Message Efficient Termination Detection in Wireless Sensor Networks

Abstract. Execution of wireless sensor network (WSN) applications typically consists of a number of successive phases, such as network reprogramming, localization, power management, health monitoring, and parameter updates. Termination detection of a phase is therefore a critical operation for a network manager to safely execute a new phase on some or all of the network nodes.

In this paper, we reformulate the well-known problem of termination detection for WSNs, and present a low-cost solution to the problem. Our algorithm, *Reporter*, exploits the reactive nature of WSN protocols as well as the broadcast communication model of WSNs. It detects termination accurately and (message) efficiently, using reports from only a small fraction of nodes in the network. It exploits existing network traffic to construct a routing tree to collect these reports at a base station, and thus reduces the control overhead of structure formation. Moreover, it has low computation and memory overhead. We have developed a TinyOS implementation of *Reporter*, which is easily composed with a number of existing WSN protocols. We provide detailed experimental performance results obtained on an indoor mote testbed which show that *Reporter* selects as few as 5% of the total number of nodes in the network for collecting termination reports while preserving accuracy.

1 Introduction

As demonstrated by several field deployments in environmental, military, industrial and other applications, wireless sensor networks (WSNs) have evolved from a network of dumb sensing devices that pump data to a central node to complex software systems. By way of illustration, the Multi-Target Tracking [1] and *ExScal* [2] systems consist of components for several tasks such as sensing, event detection, routing, tracking, network reprogramming, localization, scheduling, power management, health monitoring, parameter update, etc. The total processing and memory requirements of these software systems exceed the resources available on most existing WSN devices. Consequently, multi-phase execution is a common paradigm in such complex sensor network applications.

In multi-phase execution, the application is broken down into multiple logical tasks that are executed at different times. For instance, the task of localizing the network to determine absolute or relative positions of nodes can be one application phase. Based on this information, the reprogramming of location aware application programs onto all nodes can be a subsequent application phase. Power management, health monitoring, and parameter updates can be regarded as separate phases. To switch between phases, a network manager relies on network management tools provided by services such as Deluge [3] and SNMS [4].

Requirements for application phases. The execution of phases in applications is typically governed by the following requirements.

1. *In-order execution*: Application phases are often dependent on the output of some earlier phases. For instance, an application phase cannot of course start until the network reprogramming phase, which downloads to the network nodes the programs of that application phase, is completed. Similarly, if an event detection and tracking phase involves execution of a location-aware routing service, it must be executed after the localization phase.
2. *Atomic execution*: It is often desirable that all up nodes executing a phase complete it before the next phase starts execution on any node. For one, this may be necessary for correctness of phases which require coordination between nodes for completion; i.e., if a node that completes downloading a new program or its localization operation starts executing its next phase unilaterally, its neighbors that are waiting to receive program download or localization information from it may never complete that phase. Also, even if phases are designed to execute correctly even when previous phases have not completed, the "synchronous transition" implied by atomicity may be desirable for performance reasons; i.e., it may avoid message interference between executions of the two phases and, in several cases, may help the new phase stabilize more efficiently in time or energy consumption. For example, if some nodes were to join a phase late they may cause many nodes to expend more effort, e.g., in routing to enter a beaconing phase to update their current link quality estimates [5] or in distributed time synchronization to choose the preferred time.

Given these requirements, we identify termination detection as a critical network management service. Generally speaking, robustness and minimality are two key design considerations for network management. Robustness implies that network management must function correctly even when an application has failed, thus management services should not critically depend on application services for their operation. For instance, even if the application routing structure is broken, network health and other management data should still be received from the network. Minimality implies that since management services need to co-exist with applications, they should be lightweight and not adversely affect application performance. Hence, while it is possible to add termination detection capabilities separately to individual protocols and application phases, we argue it would be inefficient to do so, given the node and network constraints in WSNs.

A baseline solution for termination detection is to use the standard pattern of Propagation of Information with Feedback (PIF), as is done by multi-hop network querying services such as SNMS [4] or TinyDB [6]. In this approach, applications export a termination attribute to the querying service which creates and maintains its own routing data structure (e.g., a spanning tree) for collecting the query results. The network manager then periodically sends a query wave to all nodes to determine termination status and the results are collected over the pre-constructed structure. This solution is obviously message-inefficient because it requires the manager to collect periodic query results. A simple optimization

to improve the efficiency of this baseline algorithm is to define a one-time query which is triggered at each node upon termination. Regardless, this approach requires every node in the network to respond upon its completion. Moreover, this approach also incurs the additional overhead of having to maintain a data structure for query collection.

Contributions of the paper. In this paper, we reformulate the problem of termination detection for the reactive model that is basis of low power WSN protocols, and propose an efficient termination detection algorithm for WSNs, Reporter. Our Reporter algorithm uses fewer messages than the baseline described above in two ways: first, it only requires a small subset of nodes in the network to send termination reports without compromising correctness; and second, it (re)uses existing network traffic to create a structure for report collection and thus reduces control messaging overhead. Reporter also makes limited assumptions about the protocols whose termination is to be detected and, thus, is highly composable with most existing implementations. Since network based reprogramming is a critical global operation for most WSNs, we use reprogramming as a running case study to concretely illustrate the working of our algorithm. We demonstrate the improvement in efficiency and reduction in overhead through real experiments based on the TinyOS [7] implementation of Reporter, that detect termination in Deluge [3] and Sprinkler [8], which are two popular WSN reprogramming protocols. Our experiments show that although these two protocols are quite different in their design and operation, Reporter yields comparable performance for both, providing further evidence of its applicability to other protocols.

Organization of the paper. The rest of the paper is organized as follows. In Section 2, we define the termination detection problem for WSNs and the system model assumed in the rest of the paper. Section 3 describes the Reporter algorithm and derives various correctness properties, while Section 4 provides implementation details and experimental performance validation. We discuss related work in Section 5 and make concluding remarks Section 6.

2 System Model and Problem Definition

In this section, we state the system model for WSNs that is assumed in the rest of the paper and reformulate the problem of detecting termination of a distributed protocol for this model.

2.1 System model

We begin with the network model, and then present the model for the underlying application protocol for which termination is to be detected.

Network model. We assume a connected, multi-hop network of WSN nodes. Associated with each node is a unique identifier. We assume that the wireless links that nodes use to communicate with each other are bidirectional; the reliability in both directions need not however be the same: if a link exists in

one direction with some non-trivial reliability, then a link exists in the opposite direction as well with some non-trivial, albeit different reliability. Given that communications are sent over a broadcast channel, we also assume that a node can *snoop* on message traffic in its radio neighborhood, i.e., receive messages sent by nodes in its one-hop neighborhood that are not intended for it.

We assume there is a single distinguished node called the base station from which the protocol is initiated and where termination reports are collected. The network manager uses the information collected at the base station to detect termination, hence the terms base station and manager are used interchangeably in the paper.

Application protocol model. Network protocols may be classified as proactive or reactive. We assume that most WSN protocols are inherently reactive; i.e., they become busy only in response to some input from the environment, such as a message received over the radio or data/events received from a sensor. Unlike traditional distributed systems, where a node can be in two possible states, idle or active, we identify a third state *done*, and a set of transition rules that must be followed by nodes in our reactive:

1. A node is initially idle and continues to remain idle unless it receives a protocol message, in which case it becomes active.
2. Upon becoming active, a node can perform local computation, send, and receive protocol messages; however, if the node does not receive any protocol messages for an interval of time T, it goes to the done state. Note that a node can remain active indefinitely by continuing to receive protocol messages.
3. Once a node enters the done state, it does not receive any more protocol messages and does not become active again.

At first glance, this model may seem restrictive, but a closer inspection of WSN protocols reveals that most if not all terminating WSN protocols do in fact satisfy it. For instance, during network reprogramming, WSN nodes remain active as long as someone in their neighborhood has not received some part of the program being downloaded. Further, nodes with incomplete programs initiate requests for missing data and receive it within bounded time specified by the protocol. However, once all nodes in the neighborhood of a node have acquired the program, it does not receive any more requests and enters the done state as described above. A similar argument holds for cooperative localization protocols that do RSSI or acoustic ranging between neighboring nodes to figure out relative distances from one another. Once a node has finished estimating its distance with respect to all its neighbors, it is done. Other examples of protocols that satisfy our reactive model include power management (sleep/wakeup) and parameter update, and for these the time bound T is a small constant that is independent of the size of the network. The PIF protocol itself also satisfies our reactive mode, but in this cases T depends on the size of the network, as a node that sends an outgoing/propagation wave along its subtree and has to wait for its subtree to finish before it can send its incoming/completion reply to its parent.

For ease of exposition, we consider the case where the underlying protocol is executed on all nodes in the connected network, however the ideas in this paper apply to protocols that only execute on certain subsets or groups of nodes in the network.

We do not assume any knowledge of the internal operation of the protocol or its message formats for the purpose of termination detection. We do however assume knowledge of two protocol parameters. The first is the upper bound T on the reactive computation. This time bound T is usually specified as a protocol parameter by the designer or can be readily inferred from the performance of the protocol. The second parameter is the message type(s) used by nodes for exchanging protocol messages. This message type is similar to a port number in Internet applications and is usually well-advertised by protocol designers to avoid conflicts. We thus argue that these assumptions are reasonable and our approach is essentially a black-box one.

2.2 Problem

The problem of termination detection requires a network manager to detect that the underlying distributed protocol has terminated at all nodes. Termination detection requires the following standard properties of safety and liveness:

- *Safety*: If the manager detects termination, then the underlying protocol must indeed have terminated.
- *Liveness*: If the underlying protocol has terminated, then the manager must eventually detect termination.

Given the differences between traditional distributed systems and the WSN model, we impose the following additional requirements:

- *Energy efficiency*: The detection must introduce minimal additional overheads, since WSN nodes have limited energy resources and message transmission and reception is a significant energy drain.
- *Composability*: For the detection to be generally used as a management service, it should easily compose with different types of network protocols. (This motivates that detection should not exploit any particular protocol data structures or implementation details.)

3 The Reporter Algorithm

We now describe our Reporter algorithm for efficient termination detection in WSNs. The baseline algorithm described in Section 1 requires continual querying of the entire network and hence is not only energy inefficient but yields more unreliability as responses from the entire network create a traffic burst leading to congestion and contention losses in the network. Studies such as [9, 10] have shown that in large scale networks, the reliability of such global data collection typically lies between 50-90%. Further, this algorithm depends on an external routing structure for actually collecting termination reports. Constructing and

maintaining this structure imposes additional messaging and performance overheads.

The basic idea underlying Reporter is (i) to identify a small set of *local reporter* nodes that can detect termination locally by exploiting the reactive protocol model in WSNs, and (ii) to exploit existing protocol traffic to construct a routing tree over which termination reports can be collected. We can prove that receiving local termination reports from this small set of reporter nodes is sufficient for detecting global termination of the underlying protocol. Thus, the design of Reporter involves solving three subproblems, viz., **efficient selection of local reporter nodes, autonomous structure creation** and **detecting global termination from local reports**.

Reporter is specified in guarded command notation in Figure 1. We now describe how Reporter solves the subproblems listed above.

3.1 Efficient selection of local reporter nodes

Given our model of reactive, broadcast-based WSN protocols, it is possible to infer termination from only a small fraction of nodes, where each report not only implies termination of that node, but also of other nodes around it. Reporter selects these local reporter nodes (which we henceforth abbreviate as reporter nodes) via the following rule:

> *Rule 1: Reporter selection*: If, at the time of sending an underlying protocol message, a node has not received a message from any local reporter, it elects itself as a local reporter.

The reporter selection rule is implemented by actions A_1 and A_2 in Figure 1. In action A_1, a node decides whether or not it should become reporter according to the above rule. Additionally, it also piggybacks its decision on the outgoing message. Conversely, in action A_2, for every received protocol message, a node checks whether or not the sender is a reporter and if so, records that it has now seen a reporter.

It follows from Rule 1 that the set of reporters is a subset of nodes that send protocol messages during its execution. In any protocol execution, some nodes both send and receive protocol messages while others only overhear messages sent by others. For example, in a reprogramming protocol, some nodes send protocol messages in the form of the new program while some send requests for missing parts of the program from neighbors, while others simply acquire the new program by overhearing these messages. Based on Rule 1, we can prove the following property about the reporter set selected by our algorithm.

Property 1: The set of reporter nodes in a protocol execution forms a Dominating Set over the set of nodes that send protocol messages.
(Proof) A set of nodes D is defined to be a Dominating Set of a set of nodes N if every node in N is within one-hop of some node in D. Now consider the set S of nodes in a protocol execution that send protocol messages and assume that the set of reporters R is not a Dominating Set of S. Then there must exist a node s that sends a protocol message, yet there is no reporter in the one-hop

Algorithm Reporter

Var $is\_reporter$: boolean
$seen\_reporter$: boolean
$parent$: integer

Initially $(parent = -1) \ \wedge \ (!is\_reporter) \ \wedge \ (!seen\_reporter)$

Actions

$\langle A_1 \rangle :: send\_pdata = TRUE \ \longrightarrow$ **if** $(!seen\_reporter)$
$is\_reporter \ := \ TRUE;$
$seen\_reporter \ := \ TRUE;$
fi
$send(myid, \ is\_reporter, pdata);$

[]

$\langle A_2 \rangle :: rcv(id, reporter, pdata) \ \longrightarrow$ **if** $(parent \ < \ 0) \ parent \ := \ id;$ **fi**
if $(reporter) \ seen\_reporter \ := \ TRUE;$ **fi**
$process\_rcvd\_msg(pdata);$

[]

$\langle A_3 \rangle :: nbr\_activity\_timeout \ \longrightarrow$ **if** $(is\_reporter) \ send\_report(parent, myid);$ **fi**

[]

$\langle A_4 \rangle :: rcv\_report(i, \ j) \ \longrightarrow$ **if** $(i = myid) \ send\_report(parent, \ j);$ **fi**

Fig. 1. The Reporter algorithm for termination detection

neighborhood of s. However, according to Rule 1, s would itself have become a reporter, which violates the assumption. □

We now identify a rule by which reporters can detect local termination of the underlying protocol and thereby report this to the manager.

> *Rule 2: Local termination detection*: If a reporter node does not receive any messages in an interval of time T, it detects termination locally.

Given the time bound T in the reactive protocol model and the fact that our algorithm snoops on protocol traffic in the broadcast WSN model, the above rule readily detects local termination. This rule is implemented by action A_3 in Figure 1. Upon detecting local termination, a reporter sends a report message to the base station. We now prove the correctness of Reporter as follows.

Property 2: The safety and liveness requirements of termination detection are satisfied if all local termination reports from the set of reporters are received.
(Proof) From Rule 2, we know that if all nodes in the network have terminated or are in the done state, all reporter nodes will detect termination locally as they will not receive any protocol messages for time T. Thus, when all reports are received, the manager can detect termination and liveness is satisfied.

To prove safety, we must show that the manager does not detect termination when some nodes are still active. Assume that the manager receives reports from all reporters and detects termination before the protocol has terminated. If protocol execution has not yet terminated, there must exist some active node

p in the network. From our reactive protocol model, we know that for p to be active, it must have received a message from some node q within the last T time. From Property 1, the set of reporters is a Dominating Set of the set of nodes that send protocol messages. hence there must exist a reporter r within the one-hop neighborhood of q and r must have also received the message sent by q within the last T time. However, according to Rule 2, r could not have detected local termination if it received a message in the last T time. This is a contradiction. □

We therefore conclude that the set of reporters, which is a Dominating Set of the set of nodes sending protocol messages, suffices for termination detection. An optimal solution would thus require the reporter nodes to form the Minimum Dominating Set (MDS) of nodes that send protocol messages. However, computing the MDS is NP-hard and would require exponential time. Existing approximation solutions either perform much worse than the optimal or require non-local computations or some knowledge about the network topology. Also, the set of nodes that send protocol messages could vary over different protocol executions. We therefore eschew computing the MDS of the network.

We experimentally validate, in Section 4, that our simple reporter selection rule, which does not assume any topology knowledge or require non-local computations, performs comparably to the MDS solution, for the particular networks considered in our experiments.

3.2 Routing structure creation

Reporter solves the second subproblem of autonomous operation by creating its own structure for collecting termination reports. Our approach is similar to that used in the classic Dijkstra and Scholten [11] termination detection protocol, and constructs a collection structure efficiently by once again exploiting protocol traffic, according to the following rule.

> *Rule 3: Parent selection*: Every node selects the first node from which it receives a protocol message as its parent.

This rule is implemented by action A_2 in Figure 1. Initially, all nodes have invalid parents. Upon receiving a message, a node checks whether it has an invalid parent. For the first message received, this check succeeds and the node selects the sender of the received message as its parent. For subsequent messages, the check will fail as it already has a valid parent. Based on Rule 3, we can prove the following property about the structure created by the selected parent links.

Property 3: Reporter constructs a spanning tree over the entire network rooted at the base station.
(Proof) Rule 3 defines exactly one parent for every node. Since the network is connected, every node receives at least one protocol message and acquires a parent. There also cannot exist any cycles in our structure because if two nodes p and q select each other as parents then they must have received their first protocol message from each other, which is a contradiction. (A similar argument

applies to cycles of arbitrary length). Since the base station invokes the underlying protocol by sending the first message, we obtain a spanning tree rooted at the base station. □

From our link model, we know that if a parent-child link exists, then the child-parent link has some non-trivial reliability. This reliability can be further improved using acknowledgements and retransmissions. We show in Section 4 that despite using the simple spanning tree construction in our algorithm, we obtain almost 100% reliability with very basic per-hop reliability mechanisms.

Of course, an application may already be maintaining a routing structure, perhaps one that selects better links using estimation or exploiting topological knowledge. In such cases, the network manager is free to reuse the existing structure to collect termination reports. Our simple algorithm however is lightweight, designed to work with all applications and suffices to provide reliable results.

3.3 Detecting global termination from local reports

In Section 3.1, we proved that the network manager could safely conclude global termination of the underlying protocol if it received report messages from all reporters. However, the set of reporters is generally not fixed. We therefore describe two techniques whereby the manager can learn that the set of reporters from which reports are received is complete.

For the first technique, we make the additional assumption that the manager, through a localization service, knows the locations of all nodes in the network. Assuming a disk model for communication, the manager can identify regions that must have terminated from the source locations of received reports. Even if the communication range of nodes is not a unit disk, previous research by Zhao et al. [12] has identified the existence of a stable region, referred to as *inner band* within which communication is reliable. Of course, some nodes outside of the inner band may still be able to communicate with the node over what are referred to as *long links*. However, this is not a problem as our algorithm is conservative, so if a node that has not yet terminated has a long link to some reporter, then that reporter will not detect termination in its neighborhood. Thus, a non-terminated node may slow down one or more reporters, however once the protocol has terminated, our algorithm is guaranteed to detect it.

The second technique requires the manager to learn its reporter set. In this approach, when a node elects itself as reporter, it communicates this to the manager using the constructed spanning tree. During protocol execution, the manager thus learns about the reporter set and can then wait to receive termination reports from all nodes in this set before detecting termination. This approach does not assume any localization information, but it requires twice the number of messages.

To guarantee correctness deterministically, our algorithm requires messages from all reporters to be received at the base station. However, due to faults such as reporter failure or message loss, some reports may never be received. Even in the absence of such faults, some node executing the protocol might start misbehaving by sending arbitrary protocol messages. In this case, its reporter,

although correct, will not detect and report local termination. In such scenarios, a network manager may be forced to live with only probabilistic guarantees about the correctness of termination detection and deal with faults through predefined network policies. One example of such a policy could be to wait for a fixed time to receive termination reports and declare termination of the underlying protocol if more than x% of the nodes (say 90%) could be inferred to have terminated. Another policy could be to send out explicit network queries using more expensive protocols such as SNMS or TinyDB to regions from which no termination reports are received.

4 Implementation and Experimental Results

In this section, we first discuss some implementation considerations for Reporter. We then present results from a case study in which we integrated our implementation with two reprogramming protocols, Deluge [3] and Sprinkler [8], and ran several experiments on an indoor WSN testbed to detect their termination, so as to validate the correctness and performance of our implementation.

4.1 Implementation details

We implemented Reporter in TinyOS [7] as a reusable component that is easily integrated with different types of network protocols. The time bound T for protocol reactiveness and the message type(s) used by the protocol (referred to in TinyOS as handler-id), assumed known to Reporter, are specified as input parameters by the manager in a header file. The Reporter component includes two modules, one for snooping and the other for detecting and reporting local termination.

We implemented our snooping module at the level of the generic communication module, known as GenericComm, which is responsible for delivering messages to and from TinyOS components to the radio layer, for two reasons. First, our algorithm needs to snoop on protocol messages to learn about reporters in its neighborhood and to detect local termination. Second, our algorithm piggybacks reporter information on protocol messages (action A_1), hence it needs access to these messages before they are sent out. Since all message communication for TinyOS components is handled by the GenericComm component, implementing the snooping module at this level is most efficient.

The detection and reporting module maintains a timer set to expire after the time T specified in the header file, and resets this timer every time the snooping module receives a protocol message. If this timer ever expires, local termination is detected and reporter nodes send a report message to their parent to be forwarded to the manager. Since reliable delivery of termination reports is critical, we implemented a message buffer at intermediate nodes. Report messages are enqueued in this buffer and retransmitted until they are acknowledged by the parent or until the maximum number of retransmissions is exceeded. Our implementation thus uses explicit acknowledgements provided by the TinyOS MAC along with retransmissions to improve message reliability on a per-hop basis.

4.2 Performance

We first analyze the computation, memory and messaging overheads for our implementation, and then present experimental results.

Overhead. As seen from action A_2 in Figure 1, Reporter performs two extra comparisons to check whether or not the node has a parent and whether or not the sender is a reporter, for every received protocol message. Similarly, for every send operation as shown in action A_1, we require one extra comparison. However, the actions contained within each of these comparisons only need to be executed once per execution of the protocol; once assigned, a parent is not changed. Reporter adds one extra bit to each protocol message to indicate whether the sender is a reporter or not. For TinyOS, we also had to modify the message structure as the sender-id was not included in a message by default. We also require one message when a reporter's timeout expires. Reporter thus has very low computation and message overhead.

Finally, our implementation of Reporter has a very light memory footprint, requiring only few hundred lines of code and around 50 bytes of additional RAM.

Reporter is thus easy to implement with very little computation, communication and memory overheads.

Experimental Setup. We tested the correctness and performance of Reporter by performing a series of experiments using our TinyOS implementation composed with the Deluge and the Sprinkler reprogramming protocols as a case study. We used an indoor testbed of 105 WSN nodes deployed in a 15x7 grid pattern. We present here a brief overview of the two protocols, highlighting the key differences between the two.

• *Deluge*: Deluge is an unstructured, flooding-based protocol that is designed to work even when no information about the network topology is available. In Deluge, nodes periodically advertise their program version and the fraction of this latest program they possess. Whenever a node receives an advertisement for a higher version number or for a missing fraction of its current version, it sends a request message to the advertiser, which responds with the requested program fraction. Deluge specifies upper bounds on the time interval between advertisements, time within which a request for missing program fractions should be made and the time within which such a request should be responded to; it thus satisfies our reactive protocol model. To prevent network contention due to redundant transmissions, Deluge uses a suppression mechanism whereby nodes refrain from sending new request messages as long as they receive useful program fractions by overhearing request/responses from others. However, the selection of which nodes send the request messages is done randomly, hence it varies across different protocol executions.

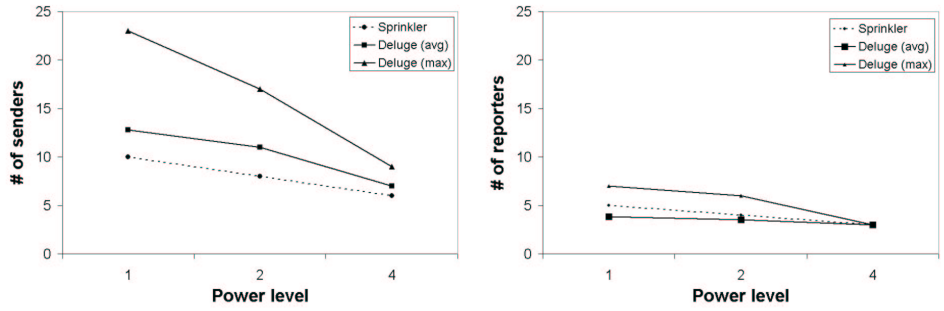
• *Sprinkler*: Sprinkler is a structured protocol that exploits topology information about a network to construct a backbone structure for program dissemination. The backbone constructed by Sprinkler is within O(1) of the Minimal Connected Dominating Set. Additionally, Sprinkler locally computes a TDMA schedule for disseminating new programs over the backbone node. Thus, Sprinkler preselects

nodes that disseminate the new program and carefully schedules transmissions across these nodes to cut down message losses due to collisions and interference. The upper bound T for message forwarding can be derived from the TDMA schedule constructed by Sprinkler; it too thus satisfies our reactive model.

We selected Deluge and Sprinkler for our experiments because they are reliable, have been widely used in field deployments, and also use two completely different approaches for reprogramming.

In each experiment run with each of the two protocols, we invoked a new reprogramming operation with a small, fixed new program. During a run, each node logged whether or not it sent any protocol messages and whether it became a reporter, the id of its parent, and the id(s) of reporter(s) in its one-hop neighborhood that it had overheard. The logged data from all nodes was collected reliably using the Ethernet backchannel in the testbed to construct the actual reporter set and the collection data structure. At the end of each run, reporters detected termination in their one-hop neighborhoods and sent report messages along their chosen parent links to the base station. Received reports were then compared with the actual reporter set to calculate reliability.

Results. Figure 2(a) plots the number of senders in the network for each reprogramming protocol and highlights the differences in their dissemination patterns. Since the inter-node separation was fixed for the testbed we used, we ran these experiments using different transmit power levels to create networks with different effective densities. The X-axis in the graph denotes power level while the Y-axis denotes the number of nodes that sent protocol messages.



(a) Dissemination patterns for Deluge vs Sprinkler

(b) Performance of reporter selection for our algorithm

Fig. 2. Efficiency of reporter selection in our algorithm.

As seen from Figure 2(a), the average number of senders in Deluge is more than that in Sprinkler, as expected. Also, while the Sprinkler backbone size remains fixed, the number of distinct senders in Deluge can vary quite a bit across different runs. For instance, at power level 1, we observed that as many as 23 out of 105 nodes sent a Deluge message in some execution while the number of senders in Sprinkler at the same power level was always 10.

However, even for these two protocols, that work quite differently, the number of reporters selected by our algorithm is comparable for the same power level (network density), as illustrated in Figure 2(b). Our experiments thus demonstrate the performance benefits of reporter selection in our algorithm. At the lowest power level (lowest node density), only **4-7%** of the total number of nodes became reporters while at higher power levels, i.e. highest node density, this number was even smaller. In most WSNs, network density is dictated by sensing range, as it is typically smaller than communication range. Reporter achieves substantial performance savings in such dense networks.

Figure 3 shows the spatial distribution of senders and reporters in the 15x7 network from one execution from both protocols at power level 2. The arrows indicate the routing paths chosen by Reporter for collecting termination reports. We see that the dissemination pattern of Sprinkler is regular and deterministic while that of Deluge is random. However, the net performance of Reporter is quite similar for both protocols. It should be noted that in a small number of executions with the Deluge protocol, our algorithm did end up with reporters that were within one-hop of each other. This occurred due to message collisions during transmission as a result of which the piggybacked reporter selection was not heard by the second node that also chose itself to be a reporter. This does not affect the correctness of Reporter, and only slightly affects its performance.

In the baseline algorithm described in Section 1, every node sends a termination report to the base station which creates a burst of network traffic. Reliable end-to-end delivery of such traffic bursts is especially challenging, with even the best known solutions only achieving about 90% reliability. As shown in Figure 3, our algorithm selects only about 5% of nodes as reporters that are somewhat evenly spread across the network. For this less severe traffic load, even our simple, per-hop reliability mechanisms gave us 98.4% routing reliability on average for report collection at the base, with 92% of the runs yielding 100% reliability.

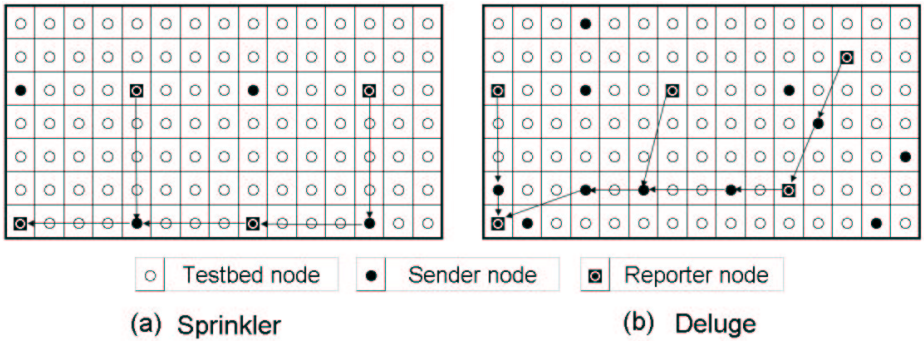


Fig. 3. Spatial distribution of reporters selected by our algorithm.

For the network setup and power level shown in the Figure 3, the size of the Minimum Dominating Set is 3 nodes, whereas our algorithm selects 4 nodes as reporters in the average case. However, our algorithm is forced to select the base

station, which for this setup is a corner node, as a reporter since it sends the first protocol message. These results are reproduced at other power levels too where the number of reporters selected by our algorithm matches the calculated size of the Minimum Dominating Set.

5 Related Work

The problem of distributed termination detection was introduced by Dijkstra and Scholten [11] in 1980 and has been extensively studied since [11, 13–15]. While our approach for tree construction for collecting reports is similar to the one proposed in [11], our ideas for detecting termination differ from existing approaches in several ways. Existing termination detection algorithms require explicit signalling of control information between neighboring nodes and to the base station to detect termination whereas we exploit the reactive nature of WSNs and the broadcast communication model to reduce global termination detection to locally detecting neighborhood termination at a small fraction of nodes in the network. We also do not require all nodes to signal termination unlike previous approaches. Rather, our reporter algorithm once again exploits the underlying protocol to select a small faction of nodes as reporters and is thereby well suited to WSNs where message efficiency is highly desirable.

The concept of a Dominating Set has been used in protocols for bulk dissemination [8, 16] and clustering [17, 18]. However, these protocols typically use location information or introduce their own control messages to create a clustering structure in the network. Our algorithm does not assume any network information and imposes minimal control overhead for selecting reporters. Clustering algorithms also typically incur control overhead for maintaining clusters in the presence of node joins and failures. Unlike such static clustering schemes, our algorithm performs a one-shot, on-demand selection of reporters for every execution of the underlying protocol. Also, as shown by our experiments with Sprinkler, Reporter automatically exploits any existing structure of the underlying protocol, however it can work equally well with unstructured protocols, as demonstrated by or results for Deluge.

6 Conclusions

Termination detection is a critical service for management of multi-phase WSN applications. In this paper, we reformulated the classical termination detection problem by identifying a reactive model that is commonly used in low-power WSN protocols, and proposed Reporter, an algorithm for efficient termination detection in this model. Reporter imposes minimal computation, communication and memory overheads while achieving significant message efficiency over existing solutions. This improved efficiency also results in an improvement in reliability, which would otherwise be hard to achieve. We implemented Reporter as a TinyOS component that is readily integrated with most WSN protocols to detect their termination. Finally, we validated the correctness and performance

of Reporter through experiments on a real WSN testbed for two popular reprogramming protocols. Our experiments show that even though the two protocols are quite different, Reporter achieves comparable performance for both, and requires only about 5% of the number of nodes required by existing protocols to send termination reports.

Future work. WSNs are subject to a wide variety of faults which may happen at anytime during protocol execution. A fault that significantly affects Reporter is the failure of a reporter node during termination detection. Since our algorithm tries to maintain only one reporter in a neighborhood, failure of this reporter implies that termination information from this neighborhood will not be sent to the base station. This failure can be dealt with either by defining network policies, as we described in the paper, or by adding fault-tolerance to the algorithm itself. One approach to fault-tolerance is to guarantee that there exist at least k reporters in every one-hop neighborhood, as opposed to the single reporter in our current algorithm. While this extension seems simple, it is not easy to implement locally in a distributed manner because for $k > 1$, a node cannot decide locally whether including itself in the set will violate the global property. Thus, we seek solutions that require local communication and synchronization between nodes, are fault-tolerant and stabilizing, and do not impose too much overhead.

An alternative approach for network management assumes that WSNs are inherently probabilistic in nature and therefore achieves efficiency by sampling the network, instead of collecting full information, to provide probabilistic guarantees about network state. Sample based approaches require white-box information about the underlying application protocol to obtain the guarantee; the approach is thus less general than the black-box approach considered in this paper. We intend to study sampling-oriented solutions for various network management tasks, including termination detection, and to compare their correctness and performance advantages over their deterministic counterparts.

Finally, automation of a WSN manager requires the development of a framework for scripting/orchestrating the execution of multi-phase WSN applications. The framework would be useful not only in the field but also in the testing and validation stages. Reporter is a useful building block for such a framework. We have recently started development of the framework.

References

1. P. Dutta and et al. Trio: enabling sustainable and scalable outdoor wireless sensor network deployments. In *5th Intl. Conf. on Information processing in Sensor Networks (IPSN)*, 2006.
2. A. Arora et al. ExScal: Elements of an extreme scale wireless sensor network. In *11th IEEE Intl. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 102–108, 2005.
3. J. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *2nd Intl. Conf. on Embedded networked sensor systems (SenSys)*, pages 81–94, 2004.

4. G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. In *Proceedings of the EWSN*, 2004.
5. H. Zhang, A. Arora, and P. Sinha. Learn on the fly: data-driven link estimation and routing in sensor network backbones. In *25th IEEE International Conference on Computer Communications (INFOCOM)*, 2006.
6. S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.
7. Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.
8. V. Naik, A. Arora, P. Sinha, and H. Zhang. Sprinkler: A reliable and energy efficient data dissemination service for wireless embedded devices. In *26th IEEE Real-Time Systems Symposium (RTSS)*, 2005.
9. S. Bapat, V. Kulathumani, and A. Arora. Analyzing the yield of ExScal, a large-scale wireless sensor network experiment. In *13th IEEE Intl. Conf. on Network Protocols (ICNP)*, pages 53–62, 2005.
10. H. Zhang, A. Arora, Y. Choi, and M. Gouda. Reliable bursty convergecast in wireless sensor networks. In *6th ACM Intl. Symp. on Mobile ad hoc networking and computing (MobiHoc)*, pages 266–276, 2005.
11. E. Dijkstra and C. Scholten. Termination detection for diffusing computations. In *Information Processing Letters 11(1):1-4*, 1980.
12. J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *1st Intl. Conf. on Embedded networked sensor systems*, 2003.
13. K. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. In *ACM Trans. on Computer Systems, 8(3):326343*, 1985.
14. S. Chandrasekharan and S.Venkatesan. A message-optimal algorithm for distributed termination detection. In *Journal of Parallel and Distributed Computing, 8:245252*, 1990.
15. F. Mattern. Global quiescence detection based on credit distribution and recovery. In *Information Processing Letters 30*, pages 195–200, 1989.
16. S. Parthasarathy and R. Gandhi. Fast distributed well connected dominating sets for ad hoc networks. Technical Report CS-TR-4559, Univ. of Maryland, 2004.
17. I. Chlamtac and A. Farago. A new approach to the design and analysis of peer-to-peer mobile networks. In *Wireless Networks, 5:149-156*, 1999.
18. Y. Chen and A. Liestman. Approximating minimum size weakly connected dominating sets for clustering mobile ad hoc networks. In *3rd ACM Intl. Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2002.