An Effective Approach for Injecting Faults in Wireless Sensor Network Operating Systems

Marcello Cinque, Domenico Cotroneo, Catello Di Martino, Alessandro Testa
Dipartimento di Informatica e Sistemistica, Universita' di Napoli Federico II,
Via Claudio 21, 80125 Napoli, Italy - ph: +39 081 7683812, fax: +39 0817683816
{*macinque,cotroneo,catello.dimartino,a.testa*}*@unina.it*

Abstract

This paper presents an effective approach for injecting faults/errors in WSN nodes operating systems. The approach is based on the injection of faults at the assembly level. Results show that depending on the concurrency model and on the memory management, the operating systems react to injected errors differently, indicating that fault containment strategies and hang-checking assertions should be implemented to avoid spreading and activations of errors.

1 Introduction

This paper presents an effective approach for injecting faults/errors in Wireless Sensor Networks (WSN) operating systems (OS) and investigates how two of the commonly adopted WSN operating systems, namely TinyOS [1] and MantisOS [2], react in presence of realistic faults/errors.
TinyOS [1] is an open-source operating system for WSNs. It features a component-based architecture which enables rapid implementation while minimizing code size. The concurrency model is based on tasks and events.
MantisOS [2] is a light-weight multithreaded operating system for WSNs. Differently from TinyOS, it features a more complex concurrency model, based on preemptively time-sliced multithreading. In addition, semaphores are provided to handle shared resources and to synchronize application threads with device drivers, which also run as threads.
The approach, presented briefly in section 2, is implemented in a framework, named AVR-INJECT [3] in charge of automating injection campaigns and outcome analysis, making it possible to effectively compare WSN OSs under a broad range of errors. Fault injection is used to stress the operating systems.
Several papers have proposed approaches and tools related to fault injection in WSN.
In [4] authors adopt a simulation-based fault injection approach to inject communication faults in the WSN gateways. In [5] the tool *Sympathy* is proposed. It provides facilities to detect and debug failures in sensor networks. It includes an algorithm to localize the root-causes of manifested failures. The presented studies propose ad hoc solutions for injecting faults but no injection tools specifically targeting WSNs. In addition, these studies focus on a high level of abstraction, considering the fault of the whole node or problems in the communication neglecting the potential erratic behavior of damaged/faulty devices. On the other hand, our goal is to study in detail the behavior of the actual OS code running on a WSN node, under realistic low-level faults, such as bit flips.
The paper ends with a presentation of the fault injection campaign on TinyOS and MantisOS. The objective is to compare the fault sensitivity of the two operating systems. Above 2,500 injections are performed in the processor registers and OS code, i.e., data, stack, and code area. We classify campaign outcomes with respect to four classes: crash, hang, unknown errors, and not manifested. The analysis provides quantitative and detailed insight into OSs behavior under faults.

2 The Fault Injection Approach

The basic idea of the approach is to perform the injection by modifying a *target instruction* of the original assembly code (including the operating system code) which aim is to produce the effects of the injected fault on the target instruction by means of a *perturbation function*. Several assembly-level perturbation functions are defined in the framework, depending on *what*, *where* and *when* the fault is injected. Regarding the types of faults to be injected, we assume Single Event Upsets (SEUs), also known as bit flips [6], as fault model, for the following motivations. First, the incidence of SEUs in real microprocessors is increasing as the scale of integration increases and the die voltage decrease. Second, they are more likely to appear in harsh environments which are subjected to environmental disturbances, such as the environments where WSNs are commonly deployed [7].
Finally, SEUs can be easily implemented in the assembly code, and leads to easily understandable results. The framework can be however extended considering other fault models with different lead of abstraction, such as, stuck-at-zero and stuck-at-value. As for the fault location, the framework takes into account three possible choices, based on the architecture of the AVR processor, namely: data memory, code memory, and internal processor registers.

3 Experimental Results

In this section we report some of the results achievable with AVR-INJECT aiming to show the potentiality of

This work has been partially supported by the Regione Campania in the context of the project Misura 3.17 POR 2000/2006, "REMOAM - Reti di sensori per il monitoraggio dei rischi ambientali".

Table 1: Breakdown of campaign outcomes

Outcome	Operating System											
	TinyOS						MantisOS					
	Total		Injection Area				Total		Injection Area			
			Code	Memory	SP	SR			Code	Memory	SP	SR
Crash	**221**	17.8%	34.8%	9.0%	52.5%	3.6%	**204**	15.5%	2.9%	7.8%	85.3%	3.9%
Hang	**338**	27.3%	28.7%	21.3%	50.0%	0.0%	**73**	5.5%	12.3%	0.0%	21.9%	65.8%
Unknown	**176**	14.2%	33.0%	51.7%	6.3%	9.1%	**553**	41.9%	49.4%	21.7%	14.5%	14.5%
Not Manifested	**505**	40.7%	15.8%	28.7%	0.0%	55.4%	**490**	37.1%	29.4%	2.0%	34.3%	34.3%
Total Activations	**1240**	100.0%					**1320**	100.0%				

the implemented fault injection approach. Results are extracted from the execution of two campaigns on TinyOS and MantisOS operating systems. A total of 2,560 fault injections have been performed by the tool in about 12 hours on a Intel P4 machine, CPU Clock 3.5 GHz, 2048 MB RAM, equipped with Linux, kernel 2.6.18. In order to provide a simple yet effective case study for the tool, we selected a simple target application executing a periodic lighting of the leds installed on the sensor node. In the performed campaigns, the fault is randomly introduced in the system by the tool, which generates all the parameters (i.e. bit to flip, activation time, location).

3.1 Outcome Classification

AVR-INJECT is able to classify the outcome of an injection. In particular, the following outcomes have been observed (and classified by the tool) in our experiments:

- *Crash*: the sensor stops working, and no more instructions are executed by the processor.
- *Hang*: the sensor does not deliver any meaningful output, even if it is active.
- *Unknown*: the execution of the instrumented code diverges from the Golden Copy (GC) in one or more intervals, then it returns to work normally.
- *Not manifested*: the target instruction is executed, but it does not cause a visible abnormal impact on the sensor.

Table 2: A subset of TinyOS function analyzed in the performed campaign (N.M.= Not Manifested)

Procedure	Hang				Crash				Unknown				Total	N.M.	Hit ratio
	C	M	SP	SR	C	M	SP	SR	C	M	SP	SR			
CC1000 write() (radio)	4	/	/	0	2	/	/	0	2	/	/	0	8	16	33.0%
__nesc_atomic_end	0	/	6	/	0	/	2	/	8	0	0	/	16	0	100.0%
VirtualizeTimerC StartOneShotAt()	0	32	/	0	0	0	/	0	32	0	/	0	64	32	67.0%
TimerCtrl getInterruptFlag()	48	/	7	/	0	/	1	/	0	/	0	/	56	0	100.0%
VirtualizeTimerC fireTimers()	0	/	/	0	32	/	/	0	0	/	/	0	32	8	80.0%
VirtualizeTimerC fired()	0	13	3	/	24	15	5	/	0	4	0	/	64	0	100.0%
Timer0Async stabiliseTimer()	0	/	3	/	0	/	5	/	0	/	0	/	8	48	14.0%
__nesc_atomic_start	0	/	25	/	1	/	15	/	7	/	0	/	48	0	100.0%
VirtualizeTimerC getNow()	/	8	5	/	/	0	3	/	/	0	0	/	16	0	100.0%
VirtualizeTimerC startPeriodic()	/	0	4	0	/	0	3	0	/	0	1	0	8	32	20.0%
Timer0 isr() (vect 15)	/	8	11	/	/	0	13	/	/	0	0	/	32	48	40.0%
GeneralIOPinP set()	/	0	5	/	/	0	3	/	/	8	0	/	16	0	100.0%

Table 3: A subset of MantisOS functions analyzed in the performed campaign (N.M.= Not Manifested)

Procedure	Hang				Crash				Unknown				Total	N.M.	Hit ratio
	C	M	SP	SR	C	M	SP	SR	C	M	SP	SR			
hardware_id_init	6	/	2	48	0	/	5	0	10	/	0	0	70	1	98.6%
mos_mem_alloc	0	/	0	0	0	/	8	8	64	/	0	0	80	16	83.3%
dispatcher	/	0	0	0	/	0	40	0	/	24	0	8	72	160	31.0%
mos_thread_new	/	0	0	0	/	0	24	0	/	8	32	24	88	8	91.7%
dispatcher_isr (vect 12)	/	0	0	/	/	0	24	/	/	56	0	/	80	8	90.9%
mos_mutex_unlock	/	0	0	/	/	8	8	/	/	0	0	/	16	0	100.0%
mos_sem_post_dispatch	/	0	1	/	/	0	15	/	/	0	0	/	16	8	66.7%
com_init	/	0	5	/	/	0	3	/	/	8	0	/	16	0	100.0%

3.2 Outcome Analysis

Table 1 shows the breakdown of the outcomes of the two fault injection campaigns on TinyOS and MantisOS. TinyOS and MantisOS present a comparable amount of *Crash* (17.8% and 15.5% of the total injections, respectively) and *Not Manifested* outcomes, 40.7% and 37.1% of the total injections, respectively. However, a different perspective is provided by looking at the injection areas. In MantisOS the larger amount of crashes is obtained when injecting errors in the SP register (85.0% of the overall crashes), while in TinyOS crashes are mainly caused by injections in code (34.8%) and in the SP register (52.5%). The observed differences can be explained by the disparity in the way MantisOS and TinyOS manage the memory and for the different concurrency model adopted. More specifically, in MantisOS all threads stack frames are in the higher part of the memory and operating systems data structures (e.g., threads and drivers tables) are in the lower part of the memory. SP register injection primarily result in Stack overflow, i.e., a corruption of the frame pointers stored on the stack. As a result, the threads attend to access memory beyond the currently allocated space for the stack. Consequently, an injection in the SP register may cause an alteration of the OS data structures likely causing a crash. Another possible cause is due to the MantisOS multithreading. Threads use their stack to save the context at a context switch. Hence, an injection in the SP register is very likely to force a wrong store/load of the context on the stack, e.g., a wrong value stored or loaded for the PC register, which we observed to cause a crash in the 98.0% of the cases.

Differently from the crashes, MantisOS and TinyOS present different values for *Hangs* and *Unknown* outcomes, which respectively constitute the 27.3% (338) and 14.2% (176) of total error activations (1240) for TinyOS against the 5.5% (73) and 41.9% (553) accounted for MantisOS (over a total of 1320 activations). In TinyOS the 50.0% of *Hangs* manifests after an injection in the SP register, while injections performed in the code area and in memory account for 29.0% and 21.0%, respectively. The higher hang rate due to SP register errors is mainly due to wrong return address fetched from the stack which is likely to cause infinite loops between the caller and the callee function. This effect is mitigated in Mantis thanks to the time-sliced multithreading which prevents a thread to hang. As another source of hangs, TinyOS is very sensitive to errors injected in code and memory area.

3.3 Operating Systems Analysis

The AVR-INJECT framework enables also the detailed study of OSs by performing the injection during the execution of OSs functions. A subset of TinyOS and MantisOS primitives studied in the performed fault injection campaign is shown in Tables 2 and 3. More specifically AVR-INJECT can inject faults/errors when a targeted function of the OS is executed. This analysis results useful for evaluating the robustness of OS components/functions to injected errors. Tables 2 shows the impact of error injected in the Code Area (C), Memory Area (M), SP register (SP) and SR Register (SR) on TinyOS functions, reporting the hit ratio of the injections (i.e., the number of activated and manifested failures due to faults/errors injected) and the number of not manifested outcomes (N.M.) accounted in the performed campaigns. The results of a similar analysis for MantisOS are shown in Table 3.
For instance, concerning TinyOS (Table 2), the most critical OS functions are those related to Timers and IO management, such as VirtualizeTimerC, Timer0 functions, and GeneralIOPins set() function. Concerning MantisOS (Table 3), the most critical OS functions are those related to the management and scheduling of threads, such as mos_thread_new, dispatcher_isr and dispatcher functions.

4 Conclusions and Future Work

This paper presented the benefits of using assembly level fault injection for comparing operating systems for WSNs. Main findings are: (i) the multi-threading concurrency model of MantisOS decreases the probability of hangs and improves the detectability; (ii) I/O and timers management are critical in TinyOS, while the simplified concurrency management decreases the possibility of fail silent violations, related to failures classified as unknown. Future work will be devoted to develop a JTAG interface for the AVR-INJECT framework, in order to perform real world case study with realistic workload and several nodes.

References

[1] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. TinyOS: An operating system for wireless sensor networks. *Ambient Intelligence by W. Weber, J. Rabaey, and E. Aarts. 2005. Springer; 1 edition (April 19)*, 2005.

[2] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. Mantis os: an embedded multithreaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl.*, 10(4):563–579, 2005.

[3] Marcello Cinque, Domenico Cotroneo, Catello Di Martino, Stefano Russo, and Alessandro Testa. Avr-inject: A tool for injecting faults in wireless sensor nodes. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–8, Rome,Italy, 2009. IEEE Computer Society.

[4] G. Gupta and M. Younis. Fault-tolerant clustering of wireless sensor networks. *IEEE*, 2003.

[5] Nithya Ramanathan, Kevin Chang, Rahul Kapur, Lewis Girod, Eddie Kohler, and Deborah Estrin. Sympathy for the sensor network debugger. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 255–267, New York, NY, USA, 2005. ACM.

[6] J.H. Barton, E.W. Czeck, Z.Z. Segall, and D.P. Siewiorek. Fault injection experiments using fiat. *Computers, IEEE Transactions on*, 39(4):575–582, Apr 1990.

[7] J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson. *Analysis of wireless sensor networks for habitat monitoring.Wireless sensor networks.399–423*. Kluwer Academic Publishers, Norwell, MA, USA, 2004.