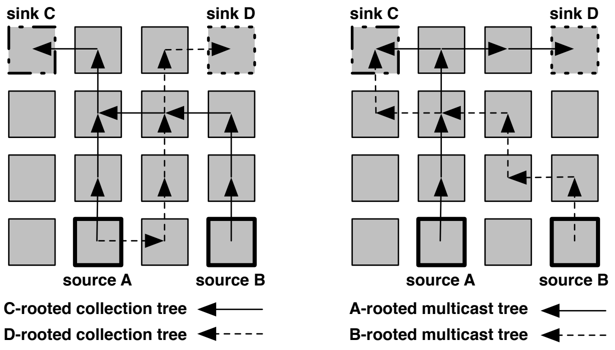
I. INTRODUCTION

Early deployments of wireless sensor networks (WSNs) focused on applications such as habitat monitoring [36], where data is collected at a single sink node for later analysis. Several works in WSN routing address similar many-to-one scenarios [4]. As WSNs are employed in more sophisticated settings, however, applications exhibit different communication patterns.

Application scenarios. WSNs can be used to control multiple actuators dispersed in the environment [3]. In these scenarios, the application requires that data sensed from *multiple sources* is delivered to *multiple sinks*. Consider for instance a decentralized building automation system [16] providing functionality such as heating, ventilation, and air conditioning (HVAC), along with fire alert. The actuator nodes distributed in the environment include air conditioning units, water sprinklers, and fire alarms. Sensor nodes (e.g., for temperature and humidity) are also deployed to feed the control loop. Often, these lie at the intersection of the operating range of different actuators, and are thus required to report to multiple destinations. For instance, the same temperature sensor may report to multiple air conditioners.

Another example is the management of road tunnels. We are part of the TRITon project [24], funded by the local government in Trento (Italy), with the goal to perform adaptive control of the tunnel lighting system. In conventional tunnels, light is often regulated based on few parameters (e.g., date and time of the day) and regardless of the actual environmental conditions. This potentially causes a waste of energy and a safety hazard. In TRITon, a WSN deployed in the tunnel is integrated with light sensors and actuators, adapting light intensity based on the lighting conditions in each tunnel sector. However, light changes in a sector may affect neighboring sectors as well. To enable accurate control, some sensors must report to multiple actuators.

L. Mottola is with the Swedish Institute of Computer Science (SICS), Stockholm, Sweden. E-mail: luca@sics.se. G.P. Picco is with the Department of Information Engineering and Computer Science (DISI), University of Trento, Italy. E-mail: gianpietro.picco@unitn.it.



(a) Two trees rooted at the two sinks are built independently.

(b) Multicast trees for the scenario in Figure 1(a).

Fig. 1. A sample multi-source to multi-sink scenario.

The need for many-to-many communication arises also in support to in-network data processing. For instance, data mining can be efficiently implemented in a distributed fashion by collecting at every node readings from different subsets of sources [45]. Similar communication patterns also emerge when the WSN, instructed by the programmer with dedicated constructs, reports data to multiple aggregation points where some application-specific processing is performed [5], [12]. Finally, many-to-many communication is also germane to scenarios where the same WSN serves multiple applications [30]. These typically run on different sinks gathering data from possibly overlapping subsets of sources.

Problem. Existing WSN routing protocols are ill-suited to the scenarios above, as they focus on a single sink or source. This leads to inefficient communication, reducing the network lifetime.

Data collection protocols typically report data to a *single sink*. The few cases considering multiple sinks address the problem by duplicating the routing infrastructure, and consequently the required resources. For instance, most protocols rely on a sink-rooted routing tree [25], [28], [54], built by the sink by flooding a message that establishes a reverse path from every node to the sink. However, consider the scenario in Figure 1(a). Node A reports data to both sinks C and D, whereas B only reports to C. The mechanism above would build two *independent* trees rooted at C and D. This may lead sources (e.g., A) to duplicate data too early along different trees and may involve in routing more nodes than needed, ultimately reducing the WSN lifetime.

Multicast protocols for WSNs, instead, aim at optimizing the path from a *single source* to multiple sinks. When separate multicast trees are used for many-to-many communication, they are affected by problems similar to collection protocols, as shown in Figure 1(b). Multicast protocols minimize a given metric computed on a per-source basis, e.g., the number of links to reach the target sinks. As sources are not aware of each other, this approach cannot optimize the routing among intermediate nodes. Moreover, aggregation mechanisms lose their effectiveness, precisely because readings from different sources (e.g., A and B) can be combined only very late along their path to the sinks [27].

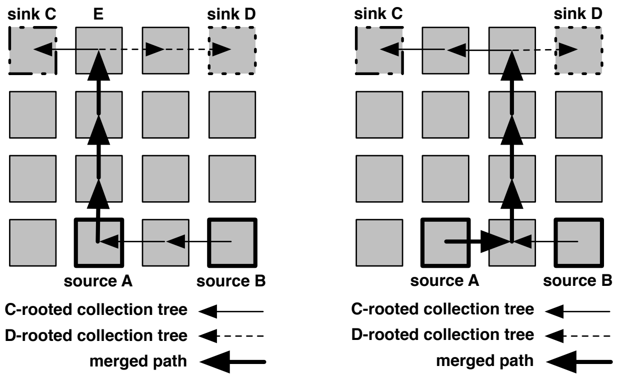
Solution. We overcome the drawbacks of independently-built trees by *reusing* routing paths across multiple trees. This leads to significant improvements when traffic flows simultaneously from different sources to different sinks, as illustrated in Figure 2(a). Unlike Figure 1(a), here the two parallel paths originating at A are merged, and reused to serve the other source B. The paths are split again as late as possible, when the message must inevitably follow distinct routes to reach the two sinks. This scheme reduces the number of nodes involved in routing—in this example, from 13 and 11 in Figure 1(a) and 1(b), to 9 in Figure 2(a). In general, minimizing the number of nodes involved in routing enables:

- a decrease in the amount of redundant information flowing in the network, as data is duplicated only if and when strictly necessary, therefore increasing the *system lifetime*;
- reduced contention on the wireless medium and packet collisions, therefore increasing the *reliability* of transmissions;
- an increase in the beneficial impact of *aggregation*, as readings can be combined much earlier, further reducing the net amount of data being funneled.

MUSTER. Minimizing the number of nodes involved in routing is at the heart of MUSTER (MUlti-Source MUlti-Sink Trees for Energy-efficient Routing), the protocol we present in this paper. MUSTER starts with independently-built trees. As nearby nodes simultaneously funnel traffic, it progressively changes the shape of different trees in a fully decentralized fashion, based on knowledge on paths in the 1-hop neighborhood. This information is compactly encoded and piggybacked on every outgoing message, allowing a node to learn about the availability of better routes and possibly switch parent. Local changes made by a node typically trigger a ripple effect that causes the nodes ahead on the same route to change their parent as well. Nevertheless, in absence of simultaneous traffic in nearby regions of the network, MUSTER still behaves as standard collection protocols.

An undesirable side-effect of this strategy is uneven energy consumption: the nodes along merged paths experience an increased routing load. For instance, in Figure 2(a) the nodes on the vertical "backbone" deplete their energy faster than the other nodes, potentially disrupting the WSN operation. Therefore, in MUSTER we complement the minimization of nodes involved in routing with a scheme to *balance the routing load.* MUSTER "juggles" routes whenever it finds alternative paths extending the system lifetime. For instance, the routing topology of Figure 2(a) may eventually morph into the one in Figure 2(b). The latter configuration involves a different set of nodes, yet their number is the same as in Figure 2(a). As energy is progressively consumed in the configuration of Figure 2(b), MUSTER may decide to return to the topology in Figure 2(a), which meanwhile has saved energy.

Contribution and road-map. In Section II, we formalize our problem using integer linear programming, inspired by the multi-commodity network design problem [55]. This technique assumes



(a) Two paths of the trees in Figure 1(a) are merged.
(b) The merged path "moves" on different nodes, to balance the load.

Fig. 2. A more efficient solution to the routing problem in Figure 1.

global topology knowledge and is therefore impractical for real WSN deployments. However, it yields an optimal topology, useful to compare decentralized solutions against. In Section III we present our protocol. We illustrate the distributed heuristics optimizing routing over multiple sink-rooted trees, as well as our load balancing scheme. MUSTER is simple enough to be implemented on resource-scarce WSN nodes, and provides programmers with hooks for aggregation, as shown in Section IV.

We evaluate the performance of MUSTER using both time-accurate emulation and a real-world testbed. The former, illustrated in Section V, shows that MUSTER enables up to *twice* the lifetime w.r.t. independently-built trees and enables an overall data yield *2.5 times* greater. Moreover, MUSTER amplifies the effectiveness of even a naïve aggregation scheme, by enabling a 180% lifetime increase and a data yield 4 times greater than previous approaches. We also show that the routing topology generated by MUSTER is very close to the one computed with the model in Section II: the number of nodes involved is within 10% of the optimum. These results are confirmed, although on a smaller scale, by experiments in a 40-node WSN deployment, described in Section VI. These show that our load balancing scheme is able to consider variations in the battery discharge induced by temperature changes, an often-overlooked issue that in practice may lead to significant performance degradation.

Finally, Section VII presents a brief survey of related efforts, while Section VIII ends the paper with brief concluding remarks.

II. System Model and Optimal Solution

We formulate the many-to-many routing problem as an integer linear program, later used to compute the optimal topology.

System model. We take inspiration from the multi-commodity network design problem [55], a formulation already applied to throughput and capacity problems in wireless networks [29], [33]. We consider a directed graph (e.g., representing a transportation network) with node set $\mathcal{N}$ and arc set $\mathcal{A}$, and a set of commodities $\mathcal{C}$ (e.g., goods). The goal is to route each commodity $k \in \mathcal{C}$ from a set of origins $O(k) \subseteq \mathcal{N}$ to a set of destinations $D(k) \subseteq \mathcal{N}$ by minimizing a given metric.

We model a WSN as a directed graph where $\mathcal{N}$ is composed of the WSN nodes, and $\mathcal{A}$ is obtained by setting an arc (i, j) between nodes i and j when the latter is within communication range of the former. Without loss of generality, we assume a commodity to flow from a single origin to a single destination [55]. Since commodities flowing from the same origin (source) to the same

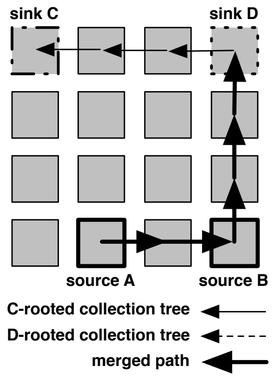
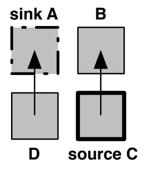


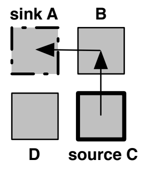
Fig. 3. A routing topology where all transmissions are pair-wise.

Variable	Value
$r^{CA}_{C,B}$	1
$r^{CA}_{D,A}$	1
Remaining $r^{CA}_{i,j}$	0



(a) Node B and D do not obey constraint (4).

Variable	Value
$r^{CA}_{C,B}$	1
$r^{CA}_{B,A}$	1
Remaining $r^{CA}_{i,j}$	0



(b) Constraint (4) holds for every node.

Fig. 4. Sample assignments for $r^k_{i,j}$, and corresponding topologies. We label CA the commodity k flowing from source C to sink A.

destination (sink) follow the same route, we can state a one-to-one mapping between the route connecting any source-sink pair $\langle o(k), d(k)\rangle$, and any commodity k.

We capture message routing with a set of decision variables:

$$r^k_{i,j} = \begin{cases} 1 & \text{if the route for the source-sink} \\ & \text{pair } k \text{ contains arc } (i,j) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

A value assignment $\forall (i,j) \in \mathcal{A}$ to these variables represents the route followed by messages from source $o(k)$ to sink $d(k)$.

Metric. The focus of the multi-commodity network design problem is usually on the number of arcs exploited, i.e., network links in our case. This fails to capture the broadcast nature of the wireless medium. For instance, compare Figure 2(a) and 3. If the goal is to minimize the number of network links used by routing, both solutions are optimal. However, the configuration in Figure 2(a) is preferable in a WSN, since node E can forward data to different receivers with a single broadcast transmission.

This observation leads to the intuition that efficient many-to-many routing can be achieved by reducing the number of *nodes* involved. Since each node along a route is responsible for one transmission, minimizing the number of nodes involved minimizes the total number of transmissions. Therefore, in our model we take the number of nodes (instead of links) participating in routing as the main metric. We capture the fact that node i is involved in *at least one* source-sink route as

$$u_i = \begin{cases} 1 & \text{if } \exists k \in \mathcal{C}, j \in \mathcal{N} \mid r^k_{i,j} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

and define our objective function as

$$\mathit{NodesInvolved}(\mathcal{C}, \mathcal{A}) = \sum_{i \in \mathcal{N}} u_i \tag{3}$$

MUSTER builds upon the relation between u_i and $r^k_{i,j}$ defined in Equation (2). To minimize *NodesInvolved*, we reuse nodes along routes serving other source-sink pairs, that is, nodes for which the cost u_i is already paid. How to achieve this behavior in a distributed setting is the subject of Section III.

Optimal Solution. Our objective is to identify the optimal set of routes to deliver messages from sources to sinks. Formally, we are to find the value assignment of $r^k_{i,j}$, $\forall k \in \mathcal{C}$, $\forall (i,j) \in \mathcal{A}$ such that $\mathit{NodesInvolved}(\mathcal{C}, \mathcal{A})$ is minimum. The optimal solution to this problem can be derived using mathematical programming techniques by specifying proper constraints.

First, we require that $r^k_{i,j}$ and u_i are integer, binary variables and that the following relation holds among them:

$$\forall (i,j) \in \mathcal{A}, \forall k \in \mathcal{C}, \quad r^k_{i,j} \leq u_i$$

The above forbids considering a node as used, unless it is traversed by at least one source-sink path. This constraint is satisfied by construction through Equation (1) and (2).

Second, we state that the assignment to $r^k_{i,j}$ must contain a connected, end-to-end path for each source-sink pair k. This constraint can be expressed by requiring every node different from source $o(k)$ and sink $d(k)$ to "preserve" messages, i.e.:

$$\forall i \in \mathcal{N}, \forall k \in \mathcal{C},$$

$$\sum_{m:(i,m)\in\mathcal{A}} r^k_{i,m} - \sum_{n:(n,i)\in\mathcal{A}} r^k_{n,i} = \begin{cases} 1 & \text{if } i = o(k) \\ -1 & \text{if } i = d(k) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The above imposes the existence of a multi-hop route from source $o(k)$ to its target sink $d(k)$. Figure 4 illustrates the concept. The solution in Figure 4(a) is not acceptable: the message originated at C and directed to A is lost at B and somehow reappears at D. Constraint (4) does not hold for B and D, its left-hand side evaluates to -1 for $i = B$ and to 1 for $i = D$, and neither node is a source or sink. The constraint holds for the solution in Figure 4(b), which represents a connected multi-hop route.

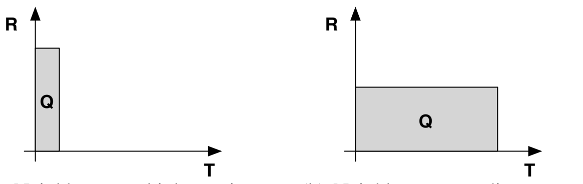
III. THE MUSTER PROTOCOL

The formulation we presented in Section II requires global knowledge of the network topology, impractical in WSNs. In contrast, MUSTER embodies distributed heuristics that minimize the number of nodes involved in routing while balancing their load, and rely only on information available within a node's 1-hop neighborhood. The optimal solution serves as a baseline for comparison against these heuristics, as illustrated in Section V.

A. Overview

MUSTER starts from independently-built trees connecting sources to their sinks. Different subsets of sources may report to different subsets of sinks. Trees are built using the flooding-and-reverse-path scheme described in Section I, and are periodically refreshed to account for node changes and link fluctuations.

The initial trees are mutated over time to optimize the routing topology. A small control header is piggybacked on all messages, received during the periodic tree refresh or overheard as data flows



(a) Neighbor n_1: high routing quality and short lifetime.

(b) Neighbor n_2: medium routing quality and long lifetime.

Fig. 5. Interplay between routing quality and expected lifetime.

towards the sinks. Based on information in the header, each node maintains, for every neighbor n and sink s, a value

$$Q(n,s) = R(n,s) \cdot T(n) \tag{5}$$

denoting the quality of n as a parent towards s. The *routing quality* $R(n,s)$, described in Section III-B, is concerned purely with the optimization of source-sink paths. Instead, $T(n)$, described in Section III-C, is an estimate of the *expected lifetime* of n.

Therefore, $Q(n,s)$ is a measure of how long a neighbor n can provide a given routing quality towards a sink s. This metric yields better configurations than routing quality R alone, as illustrated in Figure 5. A decision based solely on R would privilege neighbor n_1 in Figure 5(a) over n_2 in Figure 5(b). However, the expected lifetime T of n_1 is small. Routing through n_1 may deplete its battery in the near future, possibly disrupting connectivity. Conversely, n_2 has lower routing quality but longer expected lifetime, and is therefore preferable.

The metric Q is used at each node to adapt the source-sink paths by replacing the neighbor n serving as parent towards sink s with the neighbor enjoying maximum quality Q. As the new parent performs routing, its expected lifetime $T(n)$ decreases, along with $Q(n,s)$, and the child node eventually finds another neighbor n' with higher Q for sink s. This scheme "juggles" routes of comparable cost, distributing the routing load among available nodes. Parent switching does not incur additional costs as it is realized with a simple timeout and no extra control messages.

B. Routing Quality

In principle, the routing quality $R(n,s)$ can be defined in terms of various quantities. In this work, we consider the following ones:

- *reliability*(n,s), an indication of how reliable is the end-to-end communication from neighbor n to sink s;
- *paths*(n), the number of source-sink paths passing through a neighbor n, i.e., using the notation in Section II:

$$\mathit{paths}(n) = \sum_{k \in C} r^k_{i,n} \quad (i,n) \in \mathcal{A}$$

- *sinks*(n), the number of sinks n is currently sending data to.

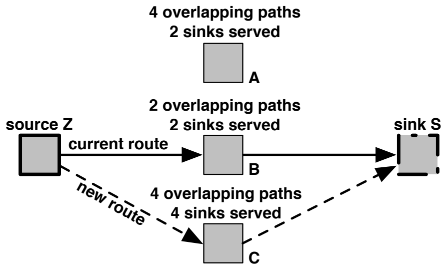
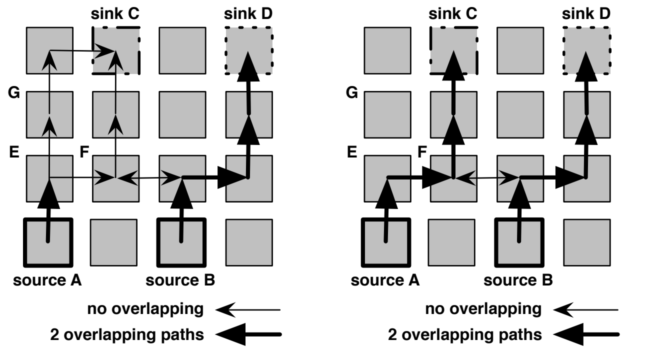


Fig. 6. Source Z generates data for sink S. The current parent of Z towards S is B. However, C is a better choice because it serves the highest number of paths and sinks among Z's neighbors.



(a) Initial configuration.

(b) E switches parent from G to F.

Fig. 7. A sample adaptation process.

Several techniques can be used to compute the *reliability* metric: we discuss our implementation choices in Section IV. Figure 6 shows an intuition for the other two constituents of $R(n,s)$, i.e., *paths* and *sinks*. Node Z has three neighbors A, B, and C. B serves as parent in the tree rooted at sink S, but both A and C are traversed by more source-sink paths than B. If either were selected as Z's new parent, path overlapping would increase. However, C serves more sinks than A, and is thus more likely[1] to be already reporting to S, possibly on behalf of some other source. Therefore, choosing C may enable reuse of a path towards S, further increasing path overlapping at no additional cost.

In this work, we define the routing quality $R(n,s)$ as a linear combination of the three aforementioned quantities:

$$R(n,s) = \delta \cdot \mathit{reliability}(n,s) + \alpha_1 \cdot \mathit{paths}(n) + \alpha_2 \cdot \mathit{sinks}(n) \tag{6}$$

where $\delta, \alpha_1, \alpha_2$ are tuning parameters. The shape of function R and its constituents can in principle be different. Although the results we obtained with this formulation are very positive, we expect that peculiar characteristics of the deployment scenario (e.g., highly fluctuating network topologies) may require adaptations to the expression in (6). Doing so is straightforward in our implementation of MUSTER, described in Section IV, as the definition of R is decoupled from the routing logic.

Figure 7 illustrates a sample adaptation process. We focus on node E and sink C, and assume $\delta = \alpha_1 = \alpha_2 = 1$ in (6). Whenever E has data to send towards C, E evaluates $R(n,C)$ for its two neighbors F and G. The former is traversed by 2 paths and serves 2 sinks, while G is traversed by only 1 path and serves 1 sink. Assuming both neighbors report the same value r for the *reliability* metric towards sink C and the same expected lifetime T, E would compute $Q(G,C) = (r+2) \cdot T$ and $Q(F,C) = (r+4) \cdot T$, thus identifying F as the best next-hop towards C, as shown in Figure 7(b). This change has immediate benefits: the topology in Figure 7(a) exploits 12 nodes, whereas the one in Figure 7(b) only involves 10 nodes.

To break ties between the current parent and a new one, we always select the latter as this will enjoy a higher value of R: its value for *paths* will increase by one. We must however avoid picking one of the current children, as this would create a routing loop. This information can be easily derived from the data messages received within a given time interval.

[1]As we know only the *number* of sinks served by n, we do not know if S is among them. We could propagate the *identifiers* of sinks instead, but their number yields good performance and generates much less overhead.

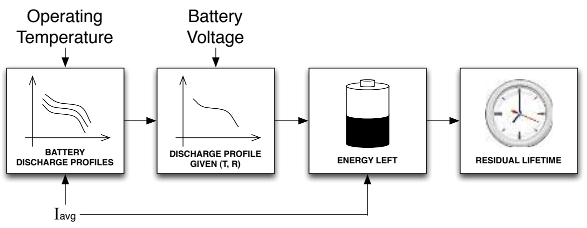


Fig. 8. Computing a node's residual lifetime.

C. Estimating the Expected Lifetime

Estimating the expected lifetime of battery-operated WSN nodes is a challenge per se, as it depends on diverse factors such as network traffic and the non-linear behavior of commercial batteries [40]. The latter is often overlooked and yet deeply affects battery performance and therefore lifetime. The *discharge profile* captures the relation between battery voltage and service hours for varying operating temperatures and current draws. In alkaline batteries, for instance, a drop in the operating temperature from 20^oC to 0^oC easily determines a 50% lifetime decrease [21].

To provide an accurate estimation of a node's residual lifetime, we rely on a lightweight energy model, described in Figure 8, customized to the operation of MUSTER. Based on average current draw I_{avg} and operating temperature, we select a discharge profile among those available for the battery employed. Next, the current battery voltage allows us to identify a point in the profile indicating the energy left in the battery. Dividing this quantity by I_{avg} yields an estimate of the node's residual lifetime.

Discharge profiles are generally available from battery manufacturers. Synthetic models also exist based on the battery physical characteristics [44]. Voltage and temperature readings are usually available from on-board sensors. We can estimate the average current draw I_{avg} by using external hardware devices [19], energy accounting [34], or software-based power profilers [17]. Only a few prototypes of the first exist, none commercially available. Energy accounting requires platform-dependent instrumentation of the entire code to monitor changes in the power level of the MCU. Although this provides very precise measurements, its applicability across different platforms is quite limited.

MUSTER does not require fine-grained lifetime estimation: the *relative* information about whether a node can operate longer than another is enough for the quality metric Q to distribute the routing load evenly. Therefore, we opted for a software-based power profiler based on the following assumptions:

1) radio communication occurs only through MUSTER;
2) MUSTER runs atop a CSMA-like MAC protocol providing some form of low-power listening [41];
3) the current draw due to processing and sensing is roughly the same on all the nodes.

Under these assumptions, I_{avg} can be computed by tracking send and receive operations at the MAC layer, based on the quantities in Figure 9. The energy drain of an operation is obtained by multiplying the corresponding current draw by the duration of the operation. For instance, a sequence $send_{ucast}, send_{bcast}, receive_{ucast}$ leads to an energy drain of:

Symbol	Description	Source
I_{rx}	Current draw when receiving data (mA)	Hardware
I_{tx}	Current draw when transmitting data (mA)	Hardware
I_{idle}	Current draw during low-power listening (mA)	Hardware
b	Radio bit-rate (bits/sec)	Hardware
p_{ucast}, p_{bcast}	Size of unicast/broadcast messages (bits)	MUSTER
t_{ucast}, t_{bcast}	Time for MAC-level handshake (e.g., strobing) in unicast/broadcast transmission (ms)	MAC

Fig. 9. Information used to compute the average current draw.

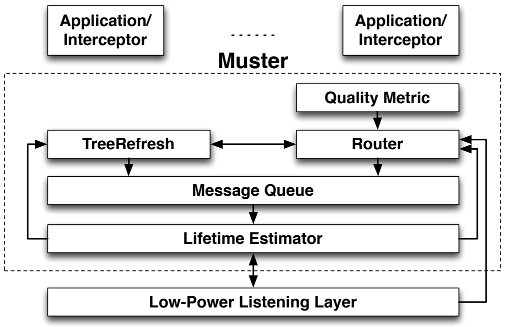


Fig. 10. MUSTER architecture.

$$E = I_{tx}(t_{ucast} + \frac{p_{ucast}}{b} + t_{bcast} + \frac{p_{bcast}}{b}) + I_{rx}\frac{p_{ucast}}{b}$$

In MUSTER, the average current draw is re-evaluated with a period τ, $I_{avg}(\tau) = \frac{E}{\tau}$. Due to routing reconfigurations caused by changes in physical connectivity, it may happen that the radio activity in the i-th time interval τ_i changes from the preceding interval τ_{i-1}. However, these behaviors are generally transient. To smooth out short-term fluctuations, the N most recent $I_{avg}(\tau_i)$ are fed as input to an exponential moving average (EMA), $\overline{I_{avg}}(\tau_i)$:

$$\overline{I_{avg}}(\tau_i) = \alpha I_{avg}(\tau_{i-1}) + (1-\alpha)\overline{I_{avg}}(\tau_{i-1}) \quad (7)$$

EMA is a reasonable trade-off between smoothing effectiveness and reactivity to permanent changes. To account for the limited memory on WSN nodes, we define α in terms of the N stored measurements [11] as $\alpha = \frac{2}{N+1}$. Equation (7) is used both to select a specific battery discharge profile and to estimate the residual node lifetime given the energy available.

We validated our technique by comparing current consumption and lifetime against our estimates at a few sample nodes in the real-world testbed described in Section VI. To measure the former, we used an Agilent 34411A digital multimeter attached to the nodes. Our estimate of current consumption was always within 5% of the value reported by the multimeter, and our lifetime estimate showed a worst-case error of $\pm$9% [43].

IV. IMPLEMENTATION

Figure 10 depicts the architecture of MUSTER, built atop TinyOS 2.0 [48]. Source and sink functionality, as well as in-network processing at intermediate nodes, interact with MUSTER through the `Collection` interfaces [51], while network communication relies on the Low-Power-Listening (LPL) layer [50]. Our implementation is compact. The state information we store on a node amounts to 8 B for every neighbor, 4 B for every sink, and 5 B for every source-sink path traversing the node. In the configuration we use for the evaluation described later, MUSTER occupies a total of about 2 KB of data memory and 8 KB of program space. MUSTER is publicly available as open source [1].

Interfaces and Interceptors. The `Collection` interfaces are designed for many-to-one communication. We modified them to add the identifiers of target sinks as parameters of the `send` command [51]. Intermediate nodes process in-transit packets with the `Intercept` interface, which contains a single event:

```
event bool forward(message_t* msg,void* payload,uint8_t len);
```

MUSTER signals this event upon receiving a packet to be forwarded. The higher layers may decide to forward immediately by returning `TRUE`, or to perform some processing and send a possibly different packet later. In-network processing schemes are therefore easily integrated. We implemented two examples:

- A *packing* scheme to include multiple payloads in the same packet. A packet received is not forwarded immediately: its payload is inserted in a buffer associated to the neighbor the packet is addressed to. When the buffer is full, or upon expiration of a timeout, a new packet containing the entire buffer content is created and sent.
- An *aggregation* operator to average sensor readings. Each node keeps track of the sources funneling data through it, and computes their time drift based on a timestamp embedded in the payload. This allows each node to compute the average of readings at different sources within the same epoch, which is returned to MUSTER instead of the original reading.

TreeRefresh and Router. The `TreeRefresh` module coordinates the periodic refresh of topology information. Each sink periodically floods a "tree construction" message. Upon reception, every node computes the end-to-end $reliability$ metric. On IEEE 802.15.4 radios we rely on a metric similar to that in MultihopLQI [49]. This is based on the Link Quality Indicator (LQI) provided by radio chips such as the ChipCon CC2420, which equips the widely-used TMote Sky [42] nodes. In absence of LQI, we use the inverse of the hop-count to a sink, a metric potentially inaccurate, yet used successfully in real deployments [6]. Since this functionality is decoupled from the rest of MUSTER, alternative metrics (e.g., ETX [13]) can be easily integrated.

The `Router` module determines the parent (i.e., neighbor with maximum Q) based on the data structure shown in Figure 11 for a given neighbor. The value of $neighborId$ serves as index. The value of $reliability$ is retrieved from the `TreeRefresh` module. The values of $paths$, $sinks$, and $lifetime$ are piggybacked on incoming messages. The `Router` module also performs packet transmissions. These occur in unicast if the packet is addressed to a single parent, or in broadcast if it is addressed to multiple next hops, e.g., when previously merged paths split. In the latter case, the packet includes the list of neighbors that are to process the message. A packet is retransmitted if the LPL layer notifies that the receiver did not acknowledge the message. If the maximum number of retransmissions is reached (e.g., because the destination died) the `Router` module defaults to the neighbor n' with second maximum Q. This procedure repeats until all candidate next hops are examined, and the message is dropped.

Lifetime Estimator. This module implements the model in Section III-C by intercepting incoming/outgoing messages. It also stores the required discharge profiles, based on the scenario and batteries employed. We implemented a simple pre-processor that converts Comma Separated Values (CSV) files—the data format normally used by battery manufacturers—into static look-up tables of constant values. This allows most C compilers to store these data structures in the code memory instead of data memory, the latter being generally more precious on WSN nodes.

Field Name	Description
neighborId	The identifier of the neighbor relative to this entry.
reliability	An associative array containing, for each sink in the system, the corresponding reliability metric when using *neighborId* as parent.
paths	The number of different source-sink paths currently passing through *neighborId*.
sinks	The number of sinks served through *neighborId*, possibly along a multi-hop path.
lifetime	The expected lifetime of *neighborId*.

Fig. 11. Information to compute the quality metric $Q(n, s)$ for a neighbor.

V. Simulation Experiments

We evaluate the performance of the MUSTER implementation described in Section IV using Avrora [52]. The latter allows for fine-grained emulation of the popular MICA2 platform [14], and includes a detailed model to evaluate its energy consumption.

Our evaluation is divided in three parts. In Section V-A, we assess whether the behavior of MUSTER matches our design criteria, using a synthetic scenario. In Section V-B we evaluate the performance of MUSTER against a *base protocol* that only optimizes the $reliability$ metric on a per-sink basis, and is therefore representative of protocols that build independent trees [25], [28], [54]. We also examine MUSTER's performance compared to the optimal routing topology we identified in Section II. In these scenarios, all protocols we test employ the multiple-payload packing scheme described in Section IV. We investigate the impact of aggregation in Section V-C by running all protocols with the average operator, also described in Section IV.

General settings. Nodes are initially provided with an energy budget equivalent to a pair of commercial AA batteries. As it is not possible to emulate LQI readings in Avrora, in both MUSTER and the base protocol we consider the inverse of the hop-count to a sink as reliability metric. We configure Avrora to use two-ray ground propagation to model wireless transmissions.

We configure LPL with a wake-up period of 1 s. We use a 32-bit integer value to represent a single sensor reading, and use the maximum message size: considering protocol and application control overhead, at most 10 readings fit in a message. The time interval separating two messages from the same source (the *epoch*) is set to 60 s. In both protocols, tree refresh is triggered every 5 min, and the transmission of data packets is retried at most 5 times. As for the packing scheme, we set to 5 s the timeout after which a (possibly partially-filled) message is sent out.

We set $\alpha_1 = \alpha_2 = 1$ and $\delta = 2$ in Equation (5). The combined contribution of $paths$ and $sinks$ is the most effective in reducing the number of nodes involved in routing. Moreover, the $reliability$ metric is key to ensuring message delivery, and is thus given higher relative importance. Nevertheless, we also analyze the influence of different combinations of parameter values.

All experiments are repeated 50 times, and the results averaged.

A. Analyzing the Protocol Behavior

To carry out a fine-grained analysis of MUSTER's behavior, we run experiments by tracing over time the remaining energy at every node, and use a custom-built visualization tool to generate "energy maps" representing the system evolution over time. We define five classes of nodes based on their remaining energy, as shown in Figure 12. To better interpret the results, we use a synthetic scenario with only two sources and two sinks, placed at the opposite corners of a grid topology where non-border nodes can communicate with four neighbors. Because of this peculiar setting, the results obtained are not representative of MUSTER's performance, which is analyzed further in Section V-B and V-C.

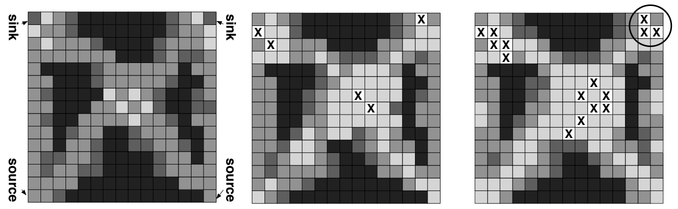
Results. Figure 13 depicts different snapshots of the WSN running the base protocol. Two source-sink paths cross in the center of the grid. Nodes in that area are exploited for routing towards two sinks, and deplete their energy faster than others. The nodes closest to the two sinks are similarly exploited, as they lie where two paths leading to the same sink converge. Consequently, nodes eventually start failing in the middle of the grid and around the two sinks, until one sink is completely disconnected, as shown in Figure 13(c). Although the pictures show the case with 225 nodes, we observed the same behavior with different system scales.

Figure 14 illustrates the behavior of MUSTER's path merging. Load balancing is disabled by setting $T(n) = 1$ in Equation (5). It is evident how the overlapping of source-sink paths leads to the formation of a "backbone" in the middle of the network. Nodes along these merged paths consume energy faster by serving multiple source-sink pairs. The generation of the backbone, however, improves the overall performance by delaying the moment when a sink is disconnected from the network, from 13109 epochs in Figure 13(c) to 15994 in Figure 14(c). The cause for disconnection is the same as with the base protocol: the nodes around the sink are the most stressed when a routing tree is

■ **100%-75% energy left** ▨ **50%-25% energy left** ☒ **dead node**

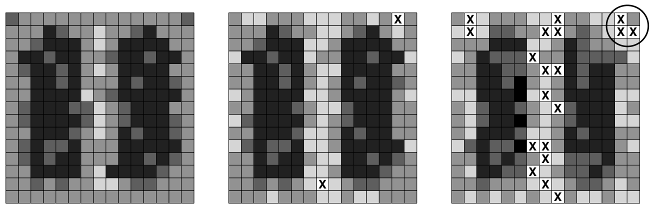
▩ **75%-50% energy left** □ **25%-0% energy left**

Fig. 12. Color codes for nodes in Figure 13-15, denoting remaining energy.



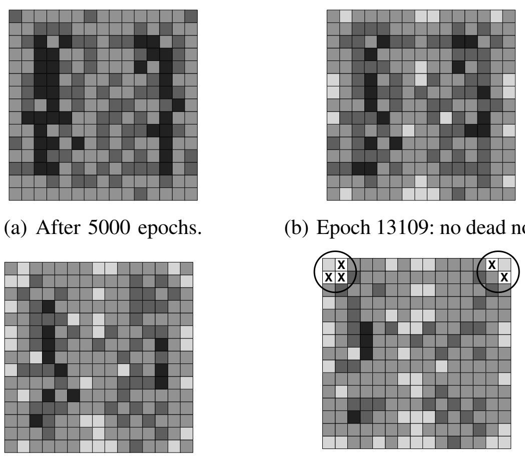
(a) After 5000 epochs. (b) After 12500 epochs. (c) Epoch 13109: one sink is disconnected.

Fig. 13. Energy consumption with the base protocol (225 nodes).



(a) After 5000 epochs. (b) Epoch 13109: system still running. (c) Epoch 15994: one sink is disconnected.

Fig. 14. Energy consumption using MUSTER *without* load balancing.



(a) After 5000 epochs. (b) Epoch 13109: no dead node.

(c) Epoch 15994: no dead node. (d) Epoch 17002: both sinks are disconnected.

Fig. 15. Energy consumption using MUSTER *with* load balancing.

used [54]. However, we verify that MUSTER causes some amount of path merging to occur also among nodes near the sinks, when paths come close enough. This reduces the number of packets these nodes must send, increasing their lifetime.

Figure 15 depicts the dynamics of our complete protocol, taking into account both routing quality and expected lifetime. A comparison of these snapshots against Figure 13 and 14 provides an immediate, visual indication of the effectiveness of our load balancing scheme. Indeed, the "backbone" effect is much less evident. Moreover, the number of nodes consuming at least 25% or 50% of their available energy increases over time, while in the previous cases there are nodes left with 100% of their energy. These phenomena are due to the ability of MUSTER to distribute the routing effort evenly. Specifically, Figure 15(a) shows that after 5000 epochs no node is yet under 25% of remaining energy. After 13109 epochs all nodes are still alive, as shown in Figure 15(b), whereas at the same epoch in Figure 13(c) the base protocol already caused a sink to disconnect. Similar considerations holds for Figure 15(c) and Figure 14(c). In the latter, MUSTER without load balancing experiences a network partition. Instead, at the same epoch all sinks are still connected when using the full protocol. The latter eventually experiences a partition due to some nodes dying around one of the sink, as shown in Figure 15(d). However, this happens far later in time, as our solution is able to merge the paths around the sinks and alternate among these critical nodes. At the time of the first partition almost all nodes spent at least 50% of their energy.

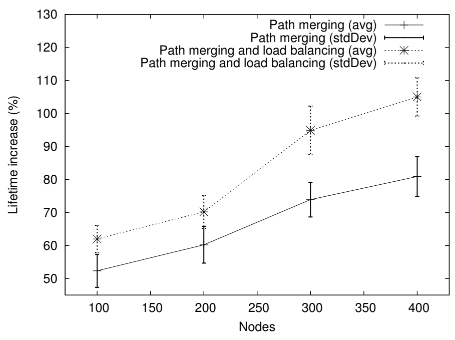
B. Performance Characterization

We compare the performance of MUSTER against the base protocol and against multiple optimal sink- and source-rooted trees. These are computed, using traditional graph algorithms, by minimizing the number of links exploited to connect a sink (source) to its sources (sinks). Moreover, we compare MUSTER against the optimal solution, computed by using GLPK [2] to solve the integer linear program we specified in Section II.

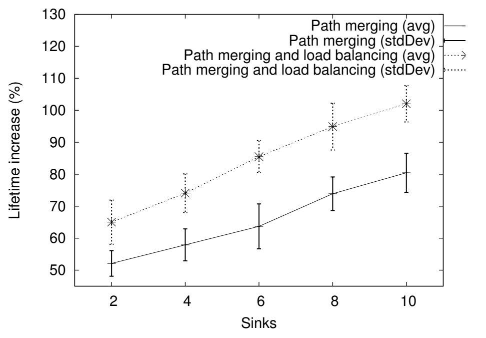
The performance of routing protocols is generally affected by topology, especially when peculiar configurations (e.g., "lines" of nodes connecting two partitions or "holes" without nodes) are present. However, it is impractical to cover all possible cases. Therefore, we use randomly-generated topologies with a pre-specified *average* number of neighbors per node, as a compromise between generality and control over the topology. In each scenario, 10% of the nodes are sources. We vary the number of sinks and nodes to study how MUSTER handles different numbers of source-sink paths. The location of sources and sinks is decided randomly, but a node cannot simultaneously operate as source and sink. These settings are inspired by real deployments [6], [7], [36], [38]. As performance metrics, we measure:

- the *system lifetime*, defined as the moment when the *last* source-sink path becomes interrupted and the experiment ends, which coincides with the point in time when the WSN becomes unusable [31];
- the *ratio of readings delivered* to the sinks over those sent, to investigate the amount of data that users successfully receive;
- the *end-to-end latency*, i.e., the time taken by a sensor reading to reach the target sink.

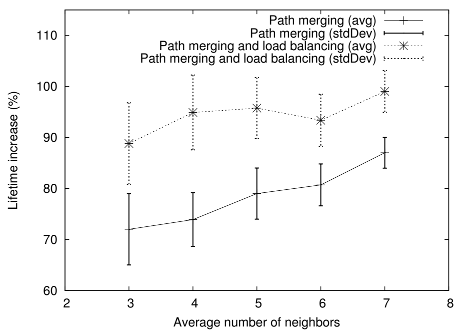
The first two metrics are commonly used to evaluate WSN routing protocols [4]. Both directly affect the *data yield*, that is, the



(a) Lifetime increase vs. number of nodes (8 sinks, 4 neighbors).



(b) Lifetime increase vs. number of sinks (300 nodes, 4 neighbors).



(c) Lifetime increase vs. average number of neighbors (300 nodes, 8 sinks).

Fig. 16. Lifetime increase enabled by MUSTER.

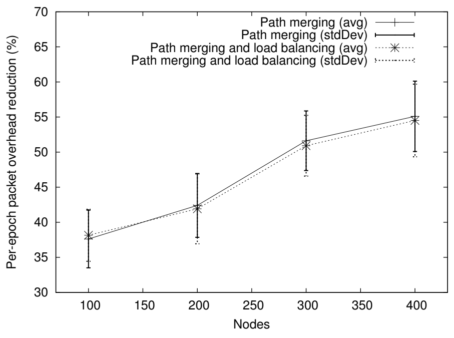
amount of data gathered by the WSN during its operation—the metric domain experts are ultimately interested in. To obtain a better understanding of MUSTER's operation, we also measure:

- the *number of active source-sink paths* over time, to understand how the system performance degrades when nodes start failing because of energy depletion;
- the *number of nodes exploited*, i.e., the metric we aim to minimize to obtain more efficient source-sink routes;
- a node's *remaining energy* at the end of the experiment, to study the effectiveness of the load balancing scheme;
- the *average length* of source-sink paths, in number of hops, to separate out the latency caused by longer routes from the one due to the packing scheme;
- the *ratio of packets delivered* to the sinks, to evaluate directly the impact of our techniques at the network level.

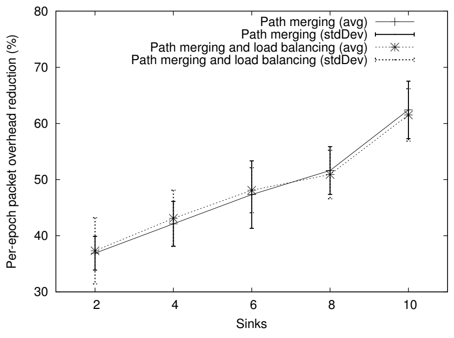
Results. MUSTER improves drastically the system lifetime compared to the base protocol, as illustrated in Figure 16. Specifically, Figure 16(a) depicts the additional lifetime allowed by MUSTER against the system scale. The path merging mechanism alone increases lifetime from about 50% to 80%, with larger improvements as the system scale increases. In combination with load balancing, MUSTER allows for more than *twice* the lifetime provided by the base protocol. Load balancing bears a greater impact as the system scale increases. Indeed, more nodes correspond to more resources available, providing our load balancing scheme with a higher total energy budget to exploit.

Similar trends can be observed in Figure 16(b), where we analyze the lifetime increase enabled by MUSTER by varying the number of sinks. Comparing this chart with Figure 16(a) shows how performance is ultimately dictated by the number of source-sink paths, rather than by the number of sources or sinks alone. For instance, the additional lifetime in Figure 16(b) with 2 sinks and 300 nodes (i.e., 60 source-sink paths) is close to the one in Figure 16(a) for 100 nodes and 8 sinks (i.e., 80 source-sink paths). The similar performance obtained in different settings indirectly confirms that the improvements are due to MUSTER's ability to overlap different source-sink paths, regardless of the combination of sources and sinks that form them.

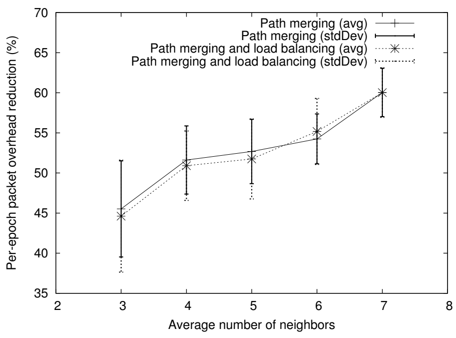
Figure 16(c) analyzes the impact of network density on system lifetime. In this case, the contribution of path merging increases



(a) Per-epoch packet overhead reduction vs. number of nodes (8 sinks, 4 neighbors).

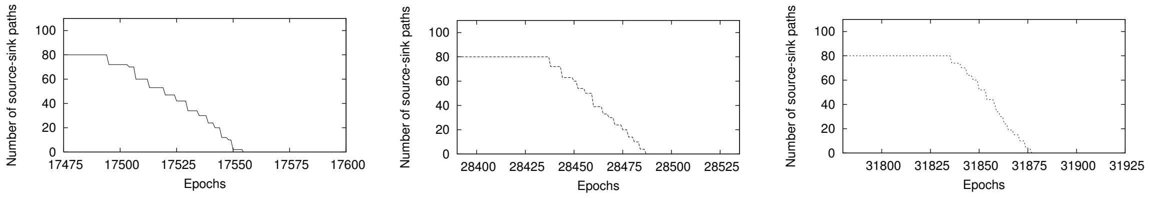


(b) Per-epoch packet overhead reduction vs. number of sinks (300 nodes, 4 neighbors).



(c) Per-epoch packet overhead reduction vs. average number of neighbors (300 nodes, 8 sinks).

Fig. 17. Per-epoch packet overhead reduction.



(a) Base. (b) Path merging only. (c) Path merging and load balancing.

Fig. 18. Number of active source-sink paths over time (100 nodes, 8 sinks).

with network density, while the dispersion of the measures we obtained around the average value decreases. More neighbors indeed correspond to more choices when selecting a parent. By inspecting our simulation logs, we verify that as network density increases, the path merging mechanism alone is sufficient to have near optimal routes during the early part of the system lifetime. The load balancing scheme, instead, begins influencing route selection when the energy left on the nodes is below 50%.

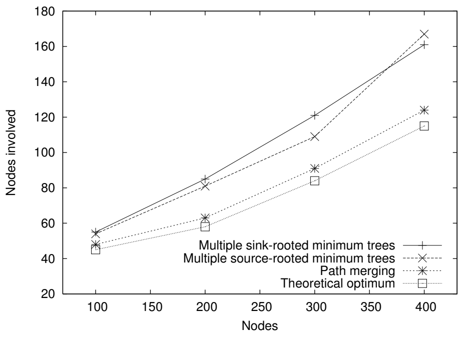
The increased lifetime is enabled mainly by improvements in transmission efficiency. Throughout all experiments, MUSTER's reduced contention on the wireless medium yields, on average, about 50% less packet retransmissions w.r.t. the base protocol. Figure 17 shows the reduction in the packet overhead (i.e., overall decrease in number of transmissions at the physical layer), computed on a per-epoch basis. The trends mirror those in Figure 16, demonstrating that the performance gains enabled by MUSTER come from reduced transmission costs. As observed earlier, these are more marked as the number of source-sink paths increases or more neighbors are available when selecting a parent. In these charts, the effect of load balancing is instead negligible, given that they show the performance within a single epoch.

To investigate how the system behaves during the additional running time allowed by MUSTER, Figure 18 shows the number of active source-sink paths over time close to the end of the experiment. Regardless of the solution employed and the system scale, this metric always decreases abruptly, as soon as the death of some node around a sink prevents communication towards the rest of the system. However, our scheme pushes much farther the moment in time when this occurs, as can be noted by comparing the values on the x-axis across the three charts in Figure 18. Therefore, during the extra time allowed by MUSTER w.r.t. the base protocol, the WSN effectively operates to its full capabilities.

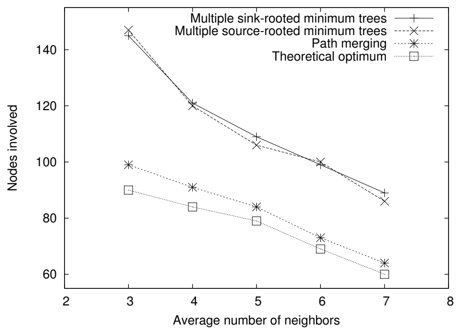
In all experiments, MUSTER delivers to the sinks roughly the same number of packets as the base protocol. However, these packets carry more application data, as multiple readings from merged paths are packed in the same physical packet. In each epoch, the ratio of readings delivered to the sinks increases of about 20% w.r.t. the base solution, mostly irrespective of load balancing. This, combined with the increased lifetime, determines a significant increase in the overall *data yield*, which in MUSTER becomes about *2.5 times* the one of the base protocol.

The results above are enabled by the combination of path merging and load balancing. To study the effectiveness of the former, we measure the average number of nodes involved in routing using MUSTER without load balancing, compared to multiple sink- and source-rooted minimum trees, as well as the optimal solution based on the model in Section II. Except for those concerning MUSTER, all results are obtained in a centralized manner and with global knowledge of the system topology.

As Figure 19 illustrates, in the network configurations we experimented with, MUSTER is always within 10% of the theoretical minimum number of nodes to connect sources to sinks, yet our protocol does *not* require any a priori knowledge of the system topology. These results hold both against a variable number of nodes in the system (Figure 19(a)), and w.r.t. varying network density (Figure 19(b)). In the latter, the gap from the theoretical minimum reduces as the network becomes more connected because, as already observed, MUSTER enjoys more options when selecting parents, and thus has more chances to approximate the theoretical optimum. The same charts also demonstrate that MUSTER reduces the nodes involved in routing compared to multiple sink- and source-rooted minimum trees. In the latter cases, the routing topology is naturally biased towards the sinks



(a) Nodes involved vs. system size (4 sinks, 4 neighbors).



(b) Nodes involved vs. average number of neighbors (300 nodes, 4 sinks).

Fig. 19. Nodes involved in routing.

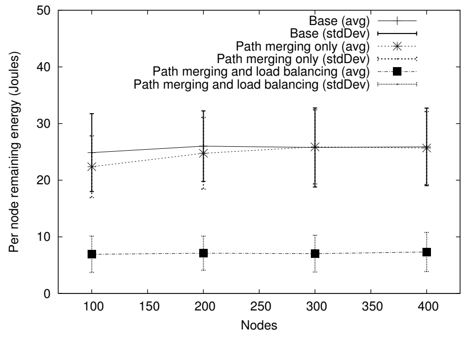


Fig. 20. Per-node remaining energy at the end of the experiment.

(sources) and parallel paths are not necessarily factored out.

To assess the contribution of our load balancing scheme, Figure 20 illustrates the energy remaining at each node at the end of an experiment, against the system scale. Using path merging alone, this quantity is almost the same as in the base protocol. In contrast, with load balancing this figure becomes much lower, and the variance of the results also decreases. This confirms that the contribution to system lifetime brought by this mechanism comes from spreading the routing load evenly, so that more nodes eventually participate in routing.

Figure 21 shows that MUSTER has higher delivery latency w.r.t. the base protocol, as expected. The absolute values at stake are, however, within the tolerance of popular WSN applications. For instance, in environmental monitoring [6] the time constants of the monitored phenomena are usually in the order of tens of minutes. As an example of stricter requirements in closed-loop control, in the adaptive tunnel lighting we are developing, mentioned in Section I, the reporting period for light samples is between 30 s and 5 min. Therefore, reporting sensed data within tens of seconds is still acceptable. Nevertheless, applications requiring real-time delivery should leverage different techniques [57].

To further investigate latency, we run experiments by disabling the packet merging scheme in MUSTER only. The results, also shown in Figure 21, reveal that MUSTER without packet merging performs almost like the base protocol, which is instead running *with* packet merging. In the latter protocol, packet merging has limited impact because paths rarely overlap, and thus almost never split towards different sinks. Therefore, a packet is frequently filled up close to the source: packet merging rarely intervenes—there is no room to pack more data—and the packet is always immediately forwarded, travelling *unaltered* up to the sink.

On the other hand, MUSTER boosts the effect of the packet merging scheme. The overlapping paths eventually split towards different sinks. In this case, the packet is also split, with parts of the payload forwarded in different directions. There is now room to pack more data in the resulting packets, and so the nodes wait for more data before forwarding further. This causes the increase in latency for MUSTER: the price we pay for increased lifetime and reliability. Nevertheless, programmers can trade latency for lifetime or reliability by modifying the packing scheme or setting a smaller timeout.

To gain a deeper insight into MUSTER's operation, we analyze the time required to converge to a stable configuration at system start-up. Based on a sample execution, Figure 22 depicts the average number of parent changes at all nodes against the epoch number. In the largest configuration we tested, it takes at most 12 epochs to stabilize the routes. This is essentially because our EMA-based lifetime estimator needs to accumulate enough samples before stabilizing, causing changes that ultimately affect the entire network. After this initial phase, however, routes tend to remain stable until energy begins to drop significantly at some nodes and the load balancing scheme intervenes.

As the operation of MUSTER can be controlled by the parameters $\delta, \alpha_1, \alpha_2$ defined in Section III-B, we analyze their impact on performance. The trends in Figure 23(a) demonstrate that α_1

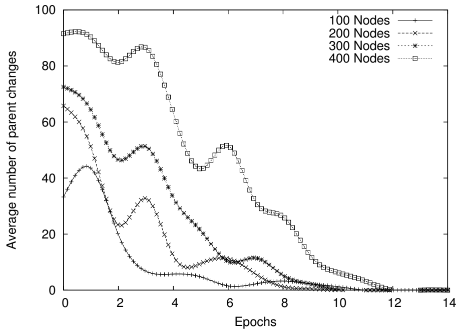
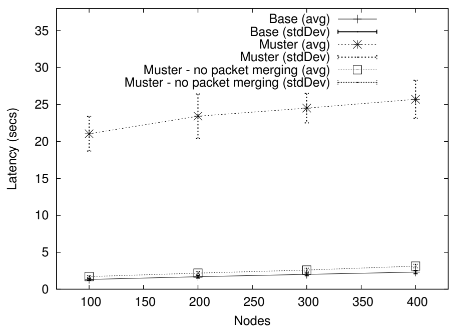
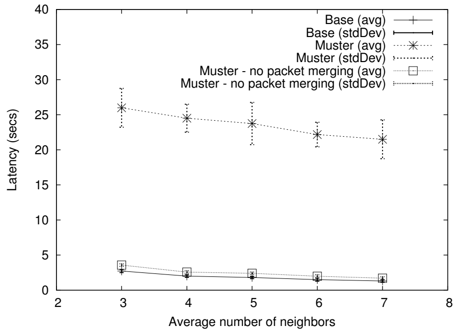


Fig. 22. Convergence time at system start-up (8 sinks, 4 neighbors).

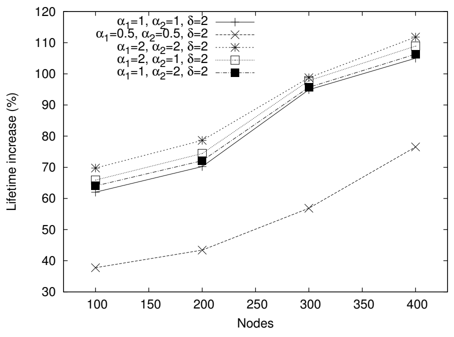


(a) Latency against system size (4 sinks, 4 neighbors).

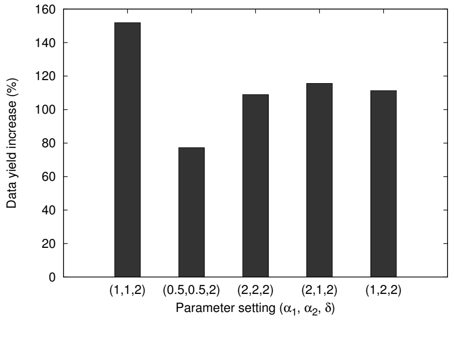


(b) Latency against average number of neighbors (300 nodes, 4 sinks).

Fig. 21. Average end-to-end latency.



(a) Lifetime increase vs. number of nodes (8 sinks, 4 neighbors).



(b) Data yield increase vs. parameter setting (300 nodes, 8 sinks, 4 neighbors).

Fig. 23. MUSTER performance with varying parameters.

and α_2 are key to increase lifetime. The less weight they have, the more routes degenerate in multiple *non*-overlapping trees, approaching the base protocol. On the other hand, increasing their importance w.r.t. δ beyond a certain threshold does not enable further improvements. Reliability also suffers in these configurations, as shown in Figure 23(b). By inspecting the simulations logs we verified that when δ has less influence routes tend to stretch too much w.r.t. the shortest path, and the probability to lose a packet increases. These results confirm that the configuration we used throughout the paper is the best trade-off among those tested.

In the experiments hitherto discussed, lifetime is determined by the nodes around the sinks. An alternative is to make the network denser around sinks, to compensate for the higher load [54]. To investigate how MUSTER behaves in this scenario, we run a set of experiments where node location is decided semi-randomly, by partially controlling the density of nodes around a sink. We divide the physical space in square sub-areas with a 200 m side. In each sub-area A, we deploy a set of nodes $N(A)$ such that:

$$|N(A)| = \sum_{s \in S} \frac{K}{distance(center(A), s)^2} \tag{8}$$

where S is the set of sinks used in the experiment, $distance(center(A), s)$ returns the physical distance between the center of A and sink s, and K is a constant large enough to yield a connected topology. Intuitively, (8) deploys more nodes around the sinks, and decreases their density away from them.

This scenario amplifies the improvements of MUSTER w.r.t. the base protocol, as shown by comparing Figure 24 against Figure 16(a). The more regular topology yields a smaller variance but the gains due to load balancing are larger, because MUSTER can juggle among the many nodes around sinks and further delay their disconnection. This occurs without protocol modifications, as MUSTER automatically adapts to the given topology.

C. Impact of Aggregation

The path merging in MUSTER causes data from different sources travel together as early as possible. This amplifies the beneficial effect of aggregation, further reducing the amount of data flowing in the network. To quantify this aspect, we re-run the experiments discussed so far by employing the average operator described in Section IV in both MUSTER and the base solution.

Results. The trends obtained using aggregation are essentially the same as in Section V-B, as shown by comparing Figure 16 and 25. The absolute values, however, change in favor of MUSTER. The use of even a naïve aggregation operator like ours boosts the improvements of MUSTER over the base protocol up to 180%.

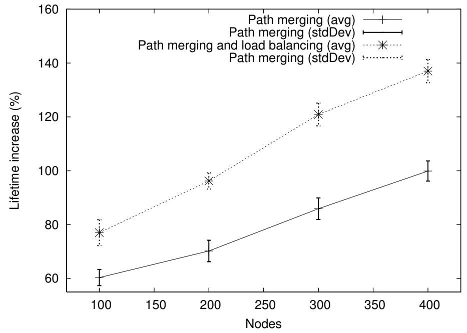
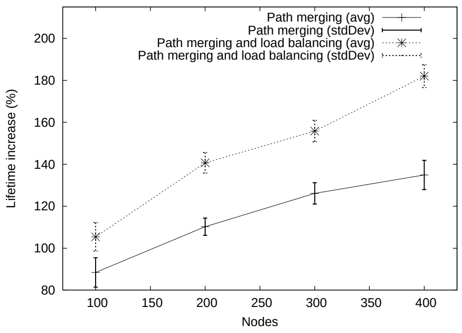
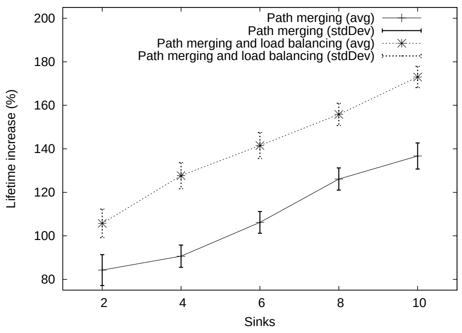


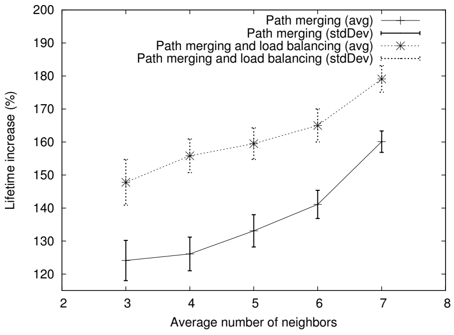
Fig. 24. Semi-random topology: lifetime increase vs. number of nodes (8 sinks).



(a) Lifetime increase vs. number of nodes (8 sinks, 4 neighbors).



(b) Lifetime increase vs. number of sinks (300 nodes, 4 neighbors).



(c) Lifetime increase vs. average number of neighbors (300 nodes, 8 sinks)

Fig. 25. Lifetime increase enabled by MUSTER when data aggregation is used both in MUSTER and the base protocol.

Path merging is mostly responsible for this improvement, as the relative contribution of load balancing in Figure 16 and 25 is comparable. Moreover, packet delivery increases by 15% using MUSTER, and the ratio of (aggregated) readings delivered to sinks now improves of about 45% over the base solution. Again, this is mainly due to path merging, which lets nodes aggregate data closer to the sources w.r.t. the base solution. This corresponds to less data being funneled through intermediate nodes, and hence fewer contention on the wireless medium and reduced packet collisions. As for data yield, MUSTER provides the final users with *4 times* the amount of raw data gathered by the base protocol.

As expected, the other metrics we examined in Section V-B are not affected by aggregation. In particular, the average end-to-end latency is comparable since—at least in the case of the average operator—data generated by the same source in different epochs contribute to different averages, and the latency we previously observed was relative to a single data epoch.

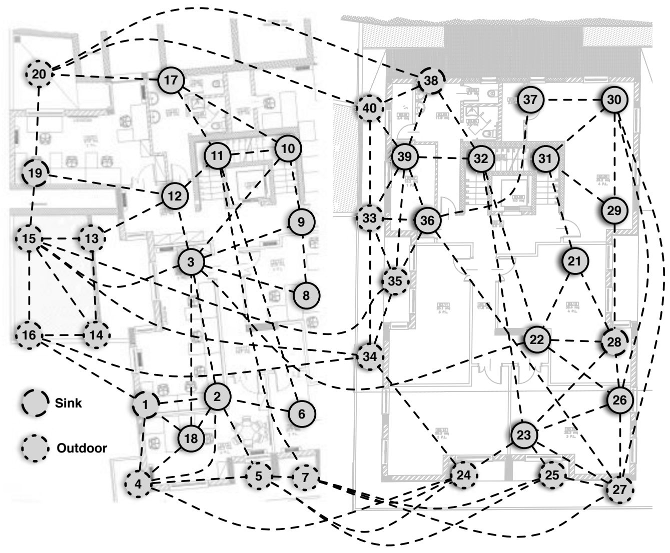


Fig. 26. Testbed deployment. (Dashed lines represent communication links active for at least 80% of the duration of all experiments).

VI. Real-World Experiments

The goal of this section is to confirm in a real setting the results described in Section V, and to assess the effectiveness of load balancing when using real battery discharge profiles in the presence of temperature changes.

We use 40 TMote Sky nodes deployed in two adjacent office floors, as shown in Figure 26, running the IEEE 802.15.4-specific implementation described in Section IV. Some nodes are placed *outdoor*, therefore directly subject to temperature changes. The sinks, whose location is fixed, are hooked via USB to 4 GumStix embedded PCs (`www.gumstix.com`) to enable remote control and collection of the experiments' results. The USB connection also powers the sink nodes. All other nodes are powered using a single Duracell CR2016 battery, for which we use discharge profiles at 20^oC, 25^oC, 30^oC, and 35^oC [18]. These batteries have about 4% of the capacity of two AA batteries, which allows us to run multiple repetitions of the experiments in reasonable time[2]. Ten nodes are randomly chosen as sources at the beginning of every experiment. All other settings are as in Section V.

We compute a subset of the metrics in Section V-B. We verified that a temperature drop may cause a node transient failure even with leftover energy. The node may become available again if the temperature raises. Therefore, we declare an experiment over when the last source-sink path is interrupted for at least 30 consecutive epochs. To factor out fluctuations of wireless links and compute the optimal routing topologies with the model in Section II, we consider a link between two nodes when they are listed in each other's neighbor set for at least 80% of the experiment duration. We cannot measure the exact end-to-end latency, as this would require time-synchronizing the nodes, creating further network traffic that may affect the experiments. Moreover, considered the small capacity of the batteries employed, it is very difficult to measure directly the energy left at the end of the experiment—as we did in Figure 20—without sophisticated tools. Therefore, we report instead the number of nodes still running at the end of experiment, as an indirect measure of the energy left.

The results below are averages over 15 repetitions carried out

[2]A TMote Sky with LPL runs for up to 1 month on 2 standard AA batteries.

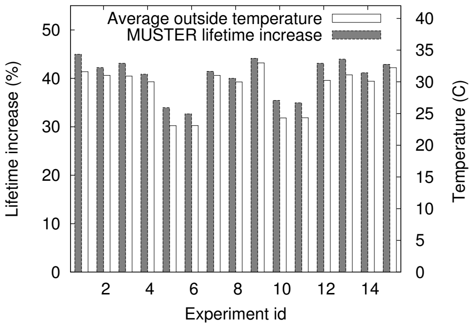


Fig. 27. Lifetime and average outdoor temperature.

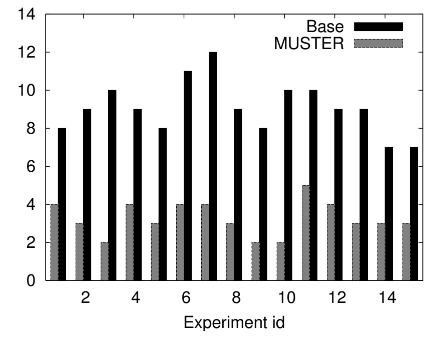


Fig. 28. Nodes still running at the end of the experiments.

by running Muster first, and then the base protocol with the same source nodes, for a total of about 35 days of experiments.

Results. Figure 27 illustrates the lifetime improvement enabled by Muster in our testbed. The simulation results in Section V are confirmed, although absolute values are smaller here because of the fewer source-sink paths. The improvement is consistent across all experiments, despite the different placement of the sources. These results are again mainly due to the improved transmission efficiency enabled by Muster that, throughout the experiments, reduces the packet overhead of about 40% w.r.t. the base protocol.

Figure 27 also shows the average temperature sensed by outdoor nodes during each experiment. Interestingly, higher outdoor temperatures correspond to higher improvements in Muster, whereas we observe no clear correspondence between the base protocol performance and outdoor temperature. Although the batteries we used provide more service hours at higher temperatures, the base protocol is oblivious to such behavior: outdoor nodes may even be left unused. Instead, Muster's leverages this information through battery discharge profiles, balancing the routing load and thus pushing farther in time the moment where the network becomes permanently partitioned.

The reasoning above is confirmed in Figure 28, where we plot the number of nodes still running when an experiment ends. With the base protocol about 23% of the nodes, on average, are still running when the WSN becomes partitioned. This ratio drops to about 7% with Muster, confirming the effectiveness of its load balancing. Furthermore, although we cannot precisely measure the energy left, our logs show that the base protocol almost *never* uses some of these nodes because, unlike Muster, it is unable to recognize that exploiting them may extend the network lifetime.

Figure 29 evaluates Muster's path merging in our testbed experiments, showing the number of nodes it involves in routing w.r.t. the theoretical minimum and multiple sink- and source-rooted minimum trees, similarly to the analysis in Section V-B. Because of the smaller scale of our testbed, the figures are fairly close to each other. Nevertheless, MUSTER constantly exploits fewer nodes than sink-(source-) rooted trees, confirming that, even in a relatively small network, our path merging technique provides significant benefits over a blind replication of routing trees.

VII. Related Work

Routing in WSNs has been studied extensively [4]. In the following, we survey the state of the art focusing on solutions closest to our scenarios, goals, and approach.

Many-to-many communication. Yuen et al. [56] present a multi-sink collection protocol where sources gather correlated data, and adjust the sensing rate at different nodes to eliminate redundant information. On the contrary, we do not make assumptions on the nature of data at different sources. Yuen et al. also leverage integer linear programming to design a distributed algorithm. They do not report, however, the performance in terms of processing overhead and memory consumption. Instead, we use the model described in Section II for comparison against a distributed heuristics easily implementable on WSN nodes, as described in Section IV.

Some solutions exist to route from multiple sources to mobile sinks. In the TTDD protocol [35], nodes are organized in a two-level hierarchy that determines a subset of nodes responsible to track the sinks' position. Kim et al. [32] use stationary nodes as anchors to build routing trees on behalf of mobile sinks. In-network aggregation may also be employed, e.g., using one mobile sink to build a primary tree opportunistically shared with other mobile sinks [26]. Although MUSTER was designed with static sinks in mind, there are synergies worth exploring. For instance, in principle, a variation of our path merging strategy could be used to build energy-efficient trees at the anchor nodes [32]. Whether this is practical, however, depends on the sink mobility. A treatment of this topic is outside of the scope of this paper.

Single-sink and single-source communication. A few works rely on multiple sinks as cluster-heads, where however each source reports to only *one* sink. Some models study the optimal placement of sinks w.r.t. given metrics. Oyman and Ersoy [39] target a pre-specified minimum lifetime. Vincze et al. [53] study how to minimize the average source-sink distance. Das and Dutta [15] define optimal policies to select, at each source, the sink minimizing energy consumption. These scenarios differ from ours in that we require sinks to *simultaneously* collect data from possibly overlapping subsets of sources.

A wealth of work exists on single-sink scenarios. Existing approaches mainly focus on metrics to build efficient routing trees [25], [28], [54] and reliability mechanisms [47]. The former require accurate link quality estimators, an issue orthogonal to our approach. Although in Section IV we describe simple and

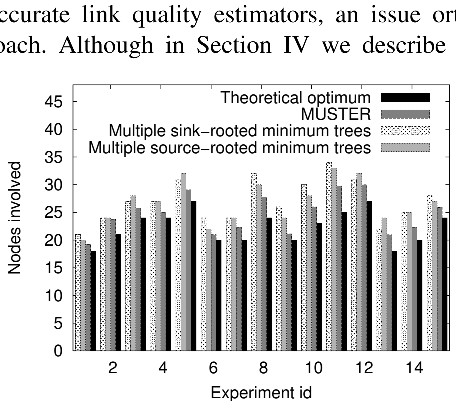


Fig. 29. Nodes involved in routing.

efficient solutions, more sophisticated metrics may be embedded in MUSTER. Similarly, further reliability mechanisms may be integrated, increasing delivery but also overhead.

Intanagonwiwat et al. [27] present a protocol to build routing trees exploiting in-network aggregation. They build a single source-sink path first, then shared by other sources. The resulting tree likely shows a "backbone" effect, which makes nodes aggregate data earlier. Although this resembles our path merging scheme, their solution is not directly applicable in multi-sink scenarios and, unlike MUSTER, it does not provide load balancing. Moreover, they study the performance of their protocol only through simulations based on a IEEE 802.11 MAC layer, which makes their results difficult to compare with ours.

As for one-to-many communication, Cao et al. [9] observe that most existing WSN multicast protocols address specific settings. Their uCast protocol targets scenarios where destinations are physically co-located. Egorova and Murphy [20] rely on a reinforcement learning approach that requires the network topology to be sufficiently stable for the protocol to converge. VLM^2 [46] and TinyADMR [10] focus on multicasting to a small set of mobile targets. A number of solutions are also based on the geographical position of nodes [22]. Common to all these approaches is the optimization of routes at each source independently of the others, whereas in MUSTER these collaborate by merging parallel paths towards multiple sinks.

Solutions providing one-to-one communication in WSNs also exist [8], [23], [37]. In these cases, the protocols optimize the source-sink paths individually, typically by reducing the stretch over the shortest path. In a sense, this represents the extreme opposite to MUSTER. Indeed, our goal is to exploit the spatial and temporal co-location of multiple source-sink paths to prolong the system lifetime. Our load balancing scheme takes advantage of the available resources in exchange for slightly longer paths.

VIII. Conclusion

We presented MUSTER, a protocol expressly conceived for many-to-many communication in WSNs. We studied the problem from an analytical standpoint, by devising a model inspired to the multi-commodity network design problem, used to compute the optimal routing topology in a centralized fashion. The distributed path merging and load balancing techniques implemented in MUSTER allow us to obtain routing paths whose cost is within 10% of the optimum, and evenly distribute the routing effort. By combining these techniques, MUSTER enjoys 2.5 times the data yield of mainstream protocols under the same settings. Moreover, it amplifies the beneficial effects of in-network data aggregation, yielding the user 4 times the amount of data delivered by other protocols. MUSTER is publicly available as open source [1].

Acknowledgements. The authors wish to thank Roberto Cordone and Marco Trubian for their suggestions on modeling the multiple sources to multiple sinks routing problem, Pietro Ciciriello and Flavio Pompermaier for their work on the implementation of MUSTER, and Renato Lo Cigno for his comments on an early draft of this paper. This work is partially supported by the Autonomous Province of Trentino, Italy, under the TRITon project, by the EU Cooperating Objects Network of Excellence (CONET), by the Swedish Foundation for Strategic Research (SSF), and by the Swedish Agency for Innovation Systems (VINNOVA).

References

[1] http://d3s.disi.unitn.it/software/muster.

[2] www.gnu.org/software/glpk.

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

[4] J.N. Al-Karaki and A. E. Kamal. Routing techniques in wireless sensor networks: A survey. *IEEE Wireless Communications*, 11(6), 2004.

[5] A. Bakshi, V. K. Prasanna, J. Reich, and D. Larner. The Abstract Task Graph: A methodology for architecture-independent programming of networked sensor systems. In *Workshop on End-to-end Sense-and-respond Systems (EESR)*, 2005.

[6] G. Barrenetxea, F. Ingelrest, G. Schaefer, M. Vetterli, O. Couach, and M. Parlange. Sensorscope: Out-of-the-box environmental monitoring. In *Proc. of the* 7^{th} *Int. Conf. on Information Processing in Sensor Networks (IPSN)*, 2008.

[7] J. Burrell, T. Brooke, and R. Beckwith. Vineyard computing: sensor networks in agricultural production. *IEEE Pervasive Comp.*, 3(1), 2004.

[8] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron. Virtual ring routing: network routing inspired by dhts. *SIGCOMM Comput. Commun. Rev.*, 36(4), 2006.

[9] Q. Cao, T. He, and T. Abdelzaher. uCast: Unified connectionless multicast for energy efficient content distribution in sensor networks. *IEEE Trans. Parallel Distrib. Syst.*, 18(2), 2007.

[10] B. Chen, K. K. Muniswamy-Reddy, and M. Welsh. Ad-hoc multicast routing on resource-limited sensor nodes. In *Proc. of the* 2^{nd} *Int. Wkshp. on Multi-hop Ad-hoc Networks: From Theory to Reality*, 2006.

[11] Y. Chou. *Statistical Analysis*. Holt International, 1975.

[12] P. Ciciriello, L. Mottola, and G.P. Picco. Building virtual sensors and actuator over Logical Neighborhoods. In *Proc. of the* 1^{st} *ACM Int. Wkshp. on Middleware for Sensor Networks (MidSens)*, 2006.

[13] D.S.J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. *Wirel. Netw.*, 11(4), 2005.

[14] Crossbow Tech. www.xbow.com.

[15] A. Das and D. Dutta. Data acquisition in multiple-sink sensor networks. *Mobile Computing and Communications Review*, 9(3), 2005.

[16] A. Deshpande, C. Guestrin, and S. Madden. Resource-aware wireless sensor-actuator networks. *IEEE Data Engineering*, 28(1), 2005.

[17] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *Proc. of the* 4^{th} *Wrkshp. on Embedded Networked Sensors (Emnets IV)*, June 2007.

[18] Duracell Technical OEM. www.duracell.com/oem/productdata/default.asp.

[19] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler. Energy metering for free: Augmenting switching regulators for real-time monitoring. In *Proc. of the* 7^{th} *Int. Conf. on Information Processing in Sensor Networks (IPSN)*, 2008.

[20] A. Egorova-Förster and A. L. Murphy. A feedback enhanced learning approach for routing in wireless sensor networks. In *Proc. of the* 4^{th} *Workshop on Mobile Ad-Hoc Networks (WMAN)*, 2007.

[21] Energizer Technical Information. http://data.energizer.com/DataSheets.aspx.

[22] C.-H. Feng and W. Heinzelman. RBMulticast: Receiver based multicast for wireless sensor networks. In *Proc. of the Int. Conf. on Wireless Communications and Networking (WCNC)*, 2009.

[23] R. Fonseca, S. Ratnasamy, J. Zhao, C. Tien Ee, D. Culler, S. Shenker, and I. Stoica. Beacon-Vector Routing: Scalable point-to-point routing in wireless sensor networks. In *Proc. of the* 2^{nd} *Symposium on Networked Systems Design and Implementation (NSDI)*, 2005.

[24] Trentino Research & Innovation for Tunnel Monitoring. http://triton.disi.unitn.it.

[25] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proc. of the* 7^{th} *Int. Conf. on Embedded Networked Sensor Systems (SENSYS)*, 2009.

[26] K. Hwang, J. In, and D. Eom. Distributed dynamic shared tree for minimum energy data aggregation of multiple mobile sinks in wireless sensor networks. In *Proc. of* 3^{rd} *European Wkshp. on Wireless Sensor Networks (EWSN)*, 2006.

[27] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proc. of the* 22^{th} *Int. Conf. on Distributed Computing Systems*, 2002.

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

[29] K. Jain, J. Padhye, V. Padmanabhan, and L. Qiu. Impact of interference on multi-hop wireless network performance. *Wirel. Netw.*, 11(4), 2005.

[30] A. P. Jayasumana, Q. Han, and T. H. Illangasekare. Virtual sensor networks - a resource efficient approach for concurrent applications. In *Proc. of the* 1^{st} *Int. Conf. on Information Technology*, 2007.

[31] D. Jung, T. Teixeira, and A. Savvides. Sensor node lifetime analysis: Models and tools. *ACM Trans. Sensor Networks (TOSN)*, 5(1), 2009.

[32] H. S. Kim, T. F. Abdelzaher, and W. H. Kwon. Minimum-energy asynchronous dissemination to mobile sinks in wireless sensor networks. In *Proc. of the* 1^{st} *Int. Conf. on Embedded Networked Sensor Systems (SENSYS)*, 2003.

[33] M. Kodialam and T. Nandagopal. Characterizing achievable rates in multi-hop wireless networks: the joint routing and scheduling problem. In *Proc. of the* 9^{th} *Int. Conf. on Mobile computing and networking (MOBICOM)*, 2003.

[34] O. Landsiedel, K. Wehrle, and S. Gotz. Accurate prediction of power consumption in sensor networks. In *Proc. of the* 2^{nd} *IEEE Wkshp. on Embedded Networked Sensors (EmNets)*, 2005.

[35] H. Luo, F. Ye, J. Cheng, S. Lu, and L. Zhang. TTDD: Two-tier data dissemination in large-scale wireless sensor networks. *Wireless Networks*, 5(11), 2005.

[36] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. of the* 1^{st} *Int. Wkshp. on Wireless Sensor Networks and Applications (WSNA)*, 2002.

[37] Y. Mao, F. Wang, L. Qiu, S. Lam, and J.M. Smith. S4: Small state and small stretch routing protocol for large wireless sensor networks. In *Proc. of the* 4^{nd} *Symposium on Networked Systems Design and Implementation (NSDI)*, 2007.

[38] K. Martinez, J. K. Hart, and R. Ong. Environmental sensor networks. *Computer*, 37(8), 2004.

[39] E. I. Oyman and C. Ersoy. Multiple sink network design problem in large scale wireless sensor networks. In *Proc. of* 1^{st} *Int. Conf. on Communications (ICC)*, 2004.

[40] C. Park, K. Lahiri, and A. Raghunathan. Battery discharge characteristics of wireless sensor nodes: An experimental analysis. In *Proc. of the Int. Conf. on Sensor and Ad-hoc Comm. and Networks (SECON)*, 2005.

[41] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proc. of the* 2^{nd} *Int. Conf. on Embedded Networked Sensor Systems (SENSYS)*, 2004.

[42] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Proc. of the* 5^{th} *Int. Conf. on Information Processing in Sensor Networks (IPSN)*, 2005.

[43] F. Pompermaier. Accurate estimation of residual lifetime in wireless sensor networks. Master's thesis, University of Trento, Italy, 2008.

[44] R. Rao, S. Vrudhula, and D.-N. Rakhmatov. Battery modeling for energy-aware system design. *Computer*, 36(12), 2003.

[45] K. Römer. Distributed mining of spatio-temporal event patterns in sensor networks. In *Proc. of the* 1^{st} *Euro-American Wkshp. on Middleware for Sensor Networks (EAWMS)*, 2006.

[46] A. Sheth, B. Shucker, and R. Han. VLM^2: A very lightweight mobile multicast system for wireless sensor networks. In *Proc. of the* 4^{th} *Int. Conf. on Wireless Communications and Networking*, 2003.

[47] F. Stann and J. Hiedemann. RMST: Reliable data transport in sensor networks. In *Proc. of the* 1^{st} *Int. Wkshp. on Sensor Network Protocols and Applications (WSNA)*, 2003.

[48] TinyOS Web Site. www.tinyos.net.

[49] TinyOS. Multi-hop Routing. www.tinyos.net/tinyos-1.x/doc/multihop/multihop_routing.html.

[50] TinyOS. TEP 105 - Low Power Listening. www.tinyos.net/tinyos-2.x/doc/html/tep105.html.

[51] TinyOS. TEP 119 - Collection. www.tinyos.net/tinyos-2.x/doc/html/tep119.html.

[52] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: Scalable sensor network simulation with precise timing. In *Proc. of the* 4^{th} *Int. Symp. on Information Processing in Sensor Networks (IPSN)*, 2005.

[53] Z. Vincze, V. Rolland, and A. Vidacs. Deploying multiple sinks in multi-hop wireless sensor networks. In *Proc. of Int. Conf. on Pervasive Services*, 2007.

[54] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proc. of the* 1^{st} *Int. Conf. on Embedded Networked Sensor Systems (SENSYS)*, 2003.

[55] B. Y. Wu and K.-M. Chao. *Spanning Trees and Optimization Problems*. Chapman & Hall, 2004.

[56] K. Yuen, B. Li, and B. Liang. Distributed data gathering in multi-sink sensor networks with correlated sources. In *Proc. of* 5^{th} *Int. IFIP-TC6 Networking Conf.*, 2006.

[57] H. Zhang, A. Arora, Y. Choi, and M. G. Gouda. Reliable bursty convergecast in wireless sensor networks. In *Proc. of the* 6^{th} *Int. Symp. on Mobile Ad-hoc Networking and Computing (MOBIHOC)*, 2005.

Luca Mottola Luca Mottola is a Senior Researcher at the Swedish Institute of Computer Science (SICS). Previously, he has been a post-doctoral researcher at the University of Trento (Italy), and a research scholar at the University of Southern California (USA). He carried out his Ph.D. studies at Politecnico di Milano (Italy), completing in 2008. His research interests include the design, implementation, and validation of modern distributed systems, with particular attention to wireless sensor networks and automatic verification of distributed software architectures. He is a member of ACM and IEEE. More information are available at `http://www.sics.se/~luca/`.

Gian Pietro Picco Gian Pietro Picco is an Associate Professor in the Department of Information Engineering and Computer Science (DISI) at University of Trento, Italy. Previously, he has been on the faculty of Washington University in St. Louis, MO, USA (1998-1999) and Politecnico di Milano, Italy (1999- 2006). The goal of his current research is to ease the development of modern distributed systems through the design and implementation of appropriate programming abstractions and of communication protocols efficiently supporting them. His work spans the research fields of software engineering, middleware, and networking, and is oriented in particular towards wireless sensor networks, mobile computing, and large-scale distributed systems. He is a member of ACM and IEEE. More information at `http://disi.unitn.it/~picco`.