Poster Abstract: Rupeas - An Event Analysis Language For Wireless Sensor Network Traces

Matthias Woehrle [#], Christian Plessl [*], Lothar Thiele [#]

[#] *ETH Zurich, Computer Engineering and Networks Lab*
8092 Zurich, Switzerland
[*] *University of Paderborn, Paderborn Center for Parallel Computing*
33102 Paderborn, Germany
matthias.woehrle@tik.ee.ethz.ch

Abstract— **Wireless Sensor Networks (WSNs) are distributed systems of embedded sensor nodes loosely coupled by wireless communication. Debugging, profiling and testing of these systems is inherently challenging due to resource constraints of the system components, a vast state space and tight integration with the environment. A prominent validation strategy is to use testbeds running an instrumented version of the software on distributed sensor nodes and to collect data during execution for off-line analysis. We provide a novel programming language named Rupeas which especially targets the analysis of these WSN logs.**

I. Introduction

Testing of WSNs is a crucial part of the development process as a functional, performing and resource effective system at deployment time is of utmost importance [6]. A main methodology for testing is using a WSN specific test platform such as a testbed [1] or simulator [2] and logging test specific information. Tests result in a substantial amount of log data, which needs to be analyzed in the context of the test platform and the application. Such an analysis requires adequate tools.

With *Rupeas*, a **R**uby **P**owered **E**vent **A**nalysi**s** language, we provide a tool for WSN analysis. It facilitates analysis of WSN logs by using an event abstraction for representing the execution traces. It provides operators to formulate queries on sets of events. Rupeas provides a high ease of use with its simple, concise notation. It greatly simplifies WSN trace analysis and testing, which are often perceived as meticulous tasks.

As an example, an interesting analysis of typical WSN applications for data gathering is whether and to what extent sensory data can be extracted from the system. Typical questions are: What is the data yield from each node? What do the routing paths look like? Are the routing paths efficient?

These are sample questions that can be answered by Rupeas queries. Rupeas allows for determining routing paths by formulating a relation on send and receive events gathered in traces, even for multi-sink systems as shown in Fig. 2.

Previously proposed approaches cannot help with this analysis: Diagnostic simulation [4] using data mining techniques on simulation data, can help you with outlier detection, but does not provide a comprehensive analysis. Assertions on distributed, global state [3], [5] can help you identify problems in parent selection of the routing protocol, but do not help in the analysis of taken routes.

II. Rupeas

Rupeas is based on ideas first introduced in the EvAnT framework [7] for analyzing WSNs. Both, Rupeas and EvAnT, use an event abstraction: Individual log entries are events, tuples of key-value pairs; logs are represented as a set of events. Both are based on four generic operators for processing events sets and allow for formulating event set queries and assertions for testing.

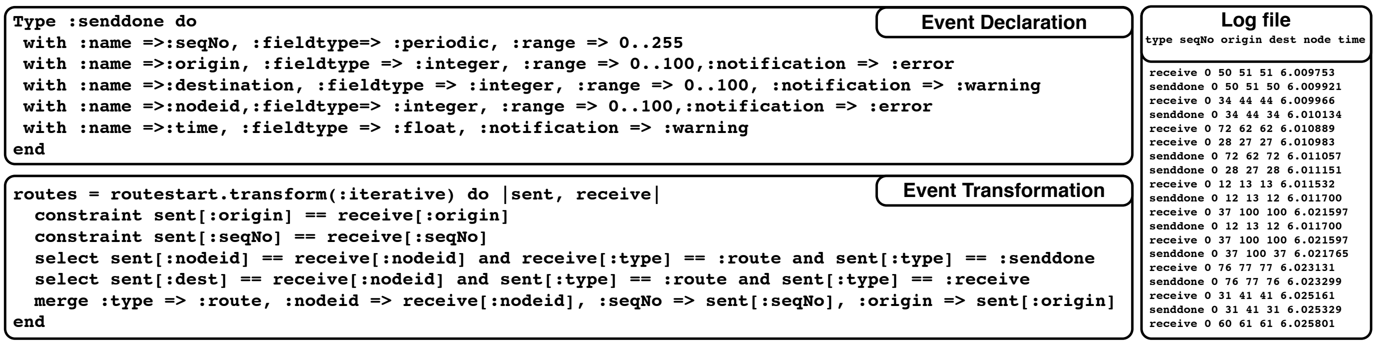
Rupeas provides two major enhancements over EvAnT: Firstly, Rupeas is a novel programming language for WSN analysis and testing, a Domain Specific Language (DSL). The EvAnT framework merely provides limited API's on top of a general purpose language, which lacks a focus on the problem domain of the analysis of WSN logs. Secondly, Rupeas introduces descriptions for the event types in the trace to specify format and expected content of each event.

Rupeas is especially designed to allow for an intuitive formulation of queries on event sets. As a DSL, Rupeas is a language specifically designed for the given problem domain, capturing its essence and neglecting any general purpose aspects. Rupeas is an internal DSL embedded in Ruby. Ruby as a host allows for elaborate analysis of the processed event sets and integration into a comprehensive test framework [6].

A Rupeas program features two main sections:

(1) The *event description* section specifies keys and expectations on values for the each type of event. The individual events are verified when being loaded from the trace. One specific feature is to provide periodic value types for events, which are used for wrapping counters such as sequence numbers to allow for determining a total order. An abbreviated description of send events in Rupeas is shown in Figure 1.

(2) The *event processing* specifies the processing that is applied to the event set. The DSL is designed to facilitate transformations and queries of event sets by means of event set operators using a simple and concise notation. An example for such processing is the analysis of routing paths. Starting with a single packet originating at a source node, Rupeas' fixed point operator allows for iterative merging of send and receive events. After the fixed point is reached, the merged


Event Declaration

```
Type :senddone do
 with :name =>:seqNo, :fieldtype=> :periodic, :range => 0..255
 with :name =>:origin, :fieldtype => :integer, :range => 0..100,:notification => :error
 with :name =>:destination, :fieldtype => :integer, :range => 0..100, :notification => :warning
 with :name =>:nodeid,:fieldtype=> :integer, :range => 0..100,:notification => :error
 with :name =>:time, :fieldtype => :float, :notification => :warning
end
```

Event Transformation

```
routes = routestart.transform(:iterative) do |sent, receive|
  constraint sent[:origin] == receive[:origin]
  constraint sent[:seqNo] == receive[:seqNo]
  select sent[:nodeid] == receive[:nodeid] and receive[:type] == :route and sent[:type] == :senddone
  select sent[:dest] == receive[:nodeid] and sent[:type] == :route and sent[:type] == :receive
  merge :type => :route, :nodeid => receive[:nodeid], :seqNo => sent[:seqNo], :origin => sent[:origin]
end
```

Log file

```
type seqNo origin dest node time

receive 0 50 51 51 6.009753
senddone 0 50 51 50 6.009921
receive 0 34 44 44 6.009966
senddone 0 34 44 34 6.010134
receive 0 72 62 62 6.010889
receive 0 28 27 27 6.010983
senddone 0 72 62 72 6.011057
senddone 0 28 27 28 6.011151
receive 0 12 13 13 6.011532
senddone 0 12 13 12 6.011700
receive 0 37 100 100 6.021597
senddone 0 12 13 12 6.011700
receive 0 37 100 100 6.021597
senddone 0 37 100 37 6.021765
receive 0 76 77 77 6.023131
senddone 0 76 77 76 6.023299
receive 0 31 41 41 6.025161
senddone 0 31 41 31 6.025329
receive 0 60 61 61 6.025801
```

Fig. 1. Rupeas event description of :senddone events, transformation syntax of fixed point operator and a sample of the according trace.

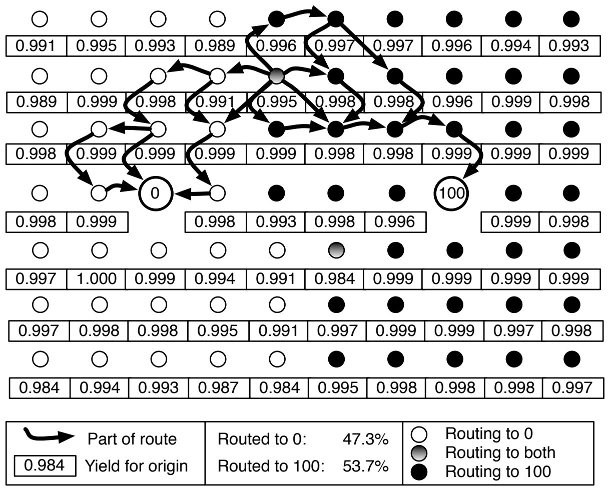


Fig. 2. Visualization of Rupeas results.

send-receive pairs represent all routing paths with detailed information on the exact paths taken for each origin node and sequence number.

III. Case Study

As a preliminary example for an analysis of a data gathering application, we apply Rupeas to data from simulating a TinyOS 2.x application. We instrument the Collection Tree Protocol to log sent and received packets and simulate the system in TOSSIM [2]. The simulation topology is a 70 node grid (cf. Fig. 2) including *two sink nodes* (0, 100). The simulation uses gain and noise models based on USC's Realistic Wireless Link Quality Model and Generator [1]. The log file and input for Rupeas captures data from a 6 hour run resulting in over 2 million events.

The analysis features two basic steps: Loading the event trace using the event description and using the fixed point processor(cf. Fig. 1). Rupeas allows for filtering routing paths for final destination, route selection or hop count. The data can be easily extracted with Ruby into different file formats. The main results obtained are visualized in Figure 2. It depicts the yield of individual origin nodes and their associated sink. The data yield of the simulation is high and the traffic is routed evenly among the sinks. As an example, Figure 2 displays the links of nine different routing paths actually taken in the experiment for node 24. The packets are routed via minimally two and maximally six intermediate hops. 50.4% of the packets are routed to sink 0.

This analysis is only a single example of Rupeas capabilities. Further details on Rupeas are available on the Rupeas project page[2].

IV. Summary

We presented Rupeas, a novel language targeted at analyzing WSN logs. The DSL provides a high ease-of-use by capturing the essence of the WSN test domain. Rupeas is an open-source project trying to facilitate testing and thereby stimulating testing efforts in the WSN community.

V. Acknowledgments

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] M. Dyer et al. Deployment support network - a toolkit for the development of WSNs. In *Proc. EWSN 2007*, pages 195–211, 2007.

[2] P. Levis et al. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proc. SenSys 2003*, pages 126–137, 2003.

[3] M. Lodder et al. A global-state perspective on sensor network debugging. In *Proc. HotEmNets 2008*, 2008.

[4] M. Maifi et al. Towards diagnostic simulation in sensor networks. In *Distributed Computing in Sensor Systems*, pages 252–265. Springer, 2008.

[5] K. Römer and M. Ringwald. Increasing the visibility of sensor networks with passive distributed assertions. In *Proc. REALWSN '08*, 2008.

[6] M. Woehrle et al. Increasing the reliability of wireless sensor networks with a distributed testing framework. In *Proc. EmNetS 2007*, pages 93–97, 2007.

[7] M. Woehrle et al. EvAnT: Analysis and checking of event traces for wireless sensor networks. In *Proc. SUTC 2008*, pages 201–208, 2008.

[1]http://anrg.usc.edu/www/index.php/Downloads

[2]http://code.google.com/p/rupeas/