Demonstration Abstract: Software-Only System-Level Record and Replay in Wireless Sensor Networks

Matthew Tancreti[1], Vinaitheerthan Sundaram[2], Saurabh Bagchi[1,2], and Patrick Eugster[2,3,*]

[1]School of Electrical and Computer Engineering, Purdue University

[2]Department of Computer Science, Purdue University

[3]Department of Computer Science, TU Darmstadt

ABSTRACT

Wireless sensor networks (WSNs) are plagued by the possibility of bugs manifesting only at deployment. However, debugging deployed WSNs is challenging for several reasons—the remote location of deployed sensor nodes, the non- determinism of execution that can make it difficult to replicate a buggy run, and the limited hardware resources available on a node. In particular, existing solutions to *record and replay* debugging in WSNs fail to capture the complete code execution, thus negating the possibility of a faithful replay and causing a large class of bugs to go unnoticed. In short, record and replay logs a trace of predefined events while a deployed application is executing, enabling replaying of events later using debugging tools. Existing recording methods fail due to the many sources of non-determinism and the scarcity of resources on nodes.

In this demo, we present Trace And Replay Debugging In Sensornets (TARDIS), a software-only approach for deterministic record and replay of WSN nodes. TARDIS is able to record *all* sources of non-determinism, based on the observation that such information is compressible using a combination of techniques specialized for respective sources. Despite their domain-specific nature, the techniques presented are applicable to the broader class of resource-constrained embedded systems.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

Keywords

Wireless Sensor Networks, Tracing, Debugging, Replay

* P. Eugster is partially supported by the German Research Foundation (DFG) under project MAKI ("Multi-mechanism Adaptation for the Future Internet").

IPSN '15, April 14 - 16, 2015, Seattle, WA, USA
ACM 978-1-4503-3475-4/15/04.
http://dx.doi.org/10.1145/2737095.2741839

1 Introduction

Debugging is one of the fundamental tools for identifying software defects ("bugs"). Debugging is particularly relevant in wireless sensor networks (WSNs), as these are susceptible to unpredictable runtime conditions. Indeed, programmers of WSNs use tools such as simulators [6, 2], safe code enforcement [1], and formal testing [4] prior to deployment of an application in the field, yet exhaustive testing of *all* conditions in the lab is infeasible, because WSNs are deployed in austere environments whose behavior cannot be easily duplicated in a laboratory.

Debugging is often performed in a cyclic process of repeatedly executing a program and tracking down bugs. In WSNs, cyclic debugging can be lengthy and laborious. For example, nodes are often not easily physically accessible, meaning that the programmer must rely on low-power wireless links to painstakingly collect any data of interest. Also, there may not be enough information available to immediately diagnose a bug, so the network must be wirelessly reprogrammed with code to collect additional debugging data. This can take minutes, and waiting for the bug to resurface may also take some time.

1.1 Record and Replay

Record and replay can potentially make the process of cyclic debugging less tedious. With record and replay debugging, program execution is recorded on-line and then reproduced off-line. Record and replay cuts down on the cyclic process of debugging by capturing a program's execution such that it can be *deterministically* reproduced and carefully examined off-line, perhaps in a hunt for elusive bugs [8]. In addition, in WSNs, the recording can happen on deployed nodes and the debugging can happen on the relatively resource rich lab machines. The typical workflow for record and replay in WSNs is that during normal execution of a deployed WSN, the nodes execute instrumented binaries that record a trace of all sources of non-determinism to flash. The trace can then be brought back to the lab for off-line replay. This can be done either through wireless data collection or by physically accessing a node. In the lab, the recorded data is fed into an emulator, which deterministically replays the node's execution. The replay allows a developer to examine the program's execution, including its interactions with the environment, at any arbitrary level of detail, such as through setting breakpoints or querying the state of memory. Such replay helps the developer identify the root cause of bugs encountered in the field.

However, realizing record and replay for WSNs (and to some extent in other embedded systems) is challenging for several reasons: the record system must fit withing the bounds of the severe resource constraints typical of WSNs, the record system must not interfere with the soft real-time constraints of WSNs, the system should be able to replay any execution and memory state to be able to reproduce all types of software bugs, and the system should be easily portable to the various embedded platforms and OSs.

1.2 Contributions

TARDIS is designed as a software-only system-level record and replay solution for WSNs. In short, we address the four challenges described above by handling *all* of the sources of non-determinism and compressing each one in a resource efficient manner using respective domain-specific knowledge. For example, one type of non-determinism is a read from what we call a *peripheral register*. These are registers present on the μC chip, but whose content is controlled from sources external to the main processor. Reads to a register containing the value of an on-chip analog-to-digital (ADC) converter are sources of non-determinism. We can reduce the number of bits that must be stored by observing that an ADC configured for 10-bit resolution in fact only has 10 bits of non-determinism, despite 16 bit register size.

The compression scheme for each source of non-determinism is informed by a careful observation of the kinds of events that typically occur in WSN applications, for example, the use of register masking which reduces the number of bits which must be recorded—instead of the full length of the register, only the bits that are left unmasked need be recorded. By using the different compression schemes in an integrated manner in one system, *we are the first to provide a general-purpose software-only record-and-replay functionality for WSNs*. By "general-purpose" we mean that it can handle *all* sources of non-determinism and thus TARDIS can be used for debugging *all* kinds of bugs, whether related to data flow or control flow. Previous work in software-based record-and-replay for WSNs has captured only control flow (e.g., TinyTracer [7]) or only a subset of system variables (e.g., EnviroLog [5]).

2 Design and Implementation

The main capability of TARDIS is to replay in an emulator the original run of a sensor node faithfully down to each instruction and the sequence between instructions. Deterministic replay is achieved by starting from a checkpoint of the processor's state, and then replaying all sources of non-determinism [3]. There are two broad sources of non-determinism in WSNs: external inputs from memory mapped I/O and the type and timing of interrupts. We will use the term *peripheral registers* to refer to memory mapped I/O, which includes registers that report the value of serial I/O, real-time clocks, interrupt flags, analog-to-digital converters, etc.

TARDIS is designed to be used *in situ* to record events in deployed sensor nodes for subsequent troubleshooting. The overall operational flow is depicted in Figure 1, which depicts three phases: compile-time, run-time, and off-line replay. In the first phase, the CIL source-to-source compiler is used to insert instrumentation for recording, into the code which will execute on the sensor node. In the second phase, the node executes *in situ*, and logs a checkpoint and a trace of its execution to flash. It operates in the manner of a black box recorder; when the flash is full, a new checkpoint is taken and

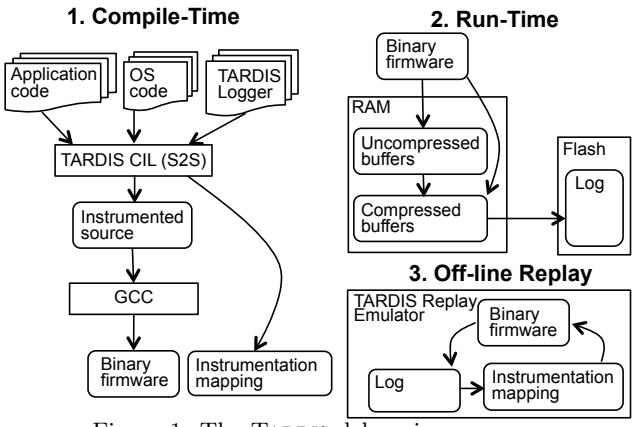

Figure 1: The TARDIS debugging process.

the oldest data is overwritten first. The third phase is the replay, which happens in the laboratory running an emulator on a (comparatively) resource-rich desktop-class machine. During execution of the application on the emulator, the trace of non-deterministic data is used to deterministically reproduce the node's execution.

3 Demonstration

With a laptop connected to a mote, we will demonstrate all three phases of TARDIS. First we will compile an unmodified TinyOS application using the TARDIS compiler wrapper. Then we will start the mote in trace mode, let it run for some time recording its trace to flash, and then dump the collected trace to the laptop using a serial connection. Finally, we will replay the trace in a modified version of the mspsim emulator, demonstrating that every instruction and state of memory can be reproduced.

4 Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Nos. ECCS-0925851 and CNS-0834529. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

5 References

[1] N. Cooprider, W. Archer, E. Eide, D. Gay, and J. Regehr. Efficient memory safety for TinyOS. In *Proc. of SenSys*. ACM, 2007.

[2] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón. Towards interoperability testing for wireless sensor networks with COOJA/MSPSim. In *Proc. of EWSN*. Springer, 2009.

[3] S. T. King, G. W. Dunlap, and P. M. Chen. Debugging operating systems with time-traveling virtual machines. In *Proc. of ATEC*. USENIX, 2005.

[4] P. Li and J. Regehr. T-check: Bug finding for sensor networks. In *Proc. of IPSN*. ACM, 2010.

[5] L. Luo, T. He, G. Zhou, L. Gu, T. F. Abdelzaher, and J. A. Stankovic. Achieving repeatability of asynchronous events in wireless sensor networks with envirolog. In *Proc. of INFOCOM*. IEEE, 2006.

[6] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with COOJA. In *Proc. of LCN*. IEEE, 2006.

[7] V. Sundaram, P. Eugster, and X. Zhang. Efficient diagnostic tracing for wireless sensor networks. In *Proc. of SenSys*. ACM, 2010.

[8] H. Thane and H. Hansson. Using deterministic replay for debugging of distributed real-time systems. In *Proc. of Euromicro-RTS*. IEEE, 2000.