Towards Lightweight Logging and Replay of Embedded, Distributed Systems[⋆]
(Invited Paper)

Salvatore Tomaselli and Olaf Landsiedel

Computer Science and Engineering
Chalmers University of Technology, Sweden
`tiposchi@tiscali.it`, `olafl@chalmers.se`

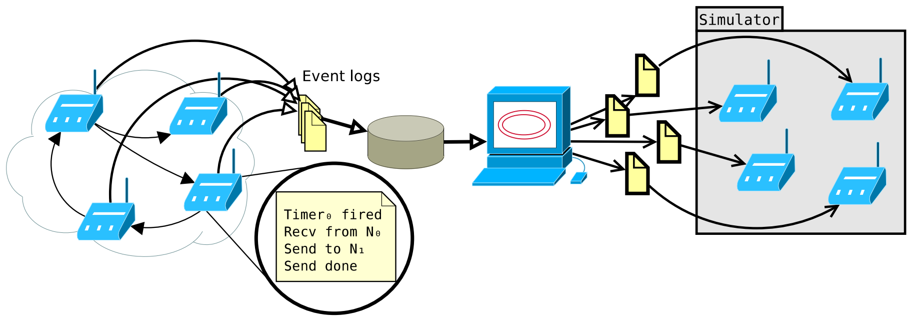
Abstract. Due to their safety critical nature, Cyber-Physical Systems such as collaborative cars or smart grids demand for thorough testing and evaluation. However, debugging such systems during deployment is challenging, due to the concurrent nature of distributed systems and the limited insight that any deployed system offers.
In this paper we introduce MILD; providing Minimal Intrusive Logging and Deterministic replay. MILD enables logging of events on deployed Cyber-Physical Systems and the deterministic replay in controlled environments such as system simulators. To illustrate the feasibility and low overhead of our architecture, we evaluate a prototype implementation based on Wireless Sensor Networks (WSN) in this paper.

1 Introduction

Cyber-Physical Systems connect the physical world with the cyber-space. Their programmability and communication capabilities enable new applications and use cases such as cooperative vehicles or smart power-networks. These safety critical applications demand for deep testing and evaluation. Due to both their interaction with the physical world, such applications are difficult to test in traditional setups such as unit-testing or simulation. Instead, we often employ large scale testbeds for testing and real-world performance evaluation. For Wireless Sensor Networks, these are often controlled deployments of hundreds nodes covering buildings, mobile robots, or outdoor environments. However, debugging such systems during deployment is challenging, due to the concurrent nature of distributed systems and the limited insight that any deployed system offers.

In this paper, we introduce MILD for Minimal Intrusive Logging and Deterministic replay of deployed Cyber-Physical Systems. MILD enables (1) logging of events with minimal intrusion of deployed systems, and (2) the deterministic replay of events in controlled environments such as system simulators. As a result, MILD offers deep insights into real-world deployments and allows debugging and testing in realistic settings.

[⋆] This work was partially supported by the EC, through project FP7-STREP-288195, KARYON (Kernel-based ARchitecture for safetY-critical cONtrol).



Timer$_0$ fired
Recv from N$_0$
Send to N$_1$
Send done

Fig. 1: Global architecture of the system. The events from the WSN are collected, processed and sent to the simulator to be replayed.

The remainder of this paper is structured as follows: We discuss related work in Section 2. Section 3 introduces the design of MILD and Section 4 presents initial evaluation results. Section 5 discusses future directions and concludes.

2 Related Work

Debugging distributed systems has received significant attention in the recent years with the raise of large-scale cloud computing and peer-to-peer networking. A common approach is to collect traces of events and to use their logical relationship in the system to build globally consistent snapshots and to enable replay [1–4]. However, these mainly target Internet based applications and their resource requirements make them not suited for resource constrained, embedded systems such as Wireless Sensor Networks. Nonetheless, their design motivated our work and we carefully designed MILD to adapt them for the use in resource constrained, embedded systems and to ensure minimal intrusion. Others [5, 6] reflect the resource constraints of sensor networks but do not focus on trace-driven replay. In this context, some approaches [5] log each functional call and its parameters to provide the user with a complete view on the system. In this paper, we argue that tracing all the function calls is costly and we show that it is not necessary to trace the them entirely if the state of a node at a given time can be restored and replayed deterministically.

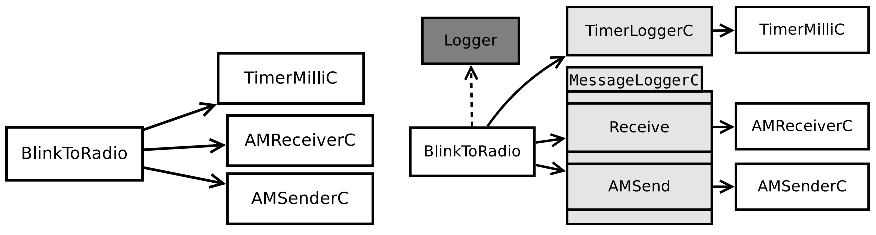
3 The Initial Design of MILD

MILD consists of three building blocks: (1) A logging element on each node, (2) a collection system that collects and combines the traces to a consistent, global view and (3) the replay environment (see Figure 1).

3.1 Distributed Logging for Deterministic Replay

In our design all nodes are equipped with lightweight instrumentation to allow them to record events. To limit the overhead, we only record the events that are of interest to a particular application. For example, when we are debugging a routing protocol, we log in- and outputs such as messages and function calls corresponding to routing. This information is sufficient to deterministically replay any code in a simulator for debugging. Thus, we can recreate the exact program execution in a controlled environment which allows for easy analysis for the program flow and detecting bugs: For example, we can step through the execution of a distributed systems and evaluate the values of individual variables:

- **Wrappers: Tracking In- and Output of Software Modules.** For each software module that shall be included in the logging and replay, we log all in- and output events such as messages or function calls (see Figure 2). Each event is coupled with a logical timestamp to ensure deterministic replay and to track interactions between nodes.
- **Central module: Minimal intrusive state collection.** All events are collected at in a central module. Acting as background task, it allows the collection of event traces via debugging ports at minimal intrusion.
- **Initial State Collection.** To allow us to dynamically enable and disable logging, we store the initial states of module, i.e., its variables, when enabling logging. Loading the initial states in the simulator and then replaying all input to one or more modules ensures deterministic replay.



(a) A generic application (b) The same application with the new wiring

Fig. 2: Wiring to add wrappers (using TinyOS as example). The original modules are indicated in white, the light gray ones are the wrappers and the dark one is an optional interface to the central module.

3.2 Global Event Sorting based on Logical Timestamps

Once all events are collected from the individual nodes via their debugging ports, we utilize their logical timestamps to construct a globally-ordered view on the system (see Figure 3). Since all the events carry a sequence number that is locally unique, sorting the local events of a node is immediate. Events such as radio events have (or can have) a received counterpart on the other nodes. These events are used to obtain a global order of events.

To obtain a partial global ordering of events, every event is treated as a node of a dependency graph. Events that are only local (e.g. timer events) depend on

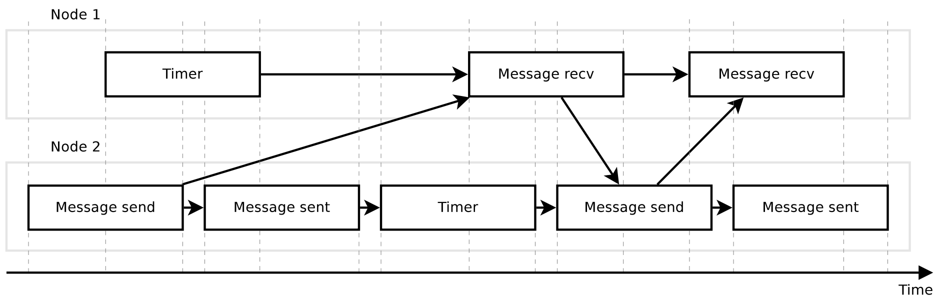


Fig. 3: Events ordering based on dependencies, including the observation that if a node receives two messages, the second one must have been sent after the first was received.

their predecessor in the event log of the node, while others can depend on events on neighboring nodes. One event on a node can only be replayed if the events it depends on have been replayed as well. To generate the dependencies, events are scanned to find the matching events on other nodes. Lost sent messages don't have a corresponding receive event, so they are automatically considered as local events instead. Additionally, we use the recorded output to determine deviations from the replay which indicate subtle system bugs such as buffer overflows etc.

3.3 Deterministic Replay in System Simulation

The final element of MILD is the replay of logs in system simulators. In the replay, we use the wrappers (see Section 3.1) in reverse operation: instead of logging all in- and out-going events of a software module, the wrappers now act as event sources. Thus, feeding from the global event log, the software module now replays the events in the same order as on the deployed system. Utilising the advanced debugging capabilities of modern system simulators (and emulators), allows monitoring of individual variables and stepping through code fragments. Such tasks commonly cause prohibitively high overhead and side effects when performed on deployed systems directly.

4 Evaluation

In this section, we show initial results of our prototype implementation of MILD for TinyOS [7], a widespread operating system for wireless sensor nodes.

4.1 Split-Phase Faults

Most long operations in TinyOS are implemented as split-phase, when, for example, a command to initialise a device is sent from the application to the lower layer, and then an event is sent back to signal that the initialisation is complete.

The authors of [5] use LEACH as a case study to explain how their tracer helped in finding an implementation error.

LEACH is a TDMA-based dynamic clustering protocol. In the example the problem was caused on the cluster head by a timer event trying to send a debug message while another component was sending the information about the cluster to a node requesting access. The bug was caused by the fact that in the timer event, the type of the message was set, although the send itself would fail, the message itself was sent with a different type (because there is only one buffer for the messages, and the original content had been modified by an interleaving event) and acknowledged and ignored by the receiver, which had no function associated with that type of message.

With our implementation the log on the non-head node would show no activity, since the wrapper is placed at high level and the message would be discarded before reaching it, and the head node would show an interleaving of a timer between send and sendDone, and also carry enough information to show that the buffer's content was altered.

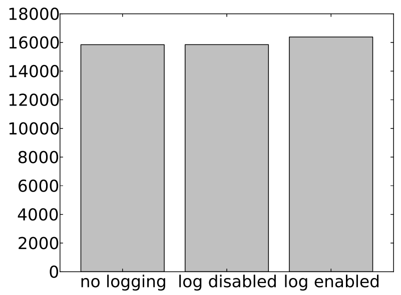
4.2 Performance and memory overhead

In this section we demonstrate that the extra cost added by our instrumentation is acceptable and brings a fixed cost to use the central module, which needs the serial stack, and a variable cost depending on the amount of wrappers used, that mostly depends on the size of the memory area for the log.

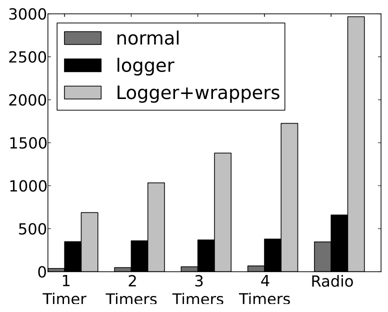
About the speed, as shown in Figure 4a, with logging disabled the overhead is almost zero (3 CPU cycles) and logging one event does not bring much overhead comparing to the effort to generate the event itself. Concerning the memory, Figure 4b shows how allocating large space for logs in the wrappers brings a significant increase in memory usage, while the central module is not so heavy in itself.

5 Conclusion

In this paper we introduced MILD, a lightweight architecture for minimal intrusive logging and deterministic replay of deployed Cyber-Physical Systems. MILD enables (1) logging of events with minimal intrusion of deployed systems, and (2) the deterministic replay of events in controlled environments such as system simulators. As a result, MILD offers deep insights into real-world deployments and allows debugging and testing in realistic settings. We discuss the architecture of and show initial results of our prototype implementation based on TinyOS, a widespread operating system for sensor networks. Future directions include optimization to further reduce logging overhead and the application of MILD in ongoing deployments to gather real-world experience in utilizing MILD.



(a) Comparison of the cycles needed by the application to send one message, in a modified BlinkToRadio application.



(b) RAM usage as reported by the compiler for: blink application with 1,2,3,4 timers and for RadioCountToLeds. Every wrapper can store 40 events.

Fig. 4: Performance and memory evaluation.

Acknowledgements

This work would not have been possible without the contribution of Elad M. Schiller. We would like to thank him for motivating this work and for the many helpful discussions, ideas, and contributions to the system design. Also, the authors would like to thank Xin Zhang for his work on an earlier version of MILD.

References

1. Geels, D., Altekar, G., Shenker, S., Stoica, I.: Replay debugging for distributed applications. In: Proceedings of the annual conference on USENIX '06 Annual Technical Conference. ATEC '06 (2006)
2. Geels, D., Altekar, G., Maniatis, P., Roscoe, T., Stoica, I.: Friday: global comprehension for distributed replay. In: Proceedings of the 4th USENIX conference on Networked systems design and implementation. NSDI'07 (2007)
3. Dao, D., Albrecht, J., Killian, C., Vahdat, A.: Live debugging of distributed systems. In: Proceedings of the 18th International Conference on Compiler Construction: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009. CC '09 (2009)
4. Liu, X., Guo, Z., Wang, X., Chen, F., Lian, X., Tang, J., Wu, M., Kaashoek, M.F., Zhang, Z.: D3s: debugging deployed distributed systems. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation. NSDI'08 (2008)
5. Sundaram, V., Eugster, P., Zhang, X.: Efficient diagnostic tracing for wireless sensor networks. In: Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems. SenSys '10 (2010)
6. Sookoor, T., Hnat, T., Hooimeijer, P., Weimer, W., Whitehouse, K.: Macrodebugging: global views of distributed program execution. In: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems. SenSys '09 (2009)

7. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. SIGOPS Oper. Syst. Rev. **34**(5) (November 2000)