EvAnT: Analysis and Checking of event traces for Wireless Sensor Networks

Matthias Woehrle [#], Christian Plessl [*], Roman Lim [#], Jan Beutel [#], Lothar Thiele [#]

[#] *ETH Zurich, Computer Engineering and Networks Lab*
8092 Zurich, Switzerland
matthias.woehrle@tik.ee.ethz.ch

[*] *University of Paderborn, Paderborn Center for Parallel Computing*
33102 Paderborn, Germany

Abstract

Testing and verification methodologies for Wireless Sensor Networks (WSN) systems in pre-deployment are vital for a successful deployment. Increased visibility of the internal state of a WSN application is established by instrumenting the application for logging execution traces at runtime. While the interpretation of the event traces is application-specific, a common method for analysis can be devised. This method should allow for a concise formulation of explorative queries to determine the occurrence and the cause of functional or performance problems.

The contribution of this paper is an event analysis methodology that is implemented in the EvAnT framework. EvAnT allows for specifying queries that are executed on the collected traces. EvAnT is specifically tailored to WSN testing and debugging. We demonstrate the applicability of EvAnT by a case study in a building monitoring project.

I. Introduction

Wireless Sensor networks are wireless networked embedded sensing systems allowing to monitor an environment in previously unprecedented ways. The close and continuous observation of a phenomenon with these tiny sensing devices allows for many novel and widely differing application areas such as fire detection alarm systems or monitoring of the environment or structures such as a building.

However, numerous failed or under-performing deployments of WSNs have shown that their design is extremely intricate. Causes for these fails differ: some can be attributed to the embedded nature of WSNs and the impact of the environment on the system, some can be attributed to software failures and simplified assumptions in protocol design, while causes for other system fails remain unknown [1].

Deployments in remote and harsh environments such as a volcano [2] prohibit debugging at the installation site or extensive field tests [3], necessitating an error-free system at deployment time. Systematic WSN verification is indispensable to arrive at a correct system. Testing in pre-deployment allows for exposing errors in the system, that may cause expensive re-deployments [3] or non-performing systems [4].

Pre-deployment testing and debugging has the major advantage that execution information may be increased by instrumenting individual sensor nodes. Using test platforms such as a testbed or a simulator, which allow for collecting the instrumentation data, a rich set of data may be used for failure detection and debugging. Accuracy, the scale of the network and the intrusiveness of the monitoring can be customized to the individual application and test requirements. Data collected in one or many test runs can be used to analyze functional and non-functional (performance) properties in detail.

Analysis on an application level is straight forward. Just consider the example of packet yield in the predominant data collection applications in WSNs: Count the transmissions on each node and the received packets on the sink node and compute the ratio of arrived versus sent nodes. This allows on an application level for checking, whether the system performs satisfactorily. However, it does not help when trying to determine the cause of unsatisfiable behavior. More information about the execution is collected by additional instrumentation of the whole software stack. As an example, the computation of transmission paths of packets can reveal where packets are actually lost. The transmission path must be reconstructed by instrumenting the network layer to extract the forwarding information of the nodes in the routing tree. Having determined the nodes causing the error, an analysis of these nodes can reveal the cause of the problem: Could it be collisions due to hidden-terminal effects? Are these nodes having synchronization issues or are they rebooting? These are some simple examples to describe the type of queries an event analysis framework must support.

For such an in-depth analysis and long test executions over multiple hours or even days allowing for statistically significant conclusions, the amount of logged monitoring data is substantial. Referring to our case study, even a well-focussed instrumentation renders megabytes of log files for just a day. The computation of routing paths in the presence of loops or retransmission is not easily determined with simple scripting. Further analysis has to take place with a suitable framework allowing to formulate expressive queries for the transmission paths, loop detection or correlation of events for failure debugging.

This paper provides the following contributions to alleviate this situation:

- We formulate the problem of analyzing WSN systems based on events collected in execution traces,
- We present operators for performing queries on the event trace and providing assertions on the query results for checking,
- We demonstrate the feasibility and applicability of *EvAnT*, our *Ev*ent *An*alysis framework for *T*esting.

Section II formulates the semantics of events and the event trace collected during test execution. Section III discusses the peculiarities of event analysis for Wireless Sensor Networks testing and debugging. Section IV outlines our event analysis framework and discusses the implementation of its major components. In section V, we present a case study based on the analysis of the *Harvester*, a data collection application used for building monitoring. Using EvAnT we were able to rapidly derive significant conclusions from application trace data, solely by implementing an application-specific event trace parser and by formulating the verification queries by the means provided by EvAnT. Section VI discusses related work. The work concludes with a summary and outlook.

II. Event trace formulation

WSNs are wireless distributed embedded system comprised of a large number of loosely coupled sensor nodes. For testing of a WSN, individual sensor nodes are instrumented with test monitors [5] to extract information about the execution of the software on the distributed system, i. e. the sensor nodes.

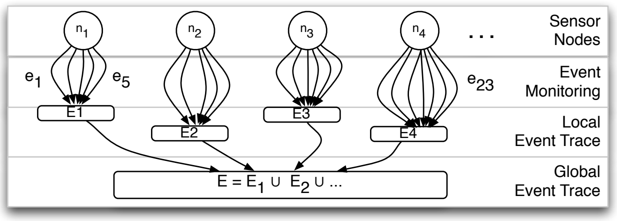

Fig. 1. Logical view of distributed system monitoring to a unified event set

A. Event specification

Test monitors collect events on the sensor nodes $n_1, ..., n_k$. Each event e is a n-tuple of key-value pairs. Events may be test- or application-specific. Minimally, an event includes a node identifier and a type identifier. Thus an event has the following format:

$e = (node : node_{id},\ type : type_{id},\ key_1 : value_1, \ldots)$

No further requirements on the content of events are defined. The events collected on the instrumented sensor node $n_1, \ldots, n_k$ form sets $E_1, \ldots, E_k$. The trace is the event set E, which is the union of the distributedly collected event sets.[1] Figure 1 illustrates the process of collecting events from the distributed targets. There is no requirement on the collection process, rather the event analysis relies on a complete view of the system including events from any node participating in the event analysis. The node identifier may be added by the test infrastructure or by the distributed node itself. Test infrastructures as described in Sec. III-A are responsible for a reliable transport of local traces rendering the event trace an accurate reflection of the actual execution of the distributed system.

The semantics of the events are defined in the analysis framework. The event analysis may work with partially ordered events based on node identifiers and local timestamps, or on a total order based on the local order and the event propagation sequence. Thus, order metrics or time are optional and do not have any syntactic restrictions for order implications.

B. Event sets and abstract events

Events in the trace represent monitoring information as atomic actions. When processing events and for behavioral abstractions events are joined into compound objects. This may be a set of events, such as the set of reboot events that have occurred. Alternatively, events can also be joined into a single event with a new representation of its constituent primitive events, e. g. for routing analysis, where individual

[1]We denote sets with upper-case letters, while events are denoted with lower-case letters.

send and receive events form a comprehensive routing path event. Nevertheless, these *abstract events* have different semantics, since they no longer represent atomic actions.

C. Temporal properties of events

Events may include temporal information. For individual events, a single timestamp suffices to describe the atomic actions. An event set or abstract event A cannot be represented with a single timestamp, since they are non-atomic. Basten et al. [6] define timestamps for abstract events. Informally, they define a function T^+ as the largest timestamp of any constituent primitive event $a \in A$ and the function T^- as the smallest timestamp of any constituent primitive event $a \in A$.

In contrast to debugging tools requiring timestamps for their analysis, our Event Analysis has no implicit semantics for temporal event information. Rather, the event analysis user explicitly introduces temporal semantics. This allows for simple queries, which do not require any temporal information as the analyses in the case study indicate. However, timestamps may be utilized to formulate temporal queries on the event set.

III. Event Analysis

A detailed analysis of a long event trace in large-scale distributed systems is tedious and error-prone. An event analysis framework alleviates this problem by allowing a systematic approach: It allows to specify behavioral aspects of a system on a higher level by formulating queries on event sets.

As depicted in Figure 2, abstracted event sets are deduced by iteratively processing the input of the event analysis, i. e. the atomic events of the trace set E. The analysis is based on selection predicates on event tuple keys and values and transformation functions on the selected events (cf. Sec. III-B). The analysis computes sets of processed events.

As Fig. 2 illustrates, the outputs of the analysis are abstract event sets, which provide extracted behavioral information such as the set of failed routing paths or a check of sets against golden results to assert a satisfiable execution, e. g. that no reboots have occurred. Figure 2 also shows that processed event sets may provide debugging or performance information such as the average hop count of routed packets or the time difference distribution for acknowledged send to the according receive events.

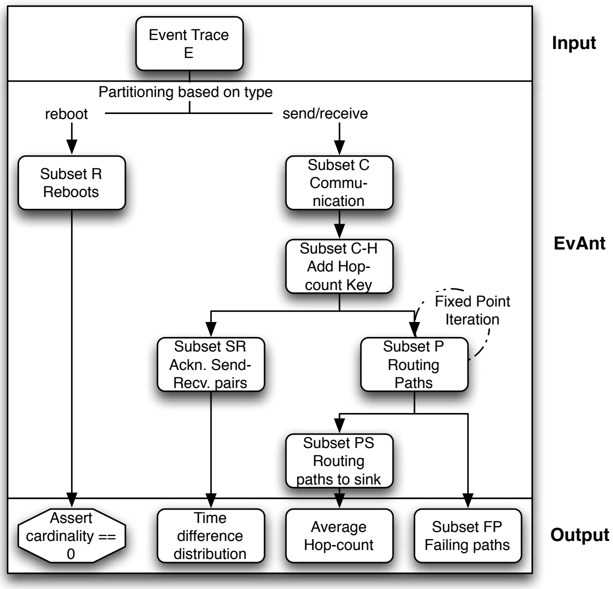

Fig. 2. Event Analysis: From trace to behavior or assertions

A. Testing and the implications for event analysis

Testing is highly application specific. In order to avoid system perturbation, monitoring is restricted to the smallest set of test monitors as required to deduce test specific information.

One reason is that access to a WSN node is bandwidth-limited. Low level interfaces such as JTAG and UART may be used for access to internal state, but only provide limited access. LEDs only allow for visual inspection of code properties, rendering them a very primitive debugging help. Instrumentation with test monitors [5] needs meticulous care to avoid perturbation of system execution. On mote platforms sharing a bus for the serial interface and the radio such as the Tmote Sky, monitor output may considerably interfere with the communication stack. Thus, tests may also be divided to only focus on specific aspects of a test execution. Furthermore, long test runs necessitate off-system logging, since memory is considerable limited on the sensor nodes.

Current Wireless Sensor Network test platforms, simulators such as TOSSIM [7] or testbeds such as Motelab [8] or the Deployment Support Network (DSN) [9], feature different logging mechanisms and formats. However, all of them typically present test results on a central test host. The test data collected by the test monitors on the distributed sensor nodes are centrally available for off-line test analysis.

Rather than imposing a structure on the monitoring and its formats, which would allow for more automation of the analysis, EvAnT imposes only minimal requirements on instrumentation and test platforms. The core of EvAnT, namely its operators, is independent of the monitoring and trace format and thus reusable across projects and platforms. With merely adding a simple event parser, EvAnT is usable for any project. Even heterogeneous logging is supported as described in Sec. V.

EvAnT provides its users flexible, yet powerful operators to describe test, application and test platform specific analysis queries and checks. Thus, the analysis framework is readily usable for any WSN project. Moreover, off-line analysis does not require any optimization, but can rely on a powerful analysis host.

B. Operators

EvAnT uses events and event sets as primitives for the analysis of a system execution. Thus, EvAnT offers the set operations of union, intersection and relative complement. Additionally, EvAnT offers four novel operators (cf. Fig. 3) especially tailored for formulating queries on WSN event sets.

- Partition Operator:

$$R_i = \{s | s \in S \land \varphi_i(s)\},\ i \in \mathbb{N} \tag{1}$$

- Set Transformator:

$$R = \{e | e \in f(A) \land A \subseteq S \land \varphi(A)\} \tag{2}$$

- Set Processor:

$$R = \{e | e \in f(s) \land s \in S\} \tag{3}$$

- Fixed Point Processor:

$$\begin{aligned} R^0 =& S \\ R^i =& \{\{e | e \in f(A) \land A \subseteq R^{i-1} \land \varphi(A)\} \cup \\ & \{e | e \in A \land A \subseteq R^{i-1} \land \neg\varphi(A)\}\} \end{aligned} \tag{4}$$

As Fig. 3 indicates, S is the set used as the input for the operators. R is the result set, i. e. the output, of a given operator. s denotes an event in the base set S. Basis for the operators are predicates φ and transformation functions $f : 2^S \to 2^S$.

A *predicate* φ is defined on one or multiple events in the selection process. The predicate uses relations on the values of specified event keys. Relations differ based on the purpose of the selection and the available trace information. The Partition Operator makes use of predicates on a single event $\varphi(s)$. The Set Transformator and the Fixed Point Processor use predicates on multiple events $\varphi(A)$.

A *transformation function* $f(A)$ is used to either add information to events or to merge events into a compound event object.

In the following, the four operators in EvAnT are described:

1) Partitioning: The partition operators allows to partition a set of events into subsets based on the values of event keys. Partitioning is performed on a single set S. The partitioning operator returns multiple sets R_i, each containing the events that satisfy a given predicate $\varphi_i(s)$.

An example for partitioning into disjunct event sets based on values of the *type* key is depicted in Fig. 2. As shown, partitioning may also be used for filtering specific event types as shown for the reboot events.

2) Set Transformator: The set transformator allows to select a subset A from a base set by using a predicate $\varphi(A)$. Selected events are processed based on a transformation function, e. g. to join multiple events into a single compound event. Processed events are added to the result set R. Processing is performed on a single set. An extension to use the operator on multiple sets is to use the union of the sets as the base set and describe a predicate that discriminates individual events based on their origin set.

An example is the set SR in Fig. 2, which selects according send and receive events, joins these into compound transmission events with an additional key-value pair describing the time difference between the sending and the reception. This allows for a computation of the time difference distribution as shown.

3) Set Processor: Set processors are available for processing single events. Each event $s \in S$ is selected and processed by a transformation function $f(s)$ on the event. This allows for adding key-value pairs for events or computations on event values.

As depicted in Fig. 2, to determine the hop-count of a routing path and computing the average, communication events need to add a hop-count key, which is used in the fixed point operator transformation function to be incremented when a send and a receive event are joined.

4) Fixed Point Processor: The Fixed Point Processor computes the least fixed point of a given function on event sets and produces a new set. Selection and processing of events in a single iteration is performed as in the Set Transformator. Iteratively the sets R^i are computed, until a fix point is determined, i. e. $R^k = R^{k-1}$. All events that are not selected by the predicate are maintained for each iteration.

An example is the computation of the routing paths: Starting from the initial transmission on the origin node, each path is traversed by joining each receive and send message into a compound transmission path event until the packet is received at the sink or the path fails.

C. Analysis for debugging versus analysis for tests

The event analysis framework is used to extract a behavioral description from the event trace E. This abstraction

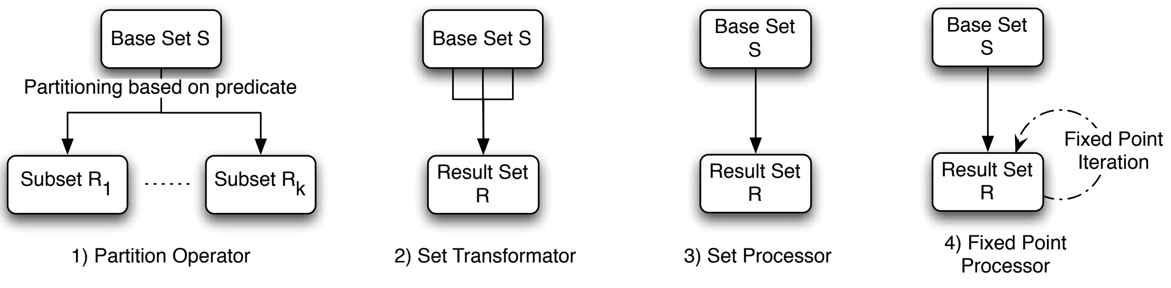

Fig. 3. The four operators in EvAnT

process allows for two different goals: the analysis for a better comprehension of actual system behavior and the checking of execution properties, e. g. for regression testing.

1) Queries
   Queries are used to extract behavioral aspects of the raw event set. Thus queries may be used for debugging, i. e. to expose the cause a failure of the WSN system. Additionally, queries and processing of the event set allow for performance evaluation. Multiple test runs with different parameter sets are easily comparable by a common analysis. Query results may be used to check for correct behavior e. g. by comparing against golden result set.
2) Checks
   Assertions [10] allow for checking execution properties and presenting assertion misses. Assertions may have different severity levels as known from hardware design. An exemplary use is to assert a warning if packets are lost in an individual run, but only assert it as an error, if packet loss occurs in more than 1 out of 100 test executions.

IV. EvAnT

EvAnT is a framework to analyze the system behavior of WSNs by the means of collected monitor events during execution of the WSN. EvAnT is generic and thus can be used for different testing platforms and programming languages. It supports heterogeneous system devices and test platforms, since it only operates on the trace. It is based on the described operators on event sets. The semantics of events is specified in EvAnT. Causality or other order relations are provided by the analysis query rather than implied by syntactical requirements.

EvAnT is implemented as a set of Python classes for event and event set handling, as well as the overall EvAnT Framework. The implementation in Python allows for easy extension with additional operators targeted at specific analyses. An analysis is started by creating an input parser for the specific test platform or monitoring data format. EvAnT already provides support for DSN database access and EvAnT-format ASCII log files. This allows for reading monitoring data and creating event sets. Further processing is performed by the implementations of the event analysis operators (cf. Sec. III-B).

Predicates are implemented as Python lambda functions returning a Boolean value. Transformation functions are formulated by constructing a new event out of the constituent key-value pairs of the selected event pairs or by introducing novel keys and values. They may also simply return the selected events. Listing 1 displays the predicate and the transformation for determining the packet yield for communication sets partitioned by origin and sequence number.

Selection predicates and transformation functions are inputs to the EvAnT operators. The operators iterate over the event sets, evaluate the predicate on events and transform selected events accordingly.

The partition function returns an associative array of the partitioned event sets addressable via their value or the index of the value interval. For enumerable values of a key, a partitioning is performed with an implicit predicate based on the discrete values, rendering disjoint sets. [2] For keys with continuous values with possibly infinite partitions, value intervals defined as boolean lambda functions may be defined allowing to divide the value range in partitions, allowing for events in multiple sets. Set Transformators and Fixed Point Processor currently support predicates and transformation functions on two events. This suffices for covering typical WSN cases such as the routing path computation as discussed. Both operators return a new event set.

Also provided are arithmetic functions on event sets returning a maximum and minimum of the numeric values of a key, as well as operators for determining mean and standard deviation of all values. These allow for simple computations for a given set as used in the drift measurement analysis in Sec. V-C for each node-neighbor pair set.

[2]Events not featuring the event key(s) are ignored.

type	key0	key1	key2	key3
received	dest addr	origin	seqNo	
senddone	dest addr	origin	seqNo	ack
drift	neighbor ID	interval	abs. drift	drift in interv.

Fig. 4. Harvester specific event tuple keys collected on each node.

V. Case Study

We evaluate EvAnT by applying it to *Harvester*, which is a typical WSN application running on TinyOS 2. Harvester collects temperature data on remote nodes to monitor the heat flows in an office building. The temperature readings are forwarded to base stations that act as gateways to the sensor network. The routing protocol bases on the TinyOS Collection Tree Protocol. To achieve a long system lifetime, Harvester minimizes the power consumption by augmenting the routing protocol with a custom low power listening (LPL) stack. The LPL stack minimizes the power consumption by the estimating wake-up times resulting in a link-based asymmetric synchronization similar to the WiseMac Protocol [11]. The sink nodes do not have power constraints because they are powered from attached PCs. Hence, for improving the bandwidth of the gateways, LPL is disabled on sink nodes.

In the case study, Harvester is executed on Tmote Sky sensor nodes that are connected to the DSN, a testbed comprised of distributed sensor and observer node-pairs and a wireless backbone channel of the observer nodes [9]. Data is forwarded to a single gateway node. We perform heterogeneous logging: the sink node directly attaches to a PC via the serial interface, while the logs of all remote nodes are collected via the DSN.

Figure 4 shows the event types that are produced by the instrumented Harvester application. Receive and send event types are extracted from the TinyOS 2 event handlers for a path and packet yield analysis. Measurement events allow for drift analysis as described in V-C. In the following analysis, we are interested in the performance of Harvester concerning its packet yield while collecting the temperature data and concerning its efficacy heavily determining the system lifetime. The events collected for this case study are rather generic. Thus, our case study is representative for a large class of WSN operating systems, sensor nodes and testing platforms.

A. Routing Analysis

To investigate the performance concerning packet yield of Harvester, one possibility is an analysis of the routing paths. The actual routing paths can be reconstructed with

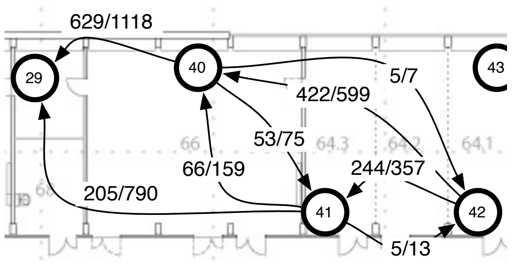

Fig. 5. Acknowledged and total number of sent packets for selected nodes

the fixed-point processors; in each iteration, send and receive events that correspond to each other are aggregated into compound transmission events.

For our analysis, we used EvAnT to determine the packet yield using the Set Transformator. The implementation of this computation in EvAnt is shown in Listing Sec. 1. While the Set Transformator could operate directly on the complete set of events, this operation takes a prohibitively long time due to the large number of events. But, we know, that all packets related to one temperature reading share the same sequence number and origin address. Thus we can first partition the initial event set based on the source address and the sequence number. The Set Transformator can then be used on each of the resulting sets separately, which significantly accelerates the computation. This methodology can also be adopted when using the computation-intensive Fixed Point Processor.

We performed multiple experiments on 17 nodes, each running for 3 hours showing that the average packet yield for each node is higher than 78%, with nodes in the one-hop neighborhood having an average packet yield higher than 90%. A more detailed analysis showed the interesting result, that for the different tests in average each temperature reading required between 2.6 and 5.2 packets to be transmitted. For a network with a small number of hops, this number of sent packets would be expected not to differ so considerably. This hints at a considerable message loss along the routes, which are counteracted by frequent retransmissions. While the impact on the packet yield is still tolerable for Harvester, the retransmissions are very expensive in terms of energy. Nevertheless, both results hint at a deeper analysis at the underlying MAC layer. The analysis of one 17-node experiment takes approximately one minute on a MacBook (2GHz Intel Core 2 Duo / 1.5 GB RAM) running OSX 10.5.1 and Python 2.5.1.

B. MAC Analysis

The MAC analysis evaluates the efficacy of the link layer and the low power listening protocol. Figure 5 shows

```
#Selection predicate matches acknowledged sent with received packet with receiver nodeid equaling sink address
pairing = lambda x,y: x.typeid == 'received' and y.typeid == 'senddone' and x.nodeid == sink
#Transforation function generates send−receive pairs updating the current node id to the receiver's node id
pair_tf = lambda x,y: event(nodeid = y.nodeid, origin = x.origin, seqNo = x.seqNo, typeid='pair')
#2−staged partitioning performs implicit equivalence selection predicate and improves EvAnT runtime
partitionOriginSet = tempset.partition('origin')
for origin in partitionOriginSet.keys():
      partitionSequenceSet = partitionOriginSet[key].partition('seqNo')
      for sequenceNumber in partitionSequenceSet.keys():
            yield_set = partitionSequenceSet[sk].set_transformator(pairing, pair_tf)
```

Listing 1. Using EvAnT's Partition and Set Transformator to determine the packet yield

ID	29	40	41	42
29	N/A	-0.964 (5539)	0.932 (3268)	0 (0)
40	0.949 (3967)	N/A	0.095 (4020)	-1.676 (4234)
41	0.444 (3610)	0.812 (3026)	N/A	-1.012 (5156)
42	0 (0)	-0.201 (2609)	0.140 (2671)	N/A

TABLE I. Drift in scaled ppm (number of measurements) for selected nodes

results from a 12 hour test run for a selected part of the network. The sink that is not shown in the figure is positioned to the left of the nodes. Each link is annotated with the acknowledged and total number of packets sent along this link. In an ideal setting (perfect synchronization and no interference) each packet should be sent only once and should be acknowledged immediately. The discrepancy between the total number of packets and the number of acknowledgments indicates that the synchronization of sender and receiver does not work properly[3]. Hence, an in-depth analysis of the wakeup time estimation and the time drift detection was performed.

C. Drift Analysis for Wakeup time estimation

We determine the drift measurement data from measurements packets, which are exchanged by neighboring nodes to determine the clock drift. In a scenario with perfect synchronization each node pair n_1 measures the same drift of the local timebase to the timebase of its neighbor n_2. That is, if node n_1 determines the drift of node n_2 as $+\tau$, node n_2 will compute a drift of $-\tau$. Thus, in a perfectly synchronized scenario the computed drifts add up to 0.

We ran this test twice for 24 consecutive hours and have accumulated the results in a single event set. Joining analysis results, i. e. abstract event sets is easily implemented in EvAnT. We collected a total of 494316 measurement events. Table I shows that certain links are synchronized accurately (e. g. 29 and 40), while some links are totally off (40 and 42).

[3]While interference problems are also possible, previous measurements on the testbed and current test results indicate that this is considerably less likely.

Hence, we have determined in our analysis that while the overall packet yield is tolerable, the implementation of the custom LPL stack needs improvement concerning the drift measurement and interpretation. One of the problems can be attributed to a timestamping problem of the CC2420 driver as discussed on the TinyOS mailing list [12], where corrupted or stale timestamps might be applied to a packet.

VI. Related work

There is no previous work concerning event analysis for WSN testing. However there exists previous work on the related fields of test platforms, WSN health monitoring, passive monitoring frameworks and on-line data mining:

Test platforms for WSNS include simulators such as TOSSIM [7] or EmTos in EmStar [13] and testbeds such as Motelab [8] or the DSN [9]. EmSim [13] allows for heterogeneous testing with simulation and actual hardware. Test platforms provide the input data to our event analysis framework, which benefits from an improved view of the system.

Yang et al. have shown the benefits of a wireless source-level debugger for WSNs [14]. This is complementary to our work, since we target analysis for testing and for debugging on a macroscopic level, while Clairvoyant targets debugging of microscopic effects such as race conditions or stack overflows.

Network health monitoring tools ([15], [16]) allow for detecting and debugging failures by providing and communicating additional state information. They extend the communication protocol to provide collaborative, automatic maintenance and recovery of WSNs after deployment. The main target is to provide on-line health monitoring for typical data collection application in an energy-efficient manner by providing common failure indications. Our approach is orthogonal, since EvAnT is targeted for pre-deployment and providing an analysis framework for testing various WSN applications on different test platforms.

Passive inspection of distributed wireless systems as proposed for WSNs ([17], [18]) relies on traffic snooping and automatic analysis of collected information. Algorithms are focussed on detecting indicators on the basis of an incomplete view. Ringwald et al. [17] presents typical indicators for WSNs focussed on passive inspection, which may be adapted for usage in EvAnT. Maifi et al. [18] use machine learning techniques and provide predicates for anomalous behavior, which could also be used in EvAnT. EvAnT rather relies on a comprehensive view and allows for specifying test- and application-specific aspects.

Data mining techniques may be used for WSNs, e. g. to perform an online analysis of collected data for traffic reduction [19]. However on-line analysis of WSNs is restricted to spatially and temporally local predicates and infer a considerable overhead if used for testing on the sensor node.

Differing to all the previous approaches, EvAnT provides the benefit of being application and platform independent and thus readily applicable to any project.

VII. Summary and outlook

We have formulated the problem of analyzing Wireless Sensor Networks systems based on events collected in traces. We defined novel and powerful operators on event sets for performing expressive queries on the event trace. We also showed the usage of assertions on the query results for checking test executions. We presented EvAnT, our event analysis framework for WSN systems. EvAnT allows for analyzing arbitrary event traces from a system execution by merely providing an input parser or using one of the supported event formats. Thus, EvAnT is generally applicable for different WSN and test platforms. EvAnT can be used for macroscopic debugging by defining queries on the event sets or for testing by formulating assertions on the query sets. A case study showed the application of EvAnT to Harvester, a typical application. EvAnT allowed us with an explorative, iterative analysis to reveal the problem of Harvester as the drift measurements.

In the future, we plan to apply the framework to more projects. The ease of use of EvAnT can be extended with an instrumentation approach that automatically extracts the event format at compile time and provides an input parser. EvAnT is available for other researches in the WSN community.

VIII. Acknowledgements

The work presented here was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

References

[1] J. I. Choi, J. W. Lee, M. Wachs, and P. Levis, "Opening the sensornet black box," *SIGBED Rev.*, vol. 4, no. 3, pp. 13–18, 2007.

[2] G. Werner-Allen *et al.*, "Monitoring volcanic eruptions with a wireless sensor network," in *Proc. 2nd European Workshop on Sensor Networks (EWSN 2005)*, 2005, pp. 108–120.

[3] V. Turau, M. Witt, and M. Venzke, "Field trials with wireless sensor networks: Issues and remedies," in *Proc. of the Int'l Multi-Conference on Computing in the Global Information Technology (ICCGI '06)*, 2006, p. 86.

[4] K. Langendoen, A. Baggio, and O. Visser, "Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture," in *Proc. 20th Int'l Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006, pp. 8–15.

[5] M. Woehrle, C. Plessl, J. Beutel, and L. Thiele, "Increasing the reliability of wireless sensor networks with a distributed testing framework," in *Proc. 4th IEEE Workshop on Embedded Networked Sensors (EmNetS-IV)*, 2007.

[6] T. Basten *et al.*, "Vector time and causality among abstract events in distributed computations," *Distrib. Comput.*, vol. 11, no. 1, pp. 21–39, 1997.

[7] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003)*, 2003, pp. 126–137.

[8] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: A wireless sensor network testbed," in *Proc. 4th Int'l Conf. Information Processing Sensor Networks (IPSN '05)*, 2005, pp. 483–488.

[9] M. Dyer *et al.*, "Deployment support network - a toolkit for the development of WSNs," in *Proc. 4th European Workshop on Sensor Networks (EWSN 2007)*, 2007, pp. 195–211.

[10] D. S. Rosenblum, "Towards a method of programming with assertions," in *ICSE '92: Proceedings of the 14th international conference on Software engineering*, New York, NY, USA, 1992, pp. 92–104.

[11] A. El-Hoiydi and J. Decotignie, "WiseMAC: An ultra low power MAC protocol for multi-hop wireless sensor networks," in *Proc. 1st Int'l Workshop Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2004)*, 2004, pp. 18–31.

[12] D. Moss *et al.*, "Bug in cc2420 timestamp," https://www.millennium.berkeley.edu/pipermail/tinyos-help/2007-October/028901.html, October 2007.

[13] L. Girod *et al.*, "Emstar: A software environment for developing and deploying heterogeneous sensor-actuator networks," *ACM Trans. Sen. Netw.*, vol. 3, no. 3, p. 13, 2007.

[14] J. Yang, M. L. Soffa, L. Selavo, and K. Whitehouse, "Clairvoyant: A comprehensive source-level debugger for wireless sensor networks," in *Proc. 5th ACM Conf. Embedded Networked Sensor Systems (SenSys 2007)*, 2007.

[15] S. Rost and H. Balakrishnan, "Memento: A health monitoring system for wireless sensor networks," in *Proc. 3rd IEEE Communications Society Conf. Sensor, Mesh and Ad Hoc Communications and Networks (IEEE SECON 2006)*, 2006.

[16] N. Ramanathan *et al.*, "Sympathy for the sensor network debugger," in *Proc. 3rd ACM Conf. Embedded Networked Sensor Systems (SenSys 2005)*, 2005, pp. 255–267.

[17] M. Ringwald, K. Römer, and A. Vitaletti, "SNIF: Sensor network inspection framework," Department of Computer Science, ETH Zurich, Technical Report 535, Oct. 2006.

[18] M. Maifi *et al.*, "SNTS: Sensor network troubleshooting suite," in *Distributed Computing in Sensor Systems*, vol. Volume 4549/2007. Springer, 2007, pp. 142–157.

[19] K. Römer, "Discovery of frequent distributed event patterns in sensor networks," in *Proc. European Workshop on Wireless Sensor Networks (EWSN 2008)*, Bologna, Italy, Jan. 2008, pp. 106–124.