A Framework for Connectivity Monitoring in Wireless Sensor Networks

Daniel Pfleger, Ulrich Schmid
Institute of Computer Engineering
Vienna University of Technology
Vienna, Austria
Email: {dpfleger, s}@ecs.tuwien.ac.at

***Abstract*—We present an overview of the features and the architecture of a framework for long-term monitoring of the communication topology of a synchronous wireless sensor network (WSN). Our framework, a prototype of which has been implemented for Atmel RZ200 motes running TinyOS, locally records the complete connectivity information for every node in every round, and disseminates this information to one or more root nodes that act as data sinks for monitoring information. Postprocessing tools allow to query the recorded communication graphs, in order to, e.g., verify structural properties like the presence of multiple strongly connected components. We demonstrate the utility of our framework by means of an experimental evaluation of the coverage of a recently introduced adversarial network model for directed dynamic networks. Our measurement results reveal that it has a very good coverage in several small-scale WSN deployments.**

Keywords–Wireless Sensor Networks; Monitoring; Network Topology.

I. INTRODUCTION

The design and analysis of algorithms and protocols for dynamic networks [1] has always been a very active area in networking and distributed computing research. In this research, many "abstract" network models, both adversarial and probabilistic, have been defined and used for validating the correctness and performance of network protocols.

One example of an adversarial model is T*-interval connectivity* [2], where one assumes that the communication topology may vary arbitrarily over time, except that a subgraph that spans all n nodes in the system must be stable in every sliding window of duration T. The advantage of an adversarial model is that it allows the design of protocols that *guarantee* certain properties (provided the assumptions of the model hold). Probabilistic models, on the other hand, are typically based on random graphs, which ensure certain basic network properties such as connectivity or k-connectivity with some (high) probability. Clearly, protocols designed atop of such models can only provide probabilistic guarantees.

Given the wealth of protocol- and algorithm-related research that is based on such models, surprisingly little can be said about their validity in real-world networks (see Section II). In particular, an adversarial model, e.g., based on T-interval connectivity [2] or the eventual occurrence of certain strongly connected components [3][4] in the communication graphs immediately raises the question of whether these are reasonable assumptions in a real dynamic network or not.

There are two fundamentally different approaches for addressing this question: (i) Analytic or simulation-based *coverage analysis* and (ii) *monitoring* of real networks. Coverage analysis relies on a detailed low-level model of the underlying communication network, and verifies either analytically or via simulations whether the communication graphs generated by the underlying model exhibit the required properties with sufficiently high probability. The main disadvantages of this approach are the critical dependency on the appropriateness of the underlying network model (= inherent coverage) and its inability to incorporate engineering details like protocol stack implementations, etc.

Obviously, monitoring experiments do not suffer from these deficiencies. However, unless one is content with very limited information, a dedicated (and typically fairly complex) measurement infrastructure is required. Surprisingly, despite the substantial body of existing experimental research on various types of wireless networks, we were unable to find an existing infrastructure that facilitates long-term monitoring of topology-related properties in evolving communication graphs. In this paper, we present a suitable framework for comprehensive topology monitoring in wireless sensor networks that does not need a special infrastructure, and demonstrate its utility by validating a recently introduced adversarial network model [5].

Detailed Contributions: Please also refer to our technical report [6], since lacking space did not allow us to include all the findings in this paper.

(1) We present an overview of the features and the architecture of a framework for the long-term communication topology monitoring in synchronous dynamic networks, which has been implemented for Atmel RZ200 wireless sensor network motes running TinyOS. In a synchronous computation, one (conceptually) assumes that all nodes execute in a sequence of perfectly synchronized *rounds* $r = 1, 2, \ldots$, each consisting of (i) the broadcast of a message, (ii) the reception of all messages from the neighbors, and (iii) some local computation that also involves received messages. Our framework locally records the complete connectivity information for every node (i.e., the set of nodes from which an application message has been successfully received) in every round and disseminates this information multi-hop to one or more root nodes that act as data sinks for monitoring information. The root nodes forward this monitoring data, via a dedicated LAN, to a PC that fuses this data to construct the complete directed communication graph for every round. Postprocessing tools allow to query the recorded communication graphs in order to verify any desired graph property.

(2) We demonstrate the utility of our prototype implementation by means of the experimental evaluation of the coverage of an adversarial network model introduced in [5] in small-scale WSNs. Essentially, the model aims at dynamic networks

 ISBN: 978-1-61208-490-9

that may behave arbitrarily (even partition) during some finite initial period, after which the system remains reasonably well-connected sufficiently long. The network assumption $\Diamond\texttt{STABLE}(D)$ (see Definition 3) has a tunable parameter D, which is related to network stability. We evaluate the coverage of $\Diamond\texttt{STABLE}(D)$, i.e., the likelihood that it actually holds in every execution, in several WSN deployments that differ in node density, fatness of the deployment area, and interference level. Our results reveal a very good coverage in the considered settings, provided D is chosen appropriately.

Paper organization. We start with the description of the general features, architecture and operating principle of our framework in Section III. Section IV describes details of our implementation, Section V outlines the user interface, including some features of the currently available postprocessing tool. Section VI presents the purpose and results of our sample experiments. Some conclusions and directions of further work in Section VII complete our paper.

II. Overview of Related Work

Theoretical analyses based on random graphs, simulations, and measurements of real systems are all abundant in the existing literature, see, e.g., [7] for an overview.

Due to lacking space, we will restrict our attention to (a subset of) existing measurement approaches here; further references can be found in [6]. There are many open testbeds [8] , which provide a powerful infrastructure for dedicated experiments. Unfortunately, however, the (statistical) data provided in existing experimental evaluations typically address the properties of individual links [9] or system-wide properties like throughput [7][10] and other end-to-end performance characteristics [11]. By contrast, we are interested in detailed structural properties of not necessarily bidirectional communication graphs.

Existing approaches for experimentally exploring network topologies use active probing or passive monitoring, and may or may not require support from intermediate nodes. However, the inferable topology information is usually quite restricted, typically to network cardinality [12] or capacity [13]. Moreover, the topology of the underlying network is often limited. For example, the approach described in [14] uses the data correlation caused by intermediate network coding for inferring tree or DAG topologies. By contrast, [15] uses active probing with traceroute data, and primarily addresses problems caused by anonymous/non-cooperative intermediate nodes and the resulting uncertainties in topology inference. Pure network tomography approaches infer the network topology solely from data available at end nodes, typically using statistical approaches [16][17].

There is also a substantial body of work on connectivity monitoring in wireless sensor networks. Both active probing [18][19], where (a subset of) the network nodes query their neighborhood and forward connectivity data to some sink node, and passive techniques using data available at end-nodes only [20], as in network tomography approaches [21], can be employed here. Typical approaches using the latter assume that the WSN topology is a convergecast tree, where all nodes periodically send their data to a sink, using data aggregation. The topology is then reconstructed from the data received at the sink.

All these solutions provide, with varying accuracy, (part of) the entire topology. Moreover, they typically assume the existence of a bidirectional spanning tree for routing purposes. We are not aware of approaches that can infer sub-graph properties such as, for example, the presence of a rooted spanning tree or a strongly connected component, in sparsely connected communication graphs.

III. Framework Description

As already stated in Section I, the goal of our framework is to continuously monitor the evolution of the possibly sparsely connected, directed communication graph of a synchronous wireless sensor network over time. Per-node-recorded connectivity data is disseminated to certain special nodes ("root motes") that act as data sinks. The latter forward the data to a PC, where it can be analyzed and visualized.

In this section, we describe the general features, architecture and operating principle of our framework. Implementation-related details are provided in the subsequent section.

A. Required features

The design of our framework started out from several goals: (1) **Synchronous applications**: We target synchronous WSNs, where the WSN nodes (called *motes* in the sequel) execute a round-based algorithm (which requires an underlying time synchronization mechanism).

(2) **Long-term monitoring**: We need to monitor the evolution of the entire communication graph of a synchronous WSN over days and more (which rules out to store the complete monitoring data locally at the motes).

(3) **Standard motes**: We do not impose any dedicated monitoring hardware infrastructure at our motes (which requires monitoring data dissemination to use the wireless interface only).

(4) **Partitionable directed communication graphs**: We must allow the WSN to possibly partition, for an arbitrary time (which precludes the existence of an underlying spanning tree for routing the monitoring data to a single sink).

(5) **Message loss**: We cannot a priori guarantee reliable delivery of all messages containing monitoring data (which requires selective retransmission).

(6) **User-supplied application**: It must be possible to plug-in a user-supplied round-based application algorithm (which requires a message-passing interface that also allows to specify transmission scheduling policies and transmission powers).

(7) **Fault-injection**: It must be possible to exercise some control over the network topology, e.g., for testing purposes (which requires to actively inhibit the application-level communication between given pairs of sender and receiver motes).

 ISBN: 978-1-61208-490-9

B. System architecture

In order to meet the above requirements, our framework consists of four different components depicted in Figure 1. The central part is the *monitoring network* itself, which consists of the WSN *motes* that both execute the synchronous application and the (low-level) monitoring infrastructure. The collected connectivity data is disseminated to a *monitoring PC*, which collects and integrates the information from all the motes in order to reconstruct the communication graph for every round.

This data dissemination is actually a two-step process: First, the per-mote recorded data is disseminated via multi-hop communication to some special *root mote*, which acts both as the primary data sink and as a root for time synchronization via the *Flooding Time Synchronization Protocol* (FTSP) [22] of the motes. For WSNs that may partition, our framework supports multiple root motes. Every root mote is serially connected to a dedicated *forwarding PC* component, which finally forwards all the data received from the serial interface to the monitoring PC component via LAN and vice versa. In other words, root motes and forwarding PC component act as "WSN-UART" and "UART-LAN" gateways, respectively.

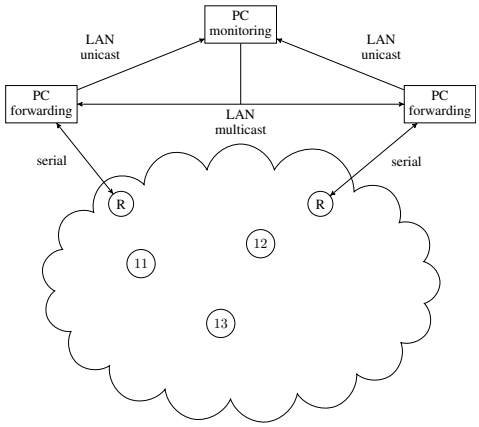

Figure 1: Overview of the framework's general structure. The framework consists of monitoring Motes (numbered), Root Motes (labeled with "R"), forwarding PC components and a monitoring PC instance.

In more detail, the components perform the following tasks:

Motes: The motes execute an user-supplied synchronous distributed algorithm that makes up the WSN application software. It has to use a single-hop broadcast to send an *application message* to all other motes in the WSN in every round. The monitoring infrastructure on every mote records from whom it successfully received a message in a round, and disseminates this data by broadcasting *report messages* via a suitable data collection protocol (described in Section IV-A) subsequently.

Root motes and forwarding PC: Each root mote is connected to an instance of the forwarding PC component and works as a WSN-UART-Gateway: Every report messages received from a mote via the wireless interface is forwarded via the serial interface. Vice versa, all control messages received from the forwarding PC component are sent to the motes using the radio interface.

In addition, the root motes are also root nodes for time synchronization via FTSP [22]. A detailed description of the time synchronization mechanism is given in Section IV-B.

Monitoring PC: Each instance of the forwarding PC component is connected to a single dedicated monitoring PC via a (wireless) Ethernet connection. Besides exercising all the framework setup and monitoring control tasks, it primarily gathers and integrates the per-node connectivity data contained in forwarded report messages and stores it in the file system for post-processing. Since report messages may be lost before they reach any root mote, the monitoring PC also checks the received connectivity information for missing data and actively requests retransmission of single report messages if needed. A detailed description of this messaging protocol can be found in Section IV-A.

C. General operating principle

Thanks to the time synchronization mechanism described in Section IV-B, our monitoring framework operates in a repeated sequence of three consecutive lockstep-synchronous *phases* sketched in Figure 2:

Phase 1 — record per-mote connectivity information: At the beginning of this phase, which actually represents a single round of the application algorithm, every mote broadcasts an application message generated by the user-supplied algorithm. It is sent via the single-hop broadcast service provided by our framework. To restrict the heavy mutual interference caused by simultaneous broadcasting of all motes, a suitable transmission schedule (+ power control) can be applied here.

During Phase 1, each node records the sender of every successfully received message (for the current round). The duration of Phase 1 must be chosen appropriately to ensure that every receiver can indeed receive and process the message from every sender. Thus, the set of motes a message has been received from by the end of Phase 1 reflects the in-edges of the communication graph ending at the receiver mote. Figure 2 (left) shows an example.

Phase 2 — disseminate collected data: When Phase 1 ends, every mote sends a *report message* containing its connectivity data, i.e., the set of motes it received a message from, to one or more root motes. Figure 2 (middle) shows an example.

Actually, since the diameter of the WSN may be large, a custom *multi-hop data collection* protocol is used for this purpose. As described in detail in Section IV-A, it uses a combination of flooding and local caching, in conjunction with a suitably long duration of Phase 2. Moreover, in order to circumvent the collisions resulting from simultaneous broadcasting of report messages, a suitable transmission schedule is used at the beginning of Phase 2.

Phase 3 — request and retransmission of missing report messages: When Phase 2 has terminated, the monitoring PC checks the received report messages for completeness and, if needed, sends messages requesting the retransmission of lost messages. Note that request messages are sent by the monitoring PC sequentially, so no transmission scheduling is necessary here. These *request messages* are in fact disseminated via a custom *multi-hop messaging* protocol (also described in Section IV-A), which uses the local caching of report messages to possibly speed-up the response time: Any

 ISBN: 978-1-61208-490-9

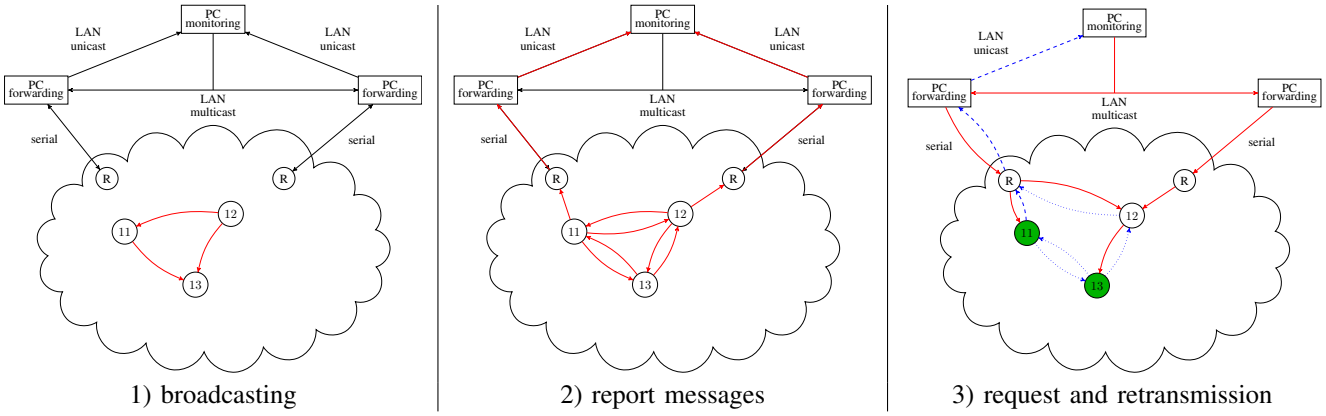

Figure 2: Our framework works in three consecutive lockstep-synchronous phases. Phase 1 (left): Recording of connectivity data. Phase 2 (middle): Disseminate connectivity data to monitoring PC. Phase 3 (right): request retransmission of missing data.

mote that has the required report message in its cache can answer the request. An example is shown in Figure 2 (right).

IV. Framework Implementation

In this section, we provide an overview of the prototype implementation of our framework, which has been developed for Atmel RZ200 motes running TinyOS. In particular, we elaborate on our custom multihop protocols, on time synchronization, and on the fault-injection capabilities provided by our implementation.

A. Multihop protocols

As already mentioned in Section III-C, our framework must implement two multi-hop data dissemination protocols for monitoring data:

(i) **Multi-hop data collection**: In Phase 2, each node sends a report message containing its connectivity data to the monitoring PC.
(ii) **Multi-hop messaging**: Since report messages may be lost in (i), the monitoring PC checks the received data for completeness and, if needed, requests the retransmission of missing report messages.

Existing routing and data collection protocols, such as the *Dynamic Manet On-demand Protocol* [23] (creating a unicast route) and the *Collection Tree Protocol* [24] (collecting data at a dedicated root node) require communication graphs that contain a reasonably stable spanning tree with bidirectional links between neighbors. Moreover, they need some sort of routing table and are hence expensive regarding RAM-usage. Since we could not afford this assumption due to our possibly sparsely connected, directed communication graphs, we had to develop a custom flooding protocol, augmented with local caching, as the basis of our multi-hop protocols. Our flooding protocol relies on several low-level facilities, which we will describe next.

Data encoding. Since our framework shall support large networks, despite small message sizes, a compact encoding of our monitoring data is needed. We implemented this by using a single bit to identify each node: Each node's ID corresponds to a single bit within an array holding the connectivity data recorded by a node, the so-called *monitor array*. If some bit is zero at the end of Phase 1, no application message has been received from the corresponding node in this round.

Message headers. Since flooding protocols are prone to sending messages to an excessively large number of nodes, possibly even resulting in cycles, we use a message header containing the following data: Besides *source* and *destination address*, it contains a unique *message ID* (sequence number) for each source address. Note that the pair of source address and message ID unambiguously identifies every message. To limit the spreading of a message, a *hop count* field, working as a time-to-live counter, is used.

Local caches. Every mote is equipped with two (small) caches, a *report-cache* (holding report messages) and a *header-cache* (holding headers of messages that are sent via the flooding protocol). Each is organized as a FIFO-buffer augmented with reordering abilities to minimize RAM-usage: A message/-header to be added is always appended to the tail of the FIFO-buffer. If a copy of the newly added message/header is already present in the buffer, the copy is deleted first. If the buffer is (still) full, the message/header at its head is deleted first. Obviously, these rules guarantee that (i) messages/headers cached earlier are dropped before later ones, and (ii) that the same message/header is never cached more than once.

Our custom **flooding protocol**, a variant of which is used both in the multi-hop data collection and in the multi-hop messaging protocol, combines the above low-level facilities to avoid cyclic sending of messages. The primary mechanism employed for this purpose is the usage of the unique message header combined with the local header-caches: First, the originator of a report or request message to be flooded broadcasts the message, with an appropriately initialized message header. Each time a node sends or forwards a message, it caches the message's header in the header-cache. Before a message is forwarded, the node checks whether the (unique!) message's header is already stored in cache. If so, the message is dropped, otherwise it is indeed forwarded by broadcasting it. Figure 3 depicts an exemplary scenario.

We can now describe the operation of our two multi-hop protocols:

 ISBN: 978-1-61208-490-9

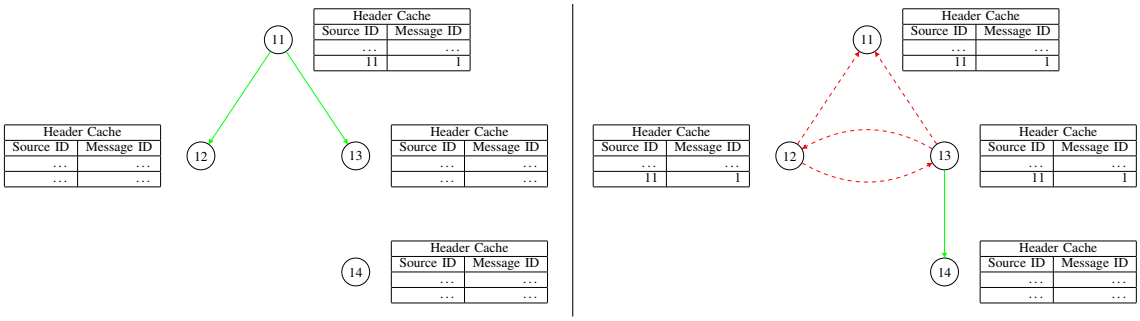

Figure 3: Our custom flooding protocol uses a header-cache at each node to avoid cyclic sending. As nodes 11 and 12 already sent the message at the left, their header-caches contain the message (11, 1). Thus they discard the message (red, dashed arrows); only node 14 accepts it (green, solid arrow).

(i) **Multi-hop data collection**: Recall that, in Phase 2, each node sends a *report message* containing its monitor array to the monitoring PC by means of this protocol. To accomplish this, the node uses a variant of our flooding protocol to reach at least one root mote: It addition to the above basic mechanism for avoiding cyclic message sending, it caches every sent or forwarded report message in a dedicated *report-cache* in order to support retransmissions (see next item).

(ii) **Multi-hop messaging**: Recall that, at the beginning of Phase 3, the monitoring PC checks the received monitoring data for possibly missing report messages. To initiate the retransmission of such missing report messages, the monitoring PC uses this protocol. It requests every root mote to disseminate the same *request message* using another variant of our flooding protocol: When receiving a request message, a node also searches its report-cache for the requested report message. If the message is found, the report message is re-sent via the multi-hop data collection protocol. If the report message is not found in the report-cache, the request message is forwarded following the standard rules to avoid cyclic sending.

Extensive tests, also using the fault-injection capabilities presented in Section IV-C, revealed that these protocols implement a very robust, fast and reasonably communication-efficient approach for convergecasting information from all nodes in the WSN to the monitoring PC, provided the relevant parameters, namely, phase durations, transmission staggering time and initial hop count, are adequately chosen.

B. Time synchronization

As mentioned in Section III-C, our round-based synchronous setting makes it mandatory to implement a common global time at all motes. In our framework, global time is defined by a 32 bit timestamp with a granularity of 1 millisecond. Every round has fixed duration R, called *round time*, measured in milliseconds; round $k \geqslant 0$ is hence started at global time $t^k_{start} = kR$ at every mote.

Due to the heterogeneous system architecture of our framework (recall Figure 1), the time synchronization mechanism consists of multiple parts:

(1) **Synchronizing the PCs**: As the forwarding and monitoring PCs are interconnected via Ethernet, we use the standard *Network Time Protocol* (NTP) to synchronize those.

(2) **Synchronizing the motes**: Since the motes do not understand NTP, they are using the *Flooding Time Synchronization Protocol* [22] (FTSP). Like NTP, it achieves a typical synchronization precision in the one millisecond range. FTSP uses a single master mote (called the *FTSP-root*) as the primary time-source for all other (reachable) motes.

(3) **Synchronizing the root motes to the forwarding PCs** As we are using two different protocols for time synchronization, we had to find a solution where both protocols are synchronizing to the same global time. As root motes do not execute the WSN application but just act as data sinks, our implementation (i) synchronizes every root mote to NTP time and (ii) forces every FTSP-root to be one of the root motes.

Since the junction between NTP and FTSP is the USB link between the pairs of forwarding PC and associated root mote, (i) is easily achieved by letting each forwarding PC periodically send its 32 bit NTP-timestamp to its connected root mote. If such a root mote is also working as an FTSP-root, it calculates the difference between NTP-time and FTSP global time and, if non-zero, adjusts the latter accordingly. All the other motes will hence effectively receive NTP-time from the FTSP-root and synchronize themselves to this time via FTSP.

In order to also achieve (ii), all root motes are configured with *unique node identifiers* (UIDs) *less* than the UIDs or ordinary motes. Since the FTSP-root is elected by all reachable motes dynamically as the mote with the minimum ID, this ensures that FTSP-roots will always be root motes (unless none of those was reachable, of course).

C. Fault-injection capabilities

To both facilitate testing of our implementation and advanced fault-injection experiments, we implemented a simple feature for dynamically enabling/disabling individual directed links between pairs of motes.

Each mote maintains a per-mote *blacklist array* for this purpose. Similar to the monitor array presented in Section IV-A, each bit in this array corresponds to a single sender mote: At the reception of a message, the mote checks whether the bit corresponding to the sender is set; if this is the case, the message is discarded and neither reported as having been received nor delivered to the application.

 ISBN: 978-1-61208-490-9

The monitoring PC maintains a blacklist array for each mote. If it is instructed by the user, via the experimental control user interface described in Section V-B, to enable resp. disable the link from x to y, it sets resp. clears the bit corresponding to x in the blacklist array dedicated for node y. It then sends a *fault injection message* holding the (updated) monitoring PC's copy of node y's blacklist array to node y, which is then updating its own copy. Note that the fault injection messages are also sent by means of our flooding protocol, following the rules to avoid cyclic sending.

V. Application Interface and Postprocessing

In this section, we provide a glimpse of how to use our framework: We survey some features that need to be configured prior to compilation, describe how to interact with the running system via the user interface provided by the monitoring PC, and list some features of our postprocessing tool.

A. System configuration

The first step setting up an experiment, is to choose certain system-wide parameters (via configuration C-Macros) before compiling and flashing the software to the motes (and forwarding PCs). Among these are upper bounds on the number of (root) motes and cache sizes, which are of course limited by the typically low RAM size.

The parameters for our flooding protocols are particularly important w.r.t. data completeness. Some of these are the initial hop count, durations of the Phases and the transmission staggering time for Phase 2. The latter also determine the substantial monitoring time overhead (i.e., the slow-down of the application's execution). Our postprocessing tool provides some meta-data that can be used to validate the chosen parameters, see Section V-C.

As mentioned before, it is possible to apply a user-supplied round-based algorithm on top of the Phase 1. Our framework hence provides an interface, which comprises functions for broadcasting/receiving messages and a *start-round* event, signalling the start of the current round and delivering the set of messages that have been received in the current round. Our interface also supplies the application with *signal-to-interference-plus-noise* (SINR) information here, However, in our prototype implementation, this data is void since the RZ200 motes do not support this feature. In response to the start-round event, the applied algorithm may compose the content of the next round's application message, as well as (optionally) select the transmission power and a staggering time for transmission scheduling. The actual broadcast is then handled, at the appropriate time, by our framwork.

B. Interactive setup and control

The user can control the experiments by means of a simple user interface provided by the monitoring PC, which reads and processes a number of commands from *stdin*:

- *start*: Once the start command has been read from *stdin*, a *start message* is flooded that causes the motes to (synchronously) start their monitoring activity.

```
1  start
2  block 11,13
3  wait 60000
4  free 11,13
```

Figure 4: Exemplary test script, which (i) starts monitoring, (ii) blocks the connections from 11 to 13, and (iii) frees this blocking after 10 minutes.

- *block* x, y resp. *free* x, y: These commands address the motes' fault-injection capabilities and block resp. free the application messages from node x to node y.
- *wait* x: Wait x milliseconds until the next line is read from *stdin*. This command allows the creation of test scripts: Using a textfile, one can easily block and free connections automatically at predefined times (which must of course be reflected in the monitoring data recorded). An exemplary script can be found in Figure 4.

C. Postprocessing tools

Once the connectivity data is stored (in a textfile) in the file system of the monitoring PC, one can apply arbitrary tools for postprocessing the collected data. The main feature of our custom postprocessing tool, which has been developed (in HTML/PHP) with answering the validation question of Section VI in mind, is to compute the sets of motes in *strongly connected components* (SCCs), using *Tarjan's* algorithm [25]. Recall that a strongly connected component is a component where every node is reachable from every other node.

Once the data has been processed one may use various features of our tool.

- A CSV-file holding the adjancency matrices for each round can be created.
- For any round, one can visualize the connectivity graph, as well as its contraction to SCCs.
- For any round and every mote, the FTSP-root's ID used for time synchronization and the root mote that delivered the report message to the monitoring PC, along with the message's hop count, can be printed. This data allows to detect inadequate hop count settings and/or needs for a revised placement of root motes, thus can be used to validate the configuration.
- As will be detailed in Section VI-A, our main interest lies in the stability of SCCs consisting of the same set of motes, i.e., the number of rounds that they persist. For this purpose, all the SCCs existing in a recorded data file can be listed over time, along with the first and the last round and its duration.

VI. A Case Study: Validating an Adversarial Network Model

In this section, we will provide the results of an experimental validation of the adversarial model introduced in [5] using our framework.

A. Validation question

The adversarial model of [5] assumes that every sequence of directed communication graphs $\mathcal{G}^1, \mathcal{G}^2, \ldots$ present in round

$1, 2, \ldots$ satisfies the graph properties summarized in Definition 3. We will now introduce the core concepts needed for defining those. Due to space constraints, we will replace some formal definitions by informal ones where possible.

Given the communication graph $\mathcal{G}$ of a round, a *root component* R is the set of nodes of a strongly connected component $\mathcal{R}$ in $\mathcal{G}$ without in-edges from nodes outside $\mathcal{R}$. Note that every graph has at least one root component. A root component R that exists in every graph of a (not necessarily consecutive) graph sequence $(\mathcal{G}^r)_{r=a}^{a+d-1} = \mathcal{G}^a, \ldots, \mathcal{G}^{a+d-1}$ is called a *common root*. Note carefully that the *interconnect topology* of the nodes in a common root R may be different in every $\mathcal{G}^i$. The term vertex-stable root component has hence been coined for common roots in [3][4].

One can show that a root component that is common in a sufficiently long graph sequence $\mathcal{G}^a, \ldots, \mathcal{G}^{a+d-1}$ guarantees multi-hop communication between *all* nodes in R. Moreover, if R happens to be the only root, termed *single root*, information from every node in R reaches all nodes in the entire network. This is captured by the following definition:

Definition 1 (Dynamic diameter D): A network has dynamic (network) diameter D, if for every graph sequence that contains a subsequence $\mathcal{G}^{r_1}, \ldots, \mathcal{G}^{r_D}$ of D not necessarily consecutive R-single-rooted communication graphs, it holds that information from every node in R from (the end of) round $r_1 - 1$ reaches all nodes in the network by (the end of) round r_D.

The following definition captures "the" central graph property used in [5]. It requires that a root that is common in a sequence of at least $D + 1$ rounds is single in a consecutive subsequence of at least $D + 1$ rounds:

Definition 2: We say that a graph sequence $(\mathcal{G}^r)_{r=\alpha}^{\alpha+d}$ has an $\texttt{ECS}(D+1)$*-common root* ("embedded $D + 1$-consecutive single common root") R, if (i) $(\mathcal{G}^r)_{r=\alpha}^{\alpha+d}$ has a common root R and (ii) $(\mathcal{G}^r)_{r=\alpha'}^{\alpha'+D} \subseteq (\mathcal{G}^r)_{r=\alpha}^{\alpha+d}$ has a single root R.

The graph property $\Diamond\texttt{STABLE}(D)$ to be validated by our experiments is the following, see also [5, Def. 12]:

Definition 3 (Message adversary $\Diamond\texttt{STABLE}(D)$): In every graph sequence $\mathcal{G}^1, \mathcal{G}^2, \ldots$ present in round $1, 2, \ldots$, the conjunction of the following three properties must hold:

(i) The first root component R that is common for at least $D+1$ consecutive rounds is a $\texttt{ECS}(D+1)$-common root.

(ii) At least one $\texttt{ECS}(D+1)$-common root R' (possibly $R' \neq R$) occurs eventually, which re-appears as a single root in at least D not necessarily consecutive later rounds.

(iii) The dynamic diameter is D.

Our validation experiments evaluate, for every deployment and every $D = 1, 2, 3, \ldots$, the coverage of $\Diamond\texttt{STABLE}(D)$ and the statistics of two important stabilization time parameters. The (experimental) *coverage* $\mathrm{Cov}(D)$ (abbreviated Cov if D is clear from the context) of $\Diamond\texttt{STABLE}(D)$ is defined as the number of testruns where $\Diamond\texttt{STABLE}(D)$ holds over the number of all testruns. Note that this coverage definition is conservative, as it also counts testruns as failed where $\Diamond\texttt{STABLE}(D)$ could have been satisfied eventually if the testrun had been continued (we very rarely encountered this situation for $D < 10$ in our experiments, though).

Given a testrun, let the *stabilization time* $r_{\text{sr}}(D)$ be the round where the first $\texttt{ECS}(D + 1)$-common root R starts to become single (Def. 3.(i)); r_{sr} delimits the end of the initial ("chaotic") period of system operation. Similarly, let $r_{\text{fi}}(D)$ be the round where the D-th single occurrence of R' (Def. 3.(ii)) happens; r_{fi} gives the (earliest) termination time of any consensus algorithm [5] in this testrun. In addition to Cov, we will also provide the averages of r_{sr} and r_{fi}, $\mathrm{Avg}\, r_{\text{sr}}$ resp. $\mathrm{Avg}\, r_{\text{fi}}$, in all testruns where $\Diamond\texttt{STABLE}(D)$, for some given D, holds.

B. Experimental Setup

To validate the coverage of the graph property given in Definition 3 experimentally, we set up four different scenarios and monitored the connectivity over time in multiple testruns. Our different scenarios were obtained by varying two main parameters, namely, deployment area and the transmission scheduling, in a WSN consisting of 20 motes.

Deployment Area. For Deployment 1, we used the rooms of our institute, where there are many obstacles and walls between the single motes and substantial interference due to WiFi accesspoints, etc. The node density is relatively high and the expected area of good radio coverage is reasonably fat. By contrast, for Deployment 2, we spread the same number of motes (more or less) in line of sight of each other on the rooftop of our building, where the interference level is considerably lower. The resulting area of good radio coverage is less fat and less dense (only 1/3 of Deployment 1). In both of our deployments, the resulting network diameter turned out to be in the range of $[2, 7]$.

Transmission Scheduling. If all the motes send their application message simultaneously at the beginning of the round, the SINR at every receiver is quite low, whereas a proper transmission scheduling where every mote has its unique time slot for transmission results in a much better SINR. Transmission scheduling is hence crucial for the network's connectivity. Therefore, we decided to use transmission scheduling as our experiments' second parameter: n ("no") means no transmission scheduling, t means transmission staggering with a dedicated 20 millisecond slot for every mote.

When we subsequently write Scenario 1n, for example, we mean Deployment 1 without transmission scheduling.

System Settings and Configuration. In all our experiments, we used a single root mote placed in the network's center. The header-cache was chosen to store up to 64 message headers, while the report-cache was able to hold up to 128 report messages, sent via our multi-hop protocol that used an initial hop count of 5. The monitoring PC requested the retransmission of missing report messages during 10 phases following the original round before it gave up.

C. Validation Experiments

Scenario 1t: Institute using transmission staggering. As transmission staggering leads to (more or less) stable reception conditions for each mote's application message in each round, the topology did not change much during the testruns. As a

result, we observed relatively long sequences where a single common root exists in each of our twelve testruns, which took from 89 to 224 application rounds.

Results. As shown in Table I (left column), for $D \in [4, 17]$, $\mathrm{Cov}(D) = 100\%$ since $\Diamond\texttt{STABLE}(D)$ held in each of our testruns. Depending on D, $\mathrm{Avg}\, r_{\mathrm{sr}}$ ranges from 4 and 48 and $\mathrm{Avg}\, r_{\mathrm{fi}}$ ranges from 12 to 215.

Scenario 2t: Rooftop using transmission staggering. As in Scenario 1t, the use of transmission staggering led to relatively long sequences of single common roots. We conducted twelve testruns that took from 24 up to 231 application rounds.

Results. As shown in Table I (right column), $\mathrm{Cov}(D) = 100\%$ for any $D \in [4, 8]$. In those testruns, $\mathrm{Avg}\, r_{\mathrm{sr}}$, resp. $\mathrm{Avg}\, r_{\mathrm{fi}}$, ranges from 9 to 69, resp. from 15 to 181, depending on D.

TABLE I: EVALUATION RESULTS FOR SCENARIOS 1t AND 2t.

Scenario 1t			
D	Cov	Avg r_{sr}	Avg r_{fi}
3	0.667	4	12
4	1.000	4	13
...			
17	1.000	23	59
18	0.917	25	62
19	0.833	29	68
...			

Scenario 2t			
D	Cov	Avg r_{sr}	Avg r_{fi}
3	0.500	9	15
4	1.000	14	23
...			
8	1.000	24	42
9	0.917	34	54
10	0.750	34	55
...			

Scenario 1n: Institute without transmission staggering. As expected, letting all motes send their application messages (almost) simultaneously causes a much higher variability of the communication topology over time. As a consequence, we observed much shorter periods of rounds were a common root exists in our twelve testruns, which took from 89 to 303 application rounds.

Results. As shown in Table II (left column), $\mathrm{Cov}(D) = 66.67\%$ for $D = 4$. $\mathrm{Avg}\, r_{\mathrm{sr}}$ is tendentially increasing from 12 up to 135, $\mathrm{Avg}\, r_{\mathrm{fi}}$ is within a range of $[19, 191]$. We encountered 4 testruns, where $\Diamond\texttt{STABLE}(D)$ was neither satisfied nor violated for any value of $D \geqslant 4$. As already hinted in the definition of $\mathrm{Cov}(D)$, $\Diamond\texttt{STABLE}(D)$ might have been satisfied eventually, for these values of D, if these testruns had been continued. Not counting these testruns, the resulting coverage $\mathrm{Cov}(D) = 100\%$ for $D = 4$.

Scenario 2n: Rooftop without transmission staggering. As in Scenario 1n only short periods with a common root component existed. For this scenario, we conducted twelve testruns comprising between 75 and 266 application rounds.

Results. As shown in Table II (right column), $\mathrm{Cov}(4) = 83.33\%$ with $\mathrm{Avg}\, r_{\mathrm{sr}} = 48$ and $\mathrm{Avg}\, r_{\mathrm{fi}} = 60$. As in Scenario 1n, we encountered three testruns where $\Diamond\texttt{STABLE}(D)$ was neither satisfied nor violated for any value of $D \geqslant 4$. Not counting these testruns, $\mathrm{Cov}(4)$ is again 100%.

D. Discussion

Comparing the results of the four scenarios presented in Section VI-C leads to a number of interesting insights.

Most importantly, we observed for no testrun a violation of properties (i) and (iii) of $\Diamond\texttt{STABLE}(D)$ for any choice of $D \geqslant 4$. For every individual scenario where staggering was

TABLE II: EVALUATION RESULTS FOR SCENARIOS 1n AND 2n.

Scenario 1n			
D	Cov	Avg r_{sr}	Avg r_{fi}
3	0.250	12	19
4	0.667	13	22
5	0.583	13	23
...			
9	0.583	61	81

Scenario 2n			
D	Cov	Avg r_{sr}	Avg r_{fi}
3	0.083	13	22
4	0.833	47	60
5	0.500	65	81
6	0.417	65	85
7	0.167	75	97

turned on, there is also a range for D that results in 100% experimental coverage of all properties of $\Diamond\texttt{STABLE}(D)$ (see below) for any fixed value of D taken from this range. In the scenarios where no transmission staggering was used, we encountered some testruns where $\Diamond\texttt{STABLE}(D)$ might still have been satisfied eventually for $D \geqslant 4$. In those testruns where we could validate $\Diamond\texttt{STABLE}(D)$, $D = 4$ resulted in 100% coverage.

For each testrun $\Diamond\texttt{STABLE}(2)$ was violated On the other hand, there was no testrun that violated condition (i) or (iii) of $\Diamond\texttt{STABLE}(D)$ for any $D \geqslant 4$.

Figure 5 shows how D influences the coverage of $\Diamond\texttt{STABLE}(D)$ for Deployment 1 and 2, irrespectively of the transmission scheduling. Recall that the area covered by the former is only about one third of the latter, fatter, and experiences more background interference. Interestingly, while the coverage of $\Diamond\texttt{STABLE}(D)$ in Deployment 1 is better for (nearly) all values of D, for $D = 4$ the coverage is higher at Deployment 2.

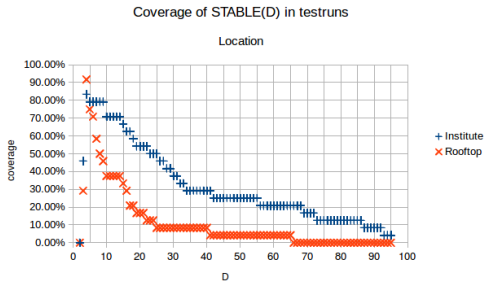

Figure 5: Coverage depending on deployment area

Figure 6 confirms the (expected) strong influence of transmission staggering on $\mathrm{Cov}(D)$, irrespectively of the deployment. It is apparent that the coverage of $\Diamond\texttt{STABLE}(D)$ is uniformly much higher with transmission staggering than without. $\mathrm{Cov}(D) = 100\%$ for $D \in [3, 8]$ for the former, while it does not achieve 100% for any value of D in the latter case.

Overall, we can conclude that the adversarial model proposed in [5] has a very good coverage in all our deployments, provided D is appropriately chosen. In more detail:

- $\Diamond\texttt{STABLE}(D)$ very likely holds for some D in the range of the actual average per-round network diameter. For smaller values of D, the coverage of $\Diamond\texttt{STABLE}(D)$ drops.
- Since in none of our testruns properties (i) and (iii) of $\Diamond\texttt{STABLE}(D)$ were violated for $D \geqslant 4$, it stands to reason that the fulfillment of $\Diamond\texttt{STABLE}(D)$ for even larger D may be only a matter of time.

 ISBN: 978-1-61208-490-9

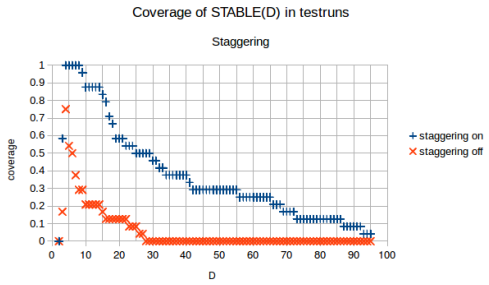

Figure 6: Coverage depending on transmission staggering

- A stable $\texttt{ECS}(D+1)$ root does not need to exist from the very beginning of the network's life. In fact, our observations confirm the hypothesis of [5] that one has to account for an initial "chaos-period" and that the network only eventually becomes reasonably stable.

VII. Conclusion

We provided an overview of the system architecture and the internal workings of a framework for long-term monitoring of the communication topology of synchronous wireless sensor networks consisting of memory-constrained wireless motes. We discussed and solved various issues, such as finding a suitable flooding protocol, adequate synchronization, data collection, and post-processing. Finally, we employed our framework in order to validate a network property introduced in [5] and found that it has a reasonable assumption coverage.

Part of our future work in this area will be devoted to decreasing the time overhead caused by our monitoring framework, to replace statically configured system parameters by on-line ones, and to adapt/scale-up the framework to other testbeds. In addition, we are working on improving the multi-hop protocols used in our framework and establish network conditions, which are sufficient for guaranteeing monitoring data completeness.

Acknowledgement

This work has been supported the Austrian Science Fund (FWF) projects P28182 (ADynNet) and S11405 (RiSE).

References

[1] F. Kuhn and R. Oshman, "Dynamic networks: Models and algorithms," *SIGACT News*, vol. 42(1), pp. 82–96, 2011.

[2] F. Kuhn, N. A. Lynch, and R. Oshman, "Distributed computation in dynamic networks," in *STOC*, pp. 513–522, 2010.

[3] M. Biely, P. Robinson, and U. Schmid, "Agreement in directed dynamic networks," in *Proceedings 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO'12)*, LNCS 7355, pp. 73–84, Springer-Verlag, 2012.

[4] M. Biely, P. Robinson, U. Schmid, M. Schwar z, and K. Winkler, "Gracefully degrading consensus and k-set agreement in directed dynamic networks," in *Revised selected papers Third International Conference on Networked Systems (NETYS'15)*, Springer LNCS 9466, (Agadir, Morocco), pp. 109–124, Springer International Publishing, 2015.

[5] M. Schwarz, K. Winkler, and U. Schmid, "Fast consensus under eventually stabilizing message adversaries," in *Proceedings of the 17th International Conference on Distributed Computing and Networking*, ICDCN '16, (New York, NY, USA), pp. 7:1–7:10, ACM, 2016.

[6] D. Pfleger and U. Schmid, "A framework for connectivity monitoring in wireless sensor networks," Research Report TUW-241107, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-2, 1040 Vienna, Austria, 2015. http://publik.tuwien.ac.at/files/PubDat_241107.pdf [retrieved: June, 2016].

[7] C. Newport, D. Kotz, Y. Yuan, R. S. Gray, J. Liu, and C. Elliott, "Experimental Evaluation of Wireless Simulation Assumptions," *SIMULATION: Transactions of The Society for Modeling and Simulation International*, vol. 83, pp. 643–661, Sept. 2007.

[8] A.-S. Tonneau, N. Mitton, and J. Vandaele, "A survey on (mobile) wireless sensor network experimentation testbeds," in *Proceedings of the 2014 IEEE International Conference on Distributed Computing in Sensor Systems*, DCOSS '14, (Washington, DC, USA), pp. 263–268, IEEE Computer Society, 2014.

[9] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis, "An empirical study of low-power wireless," *ACM Trans. Sen. Netw.*, vol. 6, pp. 16:1–16:49, Mar. 2010.

[10] A. Cerpa, J. L. Wong, M. Potkonjak, and D. Estrin, "Temporal properties of low power wireless links: Modeling and implications on multi-hop routing," in *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '05, (New York, NY, USA), pp. 414–425, ACM, 2005.

[11] L. Mottola, G. P. Picco, M. Ceriotti, c. Gună, and A. L. Murphy, "Not all wireless sensor networks are created equal: A comparative study on tunnels," *ACM Trans. Sen. Netw.*, vol. 7, pp. 15:1–15:33, Sept. 2010.

[12] H. B. Acharya and M. G. Gouda, "On the hardness of topology inference," in *ICDCN*, pp. 251–262, 2011.

[13] A. Bestavros, J. W. Byers, and K. A. Harfoush, "Inference and labeling of metric-induced network topologies," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 11, pp. 1053–1065, 2005.

[14] P. Sattari, C. Fragouli, and A. Markopoulou, "Active topology inference using network coding," *Physical Communication*, vol. 6, pp. 142–163, 2013.

[15] Y. A. Pignolet, S. Schmid, and G. Trédan, "Misleading stars: what cannot be measured in the internet?," *Distributed Computing*, vol. 26, no. 4, pp. 209–222, 2013.

[16] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu, "Network tomography: Recent developments," *Statistical Science*, vol. 19, pp. 499–517, 08 2004.

[17] M. Coates, A. O. Hero III, R. Nowak, and B. Yu, "Internet tomography," *Signal Processing Magazine, IEEE*, vol. 19, no. 3, pp. 47–65, 2002.

[18] B. Deb, S. Bhatnagar, and B. Nath, "Stream: Sensor topology retrieval at multiple resolutions," *Telecommunication Systems*, vol. 26, no. 2-4, pp. 285–320, 2004.

[19] M. Zhang, M. C. Chan, and A. L. Ananda, "Connectivity monitoring in wireless sensor networks," *Pervasive and Mobile Computing*, vol. 6, no. 1, pp. 112–127, 2010.

[20] Y. Liu, K. Liu, and M. Li, "Passive diagnosis for wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 18, pp. 1132–1144, Aug. 2010.

[21] M. Keller, J. Beutel, and L. Thiele, "How was your journey?: Uncovering routing dynamics in deployed sensor networks with multi-hop network tomography," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, (New York, NY, USA), pp. 15–28, ACM, 2012.

[22] M. Maròti, B. Kusy, G. Simon, and A. Lèdeczi, *The Flooding Time Synchronization Protocol*. Verderbilt University, Institute for Software Integrated Systems, Nov. 2004.

[23] R. Thouvenin, "Implementing and evaluating the dynamic manet on-demand protocol in wireless sensor networks," master's thesis, University of Aarhus, Department of Computer Science, 2007.

[24] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," SenSys, pp. 1–14, 2009.

[25] R. Tarjan, "Depth-first search and linear graph algorithms," in *SIAM J. Computation*, vol. 1, pp. 114–121, Jan. 1972.

 ISBN: 978-1-61208-490-9