Demo Abstract: Integrating Symbolic Execution with Sensornet Simulation for Efficient Bug Finding

Fredrik Österlind, Adam Dunkels
Swedish Institute of Computer Science
fros@sics.se, adam@sics.se

Raimondas Sasnauskas,
Oscar Soria Dustmann, Klaus Wehrle
Communication and Distributed Systems,
RWTH Aachen University, Germany
{lastname}@comsys.rwth-aachen.de

Abstract

High-coverage testing of sensornet applications is vital for pre-deployment bug cleansing, but has previously been difficult due to the limited set of available tools. We integrate the KleeNet symbolic execution engine with the COOJA network simulator to allow for straight-forward and intuitive high-coverage testing initiated from a simulation environment. A tight coupling of simulation and testing helps detect, narrow down, and fix complex interaction bugs in an early development phase. We demonstrate the seamless transition between COOJA simulation and KleeNet symbolic execution. Our framework enables future research in how high-coverage testing tools could be used in cooperation with simulation tools.

1 Introduction

Debugging sensornets is a notoriously difficult and tedious task. The embedded nature of resource-constrained devices and their non-deterministic environment inevitably contains a large number of unforeseen error sources. Therefore, integrated debugging tools play an essential role in increasing sensornets' reliability before deployment.

Proof-of-concept tools such as T-Check [1] and KleeNet [3] have shown that high-coverage testing is a viable tool for finding sensornet bugs. We present a tool that allows high-coverage testing to be executed directly from a large-scale sensor network simulation, thereby significantly extending the usefulness of high-coverage testing for sensor network bug finding.

Our contribution in the context of this demonstration is an integrated framework for high-coverage testing of Contiki applications using KleeNet. Aiming at usability with low manual effort we offer the community an automated symbolic execution engine with configuration and bug replay capabilities in the COOJA network simulator [2]. Moreover, we demonstrate how different testing setup strategies affect the testing coverage.

2 Bug finding in COOJA/KleeNet

The workflow of bug finding with COOJA/KleeNet is summarized in Figure 2. First, a COOJA simulation scenario is created containing the simulated applications, the network size, and the network topology. The scenario may be repeatedly simulated using different initial random seeds, thus forming multiple execution paths. Varying the random seeds is commonly used to increase test coverage of simulation, however, in comparison to symbolic execution the test coverage remains low.

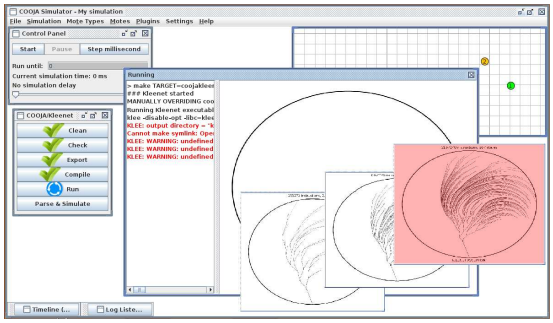

Figure 1. The COOJA simulation is seamlessly exported and high-coverage tested in KleeNet. When the test phase is completed, the resulting execution paths and symbolic accesses can be replayed in simulation.

Second, the simulation is configured for execution in KleeNet: the user chooses which variables should be accessed symbolically and specifies (distributed) assertions to be checked during applications' execution. For example, to test a flooding algorithm the contents of the first radio transmission may be set to be symbolic, with the application assertion that the packet is eventually received by the entire network. In addition, KleeNet supports node failures and reboots, radio packet loss, corruption and duplication. The simulation scenario is now exported and recompiled for execution in KleeNet. The export is seamless in the sense that the same code can be both simulated, symbolically executed, and later replayed.

During execution, KleeNet simultaneously executes and tests multiple execution paths, resulting in high-coverage testing. Explored paths and any triggered assertions are presented during runtime; see the screenshot in Figure 1.

When the KleeNet execution is finished, a summary of all the explored paths and assertion failures are presented, allowing the user to load and replay the scenarios in simulation. During replay, the symbolic variables will assume the

SenSys'10, November 3–5, 2010, Zurich, Switzerland.
ACM 978-1-4503-0344-6

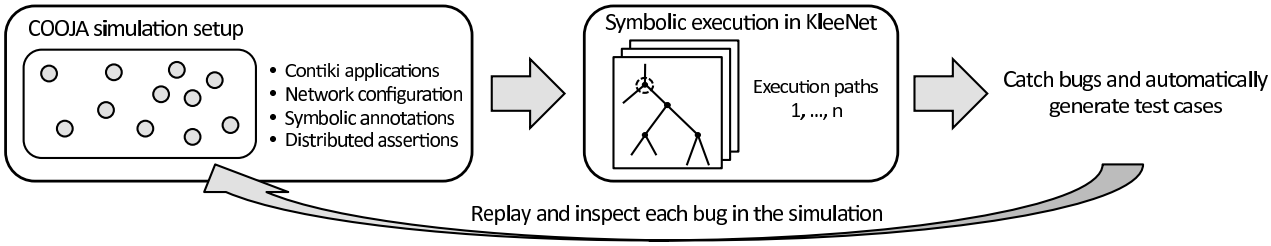

Figure 2. **COOJA/KleeNet workflow: after configuring the network and the application assertions in simulation, a developer migrates to symbolic execution in KleeNet. Assertion failures found by KleeNet are finally replayed and debugged in simulation.**

values of the loaded scenario, thus repeating the same execution path that previously triggered the assertion in KleeNet. Since the application is now simulated, the user has full access to simulation tools.

3 Integrating COOJA and Kleenet

Both COOJA and KleeNet can test unmodified Contiki applications in various network scenarios. Although being architecturally different, they execute the same code at the same level of abstraction. We argue that this is a major prerequisite for accurate integration of any testing tool into a existing network simulator. Nevertheless, during the integration phase we had to implement the following extensions:

Simulation scenario export. A simulation setup in COOJA is configured over the GUI where a user can select different mote types, applications, and radio mediums for a particular network scenario. Each simulation scenario can now be directly exported as KleeNet test executable.

Execution model. COOJA—as any other discrete event simulator—employs an event queue for efficient simulation execution. We extended KleeNet with an event queue searcher to switch between the explored execution paths in a COOJA compliant manner.

Distributed scenario replay. During its execution, KleeNet generates test cases for each explored distributed scenario, bug, or distributed assertion violation. In COOJA, we parse the resulting test cases, display a summary, and allow the user to replay a distributed scenario of choice. Consequently, all symbolic variables are replaced by concrete test case values. Upon reaching the detected bug, the simulation is paused for detailed failure analysis and debugging.

4 Demo Setup

We demonstrate the benefits of a tight integration of high-coverage testing and sensornet simulation. More specifically, we show how to prepare a sensornet application for symbolic execution, and discuss advantages and limitations of the symbolic execution approach within sensor networks. We further demonstrate how a triggered assertion is replayed into simulation, to finally locate the bug that caused the assertion failure.

Simulation scenarios. We prepare simulation scenarios with sensornet failures due to (1) packet-loss at the network level, (2) mote outage/reboot, and (3) unexpected packet input. We present the appropriate assertions to catch these failures using KleeNet. Then, we replay the execution paths hitting those bugs for detailed analysis in COOJA.

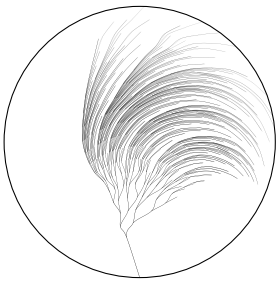

Figure 3. **An execution path tree graph generated at runtime by COOJA/KleeNet. Depending on the number of symbolic variables, the number of execution paths grows exponentially.**

Visualizing execution paths. Our framework visualizes[1] execution paths explored by KleeNet at runtime (see the example tree graph in Figure 3). We visually show how the number of symbolic variables and KleeNet's search policy affects the resulting tree graph, and discuss its potential implications on bug finding efficacy.

5 Conclusions

Currently, the COOJA simulation is exported in its initial state, i.e. it is not possible to migrate an arbitrary simulation state to symbolic execution. As a result, potential corner-case states which only emerge after a long period of time including intensive mote interactions might not be explored due to state explosion or testing time constraints. Therefore, our future work includes selective symbolic execution—a seamless state transition from simulation to symbolic execution at any moment of simulation time.

6 References

[1] Peng Li and John Regehr. T-Check: Bug Finding for Sensor Networks. In *IPSN '10: Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2010.

[2] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, Tampa, Florida, USA, November 2006.

[3] R. Sasnauskas, O. Landsiedel, M. H. Alizai, C. Weise, S. Kowalewski, and K. Wehrle. KleeNet: Discovering Insidious Interaction Bugs in Wireless Sensor Networks Before Deployment. In *ACM IPSN '10*, Stockholm, Sweden, 2010.

[1] We use the execution path tree generation script from KLEE software repository at `http://klee.llvm.org`.