Journal of Computers Vol. 32 No. 1, 2021, pp. 1-10
doi:10.3966/199115992021023201001

Towards Understanding Bugs in WSN Applications

Jing-Jing Zhao, Yu-Xia Sun*, Qi Deng, Ming Cheng, Qing-Xuan Kuang

Department of Information Science and Technology, Jinan University, Guangzhou, China
Zjjjndxrjgc@163.com, tyxsun@email.jnu.edu.cn, {799206709, 106855433, 1910602149}@qq.com

Received 23 October 2019; Revised 18 May 2020; Accepted 24 June 2020

Abstract. Software quality of sensor network programs plays a critical role in the application of WSNs (Wireless sensor networks). However, due to severe resource constraint on nodes, node programs typically lack of system protection mechanism for hardware access, synchronization mechanism for concurrency executions, dynamic type detection mechanism, and so on. As a result, WSN applications are error-prone. Thus, it is important to analyze various common bugs in WSN applications and then develop bug-detection techniques for them. Currently, no work has been conducted to comprehensively analyze multiple types of bugs in WSN programs, nor offer a benchmark suite to evaluate bug-detection techniques for them. In this paper, multiple types of bugs, which are collected from research papers or development community of WSN programs, are comprehensively analyzed. The bugs' characteristics are systematically described, including causes, patterns and repairing strategies. To facilitate the bug-detecting research on TinyOS programs, a benchmark suite involving the above bugs is created, which includes buggy programs, fixed programs and test cases.

Keywords: wireless sensor network, TinyOS, bug pattern, benchmark suite

1 Introduction

Wireless sensor networks are important to monitor, collect and transmit data of target objects in IoT (Internet of Things). Typical WSN programs, such as those running on TinyOS system and programmed in nesC language, have the features of component programming, event-driven, two-level scheduling with interrupts and tasks, and so on [1]. Those features enable sensor nodes to collect and process data favorably even with limited resources, but also makes the programs error-prone at least but not completely due to the following reasons: (1) With severely restricted resource on the nodes, the programs have no access protection mechanism provided by the system when it accesses such hardware as node memory, and can easily lead to memory-security bugs; (2) Implemented in non-type-safe languages (e.g. nesC), the programs cannot utilize dynamic type detection mechanisms, which probably causes type-security defects; (3) The concurrent execution model of WSN programs (e.g. TinyOS programs), involving both interrupt preemption and deferred execution of tasks, is complicated, which makes the concurrent execution flows of the programs hard to predicted by the programmers when coding, and results in concurrency bugs. Thus, in order to improve the quality of sensor network programs, it is important to study the bug characteristics of WSN programs, and develop bug-detection techniques based on the characteristics.

Some researchers have depicted the characteristics of some types of bugs in WSN programs and proposed bug-detection techniques (see Sect. 2). However, most of them focus on only few types of bugs. To the best of our knowledge, no work has been conducted to comprehensively analyze of multiple types of bugs in WSN programs. Empirical studies are important to the research of software testing and debugging technology, and benchmark suites are always needed to evaluate the technologies [2]. However, the existing bug-detection works for TinyOS programs usually studies no more than three program defects, and use no more than three self-selected tested programs for each defect. Thus, in this paper we will comprehensively study multiple types of bugs in WSN programs, and construct a

* Corresponding Author

benchmark suite involving buggy TinyOS programs with the above types of bugs and the test cases.

The main contributions of this paper are as follows:

(1) Analyze the characteristics of 15 common bugs of multiple bug types in WSN programs, propose a set of bug patterns to depict the bugs, and present the repair strategies for the bugs.

(2) Create a benchmark suite to facilitate the bug-detecting research on TinyOS programs. For each target bug type, the suite contains a buggy subject program, the correct program, and the description on the test inputs and outputs.

2 Related Work

2.1 Bug Characteristics of WSN Applications

Prior researches have been conducted on depicting the characteristics some types of bugs in WSN programs, and detecting the bugs. Tchamgoue et al. [3-4] described the characteristics of the following concurrency bugs in WSN programs: data races, atomicity and order violations, and deadlocks. Li et al. [5] developed a tool called T-check to find safety and liveness errors in sensor network applications using random walks and model checking. Zhou et al. [6] described transient bugs due to data race, and developed a tool named Sentomist to find transient bugs. Sasnauskas et al. [7] developed a concolic testing and symbolic execution tool called kleenet to detect interactive bugs caused by such non-deterministic events as node reboots or packet duplicates. Tchamgoue et al. [8] focused on data race bugs and developed a tool to detect data race bugs. Midi et al. [9] proposed a tool named nesCheck to detect memory security bugs. However, only few researchers studied bug patterns of WSN applications. For an example, Khan et al. [10] proposed a pattern extraction technology to generate symbolic bug patterns, and utilized the patterns to disclose bugs. For another example, Sun et al. [11] proposed three bug patterns to characterize concurrency errors due to interleaving access between interrupt process instances, put forward a dynamic analysis method and an automatic tool to find and locate concurrent bugs in WSN programs. Different from the above work, this paper focuses on the characteristics of multiple types of bugs in WSN applications and proposes a set of bug patterns to depict them, where the bugs include concurrency bugs, task-related bugs, memory security bugs, and so on.

2.2 Bug Characteristics of Other Programs

There have been works on bug characteristics of c/c++ programs [12-14]. For example, Lu et al. [13] launched the first comprehensive study on the real concurrent bug characteristics of object-oriented programs. Some researchers studied bug characteristics of Android programs, and the bugs involve performance bugs [15] and compatibility bugs [16]. Wan et al. [17] conducted the first empirical study on the bug characteristics of Blockchain systems, and analyzed the frequency distribution of bug types in different projects and different programming languages (such as C, Go, JavaScript). The relationship between bug type and bug repair time is also studied. Franco et al. [18] comprehensively studied 269 real numerical bugs from 5 numerical software libraries. Clow et al. [19] studied portability bugs in applications on different platforms such as desktop, server, and mobile devices. The existing studies on bug characteristics of various programs have promoted the testing and debugging techniques of the corresponding programs, and enlightened the design of novel programming language. However, the bug characteristics of WSN applications have not been studied comprehensively.

3 Bug Categories

In this section, we categorize the bugs of TinyOS applications into the following five groups: concurrency bugs, task-related bugs, interface-related bugs, memory-safety bugs, and arithmetic bugs, as the first column shows in Table 1. The example bugs of the above categories have been reported in academic papers ([4-7, 11, 22-25, 28]) or TinyOS development community ([26-27, 29-31]). In Table 1, the following two columns list each bug's subcategory and the source paper or webpage that reports the example bug. The columns of "Cause Analysis" and "Fixing Strategy" denote whether the source paper or the webpage offers bug cause discussions and bug fixing analyses or not, with the symbol "√" (Yes)

and "-" (No), respectively. The last column "Subject" indicates whether a buggy subject application that contains the bug is publicly available or not.

Table 1. Bugs Reported by Developers and Researchers

Bug category	Sub-category	Reported by	Cause Analysis	Fixing Strategy	Subject
Concurrency bugs	Atomicity violation	[4, 11, 22]	√	√	√
	Order violation	[4, 22-23]	√	-	√
	Invalid data race	[4, 6, 24]	√	-	√
	Deadlock	[4, 25]	√	-	√
	Conflict radio access	[11]	√	√	√
	Erroneous split-phase synchronization	[11, 31]	√	√	√
Task-related bugs	Failed task-posting	[26]	-	-	-
	Long task-running	[27]	√	√	-
Interface-related bugs	Incorrect many-to-one binding	[28]	-	√	-
	Erroneous interface-initialization	[28]	-	-	-
Memory-safety Bugs	Array access out-of-bound	[5, 7, 29]	√	√	√
	Null pointer	[5]	√	√	-
	Mismatched data type	[7]	√	√	-
	Memory leak	[30]	√	√	-
Arithmetic Bugs	Divided by 0	[7]	√	√	√

4 Bug Patterns and Fixing Strategy

Bug pattern is an abstraction of recurrent defects with common characteristics [12]. Analyzing the bug patterns of WSN programs can help developers better understand the characteristics and triggers of defects, and thus to avoid the bugs while programming. Moreover, automatic testing and debugging tools can be developed based on the bug patterns and the common fixing strategies. In this paper, we will summarize the bug patterns of 15 sub-categories of TinyOS program bugs, and investigate the repair strategies for the bugs.

4.1 Concurrency Bugs

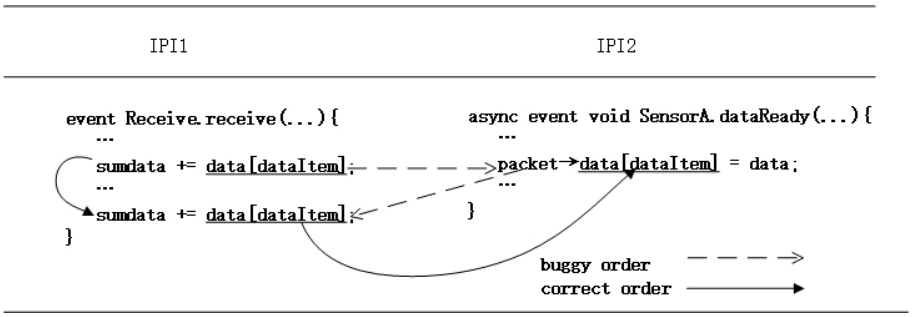
TinyOS programs are typically event-driven, and its scheduling and synchronization mechanism are different from the traditional programs [20]. For instance, TinyOS programs lack standard concurrent synchronization mechanism, such as lock, semaphore and so on, due to the serious limitation of node resources. When a TinyOS program runs, the executions of various Interrupt Processing Instances (IPI) are complicatedly interlaced and hard to predict [11, 21], and possibly lead to atomicity violation, order violation, invalid data race, deadlock, conflict radio access, and so on.

A concurrency bug occurs when an unexpected operation sequence operates on a shared resource, such as a shared memory location or a shared radio. Thus, bug patterns of concurrency bugs can be depicted in the form of unexpected operation sequences, as Table 2 shows. Table 2 consists of five types of concurrency bugs, and columns 2 to 4 denote the types, patterns and descriptions of the bugs, respectively.

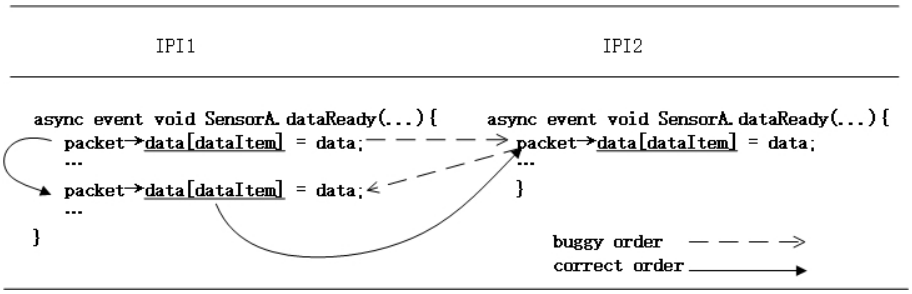
The concurrency bugs on a shared pointer to a package/array can be divided into two sub-types that are depicted with two patterns, namely "R1 W2 R1" and "W1 W2 W1", respectively. A concurrency bug on a shared Boolean variable can be depicted with the pattern "W1 W2 W1". To exemplify the above three bug patterns, Fig. 1 illustrates three segments of buggy codes. In the figure, buggy execution orders are represented with dotted lines and correct orders are denoted with solid lines. In addition, all the above bugs can be fixed by applying atomic operators to the R/W operations on the shared variables of IPI1, so as to protect IPI1' execution from IPI2's preemption. The sample code for other types of concurrency errors can refer to the examples in [11].

Table 2. Concurrency Bug Patterns

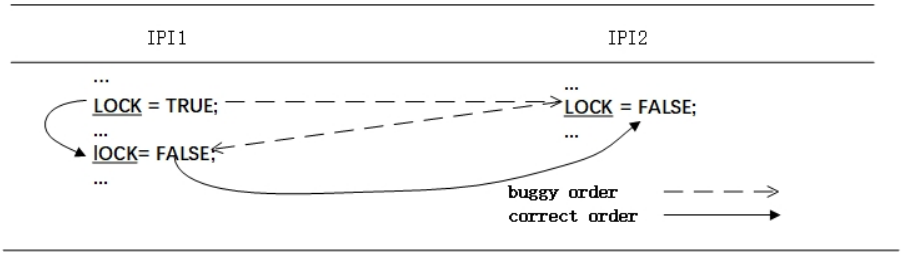
No.	Concurrency bug type	Patterns	Description
1	On shared pointer to package/array	R1 W2 R1	The values in package/array are expected to be read consecutively by IPI1, but polluted by IPI2
		W1 W2 W1	The values in package/array are expected to be written consecutively by IPI1, but polluted by IPI2
2	On shared Boolean variable	W1 W2 W1	The value in Boolean variable are expected to be updated consecutively by IPI1 (e.g. with T and F, respectively), but polluted by IPI2 (e.g. with F)
3	on split-phase synchronization	Call1 Call2 Signal1	The operation launched by the second Call is expected to be completed, but failed with no Signal.
4	On shared ordinary variable	W1 W2 R1	The value written by IPI1 is expected to be read by itself, but polluted by IPI2
		W1 W2 R2	The value written by IPI1 is expected to be read by itself, but lost
5	On shared radio	Send1 Send2 SendDone1	The second transmission-operation is expected to be completed, but failed due to the busy radio.



(a) Bug pattern on shared pointer (R1 W2 R1)



(b) Bug pattern on shared pointer (W1 W2 W1)



(c) Bug pattern on shared Boolean variable (W1 W2 W1)

Fig. 1. Sample codes of concurrency bugs

4.2 Task/Interface-Related Bugs

Task-Related Bugs. To ensure the real-time response of nodes, a WSN program typically divide a time-consuming computation into several computation tasks and submit them to the system task queue, and tasks will be scheduled by the OS in a first-in-first-out order. However, if a task is not appropriated coded and thus takes long time to execute, program defects related to tasks may occur. As the code shown in Fig. 2, a task named computeTask involves a loop that will take long time to execute, which defers the execution of subsequent tasks in the queue and consequently degrades the system response. Furthermore, during the execution of the task, if a subsequent computation task with the same task name is also formed and submitted to the task queue, the latter task cannot be submitted to the task queue, and a task posting failure occurs. In order to fix such a bug, the computeTask computation needs to be broken down into short-term subtasks, and then be posted to the task queue in turn. In this way, no task can occupy the processor for too much time and all the tasks can be completed in time.

```
uint32_t i ;
task void computeTask(){
    for(i = 0 ; i < 4000001 ; i++){
        ...
    }
}
```

Fig. 2. Sample code of Long task running bug

Interface-Related Bugs. Many WSN programs such as TinyOS programs adopt component programming, where incorrect component binding can lead to interface-related program defects, such as erroneous interface-initialization and incorrect many-to-one binding. Since each module is responsible for initializing and starting all the sub-modules it uses, the underlying interfaces shared by the sub-modules are repeatedly initialized and started, which not only affects the efficiency of program execution, but also easily introduces defects. For example, when a wireless module CC1000RadioIntM of TinyOS1 program is initialized on under the mica platform, a one-time timer will be started. However, if the wireless module is initialized frequently, the startup operation of the one-time timer will be triggered repeatedly. As a result, the timer's basic timing function cannot work well, and the events to be triggered by the timer cannot be handled within the expected time interval.

Some WSN programs utilize the execution mode called "Split-Phase Operations" to divide the calling and the execution job into two separate stages. In the phased operation, multiple consumer components can form a "many-to-one" binding relationship through the same interface and the same provider component. When a consumer component invokes a command in the interface, it triggers not only its own corresponding event handler but also the ones of other consumer components. As demonstrated in Fig. 3, two consumer components, named user1 and user2, form a "many-to-one" binding relationship with the provider component AMSendC through the interface AMSend, when user1 calls the send method of the AMSend interface, two SendDone event handlers of user1 and user2 are triggered at the same time, where the execution of user2's SendDone function is unexpected, and thus causes an error. To avoid this type of bug, at the very beginning of SendDone function, a judgment statement needs to be added to tell whether the sent message's type is consistent with the cache message's type or not.

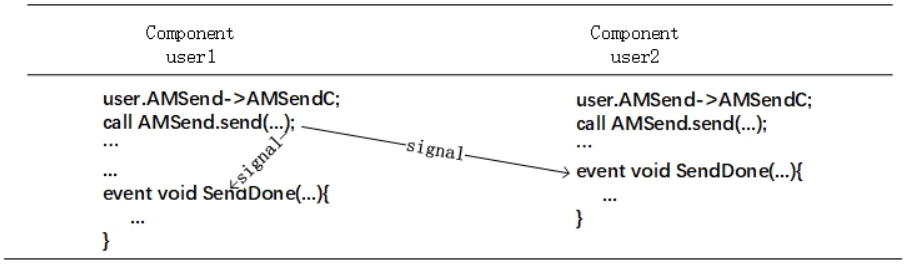


Fig. 3. Sample code of incorrect many-to-one binding bug

4.3 Other Bugs

Memory-Safety Bugs. WSN programs are not protected by the access protection mechanism due to the restricted limitation of node resources, and they cannot use dynamic type detection mechanism. Thus, they are prone to memory security defects, such as out-of-bound array access, null pointer exception, data type mismatch, and so on.

Fig. 4 illustrates the sample code of an out-of-bound access bug. When the timer reaches a timing cycle, the Timer.fired function is expected to run to start the sensing module to collect environmental data, and when the array is full, the index needs be reset to 0; upon successful data acquisition, the Read.readDone event is triggered to save the data to the readings array. However, more than one consumer component might be bound to the provider component through the same Read interface, and if one consumer component calls the Read.read command, all the consumer components' Read.readDone event handlers will be triggered to put data into the array. At this moment, if the timing cycle does not reach and thus the Timer.fired is not triggered, the array index will be not reset to 0. As a result, the array index will be increased by multiple Read.readDone event handlers and finally out of bound.

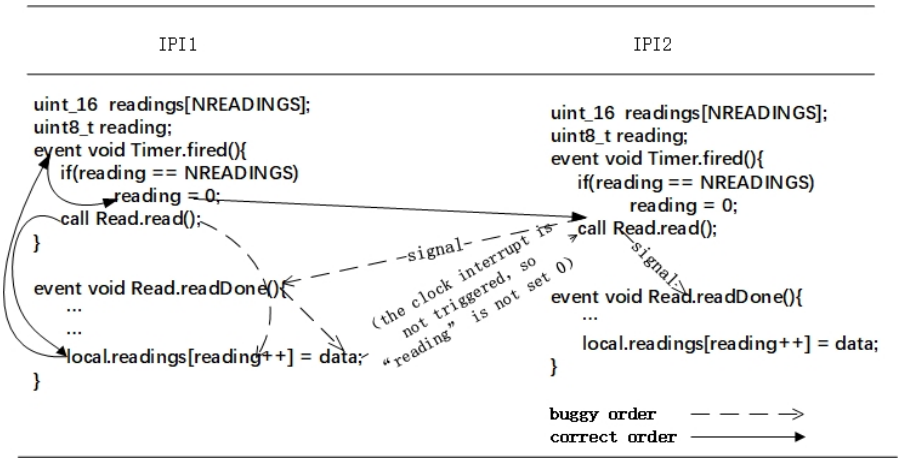


Fig. 4. Sample code of array access out of bound bug

Fig. 5 shows the sample code of a null-pointer exception bug in a distribution protocol (DIP) of TinyOS, which is used to reliably delivering messages to each node in the network. In Fig. 5, the DipDecision.send command implements the function of sending a SummarySend packet and calling the getPayLoadPtr command to obtain the payload pointer dmsg to the packet. However, when the packet is in the busy state (e.g. when other programs are using this packet), the payload pointer dmsg to the packet is empty, which causes a null-pointer reference error. The bug can be fixed by telling whether the pointer is null or not before referring to the pointer.

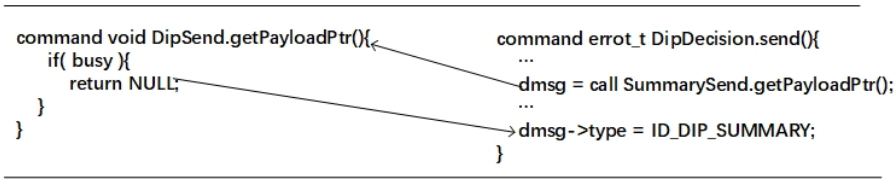


Fig. 5. Sample code of null-pointer exception bug

A sample code of mismatched data type bug is depicted in Fig. 6, where the Receive.receive event handler is triggered when a node receives a packet, and the pointer btrpkt of type BlinkToRadioMsg refers to the message payload. However, when the payload type of the packet is inconsistent with the data type BlinkToRadioMsg, the defect "mismatched data type" will occur. This bug can be fixed by casting the data type of payload as the type of pointer btrpkt.

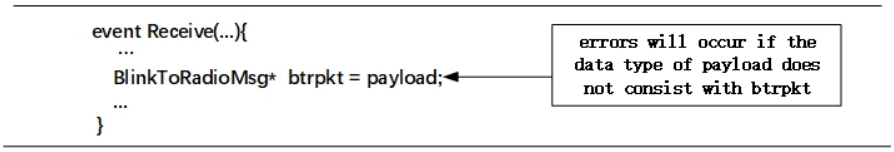


Fig. 6. Sample code of mismatched data type bug

Arithmetic Bugs. A sample code of division-by-zero bug is shown in Fig. 7, where a division operation is used to calculate the ratio. But when the divisor is 0, a division-by-zero bug will be introduced. This bug can be fixed by judging whether the value is 0 or not before the division operation.

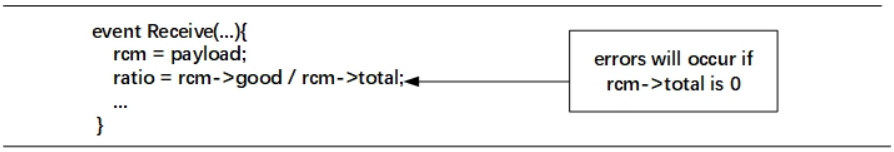


Fig. 7. Sample code of divide by 0 bug

5 Benchmark Suite

To facilitate the research on testing and debugging WSN programs, we create a benchmark suite consisting of a group of TinyOS programs with 15 bugs of the bug types discussed in Section 3 and 4, which can be obtained by contacting the author. For each buggy program, the suite also offers a fixed version (namely a correct program), and the bug's description (i.e., test cases) including how to induce the bug (i.e., inputs) and how to tell the symptom (i.e., the outputs). The inputs involve sensor's sampling rate, timer's timing value, packets received and so on. The outputs include output strings, LED state switching, packets sent, debugging information and so on.

Next, we illustrate how to run the benchmark programs and observe the bug symptoms with two buggy programs in the benchmark suit. We utilize a WSN simulator, called Avrora (an extensible simulator with precise clock characteristics [32]), to run the programs and show the outputs. The first program (named bm_NullPointer) contains a null-pointer exception bug which wrongly leads to repeated sending of a packet. The Avrora command to run the bm_NullPointer program is "platform=mica2 -monitors=serial, packet -ports=1:0:2390 -stagger-start=6140 -seconds=4 -simulation=sensor-network -nodecount=1, 1 np.elf bs.elf". Fig. 8 shows the running results of the program where the repeatedly sent packages are highlighted. The second program (named bm_TestTaskPost) includes a task-posting related bug which manifests as wrong LED switching. The Avrora command to run the bm_ TestTaskPost program is "-platform=mica2 -monitors=serial, packet, c-print -ports=1:0:2390 -simulation=sensor-network -nodecount=1, 1 -seconds=10 tp.elf bs.elf". As Fig. 9 reveals, the running result includes a switching of LED0 that is expected to be the switching of LED1.

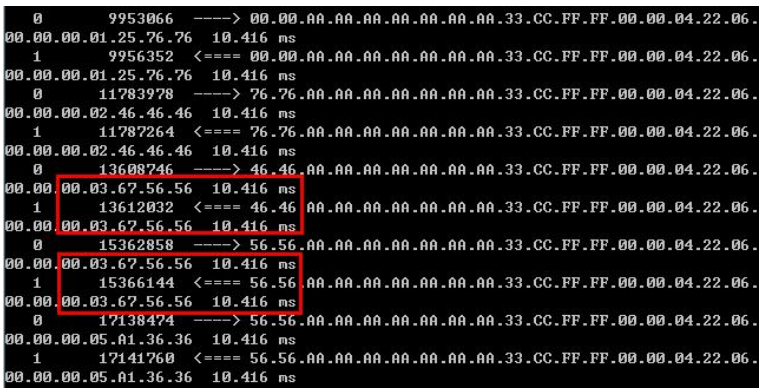


Fig. 8. Repeated packet sending by bm_NullPointer program

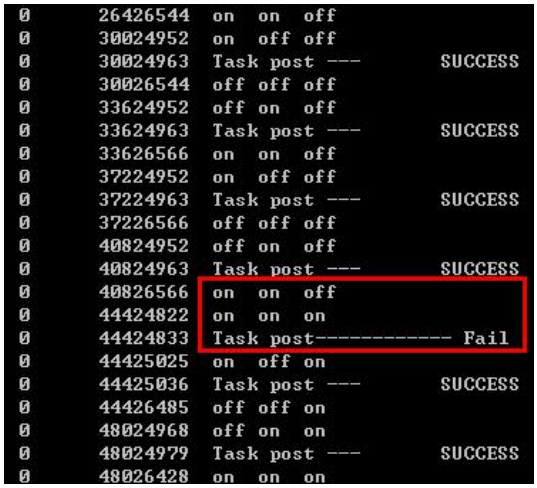


Fig. 9. Wrong LED switching by bm_TestTaskPost program

6 Conclusion and Future Work

Due to limited resource constraint on nodes, WSN programs typically lack of mechanisms for system protection of hardware access, concurrency synchronization, and dynamic type detection. Thus, defects are prone to be introduced in WSN program. To improve the software quality of WSN programs, it is necessary to study the bug characteristics of WSN programs, and then develop corresponding testing and debugging techniques to find and fix the bugs. In this paper, we categorize common bugs in WSN programs into five types, analyze their bug characteristics, and present five groups of bug patterns. We also propose a benchmark suite for studying the above bugs and the potential bug-detection techniques of TinyOS programs, which involves buggy programs and test cases. In the future, we will collect more buggy programs on more WSN OSs and platforms, conduct further analysis on the bug characteristics, and develop bug-detection techniques based on the analyses.

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China under Grant 61402197, Grant 61772211.

References

[1] TinyOS, TinyOS Home Page. <http://tinyos.stanford.edu/tinyos-wiki/index.php>, 2019 (accessed 10.09.19).

[2] S.-E. Sim, S. Easterbrook, R.-C. Holt, Using benchmarking to advance research: A challenge to software engineering, in: Proc. 2003 International Conference on Software Engineering, 2003.

[3] G.-M Tchamgoue, O.-K. Ha, K.-H. Kim, Y.-K. Jun, A taxonomy of concurrency bugs in event-driven programs, in: Proc. 2011 International Conference on Advanced Software Engineering and Its Applications, 2011.

[4] G.-M. Tchamgoue, K.-H. Kim, Y.-K. Jun, Testing and debugging concurrency bugs in event-driven programs, International Journal of Advanced Science and Technology, 40(2012) 55-68.

[5] P. Li, J. Regehr, T-check: bug finding for sensor networks, in: Proc. Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks, 2010.

[6] Y. Zhou, X. Chen, M.-R. Lyu, J. Liu, Sentomist: Unveiling transient sensor network bugs via symptom mining, in: Proc. 2010 IEEE 30th International Conference on Distributed Computing Systems. IEEE, 2010.

[7] R. Sasnauskas, O. Landsiedel, M.-H. Alizai, KleeNet: discovering insidious interaction bugs in wireless sensor networks before deployment, in: Proc. Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks, 2010.

[8] G.-M. Tchamgoue, K.-H. Kim, Y.-K. Jun, Verification of data races in concurrent interrupt handlers, International Journal of Distributed Sensor Networks, 9(11) (2013), 953593.

[9] D. Midi, M. Payer, E. Bertino, Memory safety for embedded devices with nesCheck, in: Proc. Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, 2017.

[10] M.-M.-H. Khan, T. Abdelzaher, J. Han, H. Ahmadi, Finding symbolic bug patterns in sensor networks, in: Proc. 2009 International Conference on Distributed Computing in Sensor Systems, 2009.

[11] Y. Sun, S.-C Cheung, S. Guo, Disclosing and Locating Concurrency Bugs of Interrupt-driven IoT Programs, IEEE Internet of Things Journal, 2019.

[12] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou, C. Zhai, Have things changed now?: an empirical study of bug characteristics in modern open source software, in: Proc. Proceedings of the 1st workshop on Architectural and system support for improving software dependability, 2006.

[13] S. Lu, S. Park, E. Seo, Y. Zhou, Learning from mistakes: a comprehensive study on real world concurrency bug characteristics, in: Proc. 2008 ACM SIGARCH Computer Architecture News, 2008.

[14] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, C. Zhai, Bug characteristics in open source software, Empirical Software Engineering, 19(6) (2014), 1665-1705.

[15] Y. Liu, C. Xu, S.-C. Cheung, Characterizing and detecting performance bugs for smartphone applications, in: Proc. 2014 International Conference on Software Engineering, 2014.

[16] L. Wei, Y. Liu, S.-C. Cheung, Taming Android fragmentation: Characterizing and detecting compatibility issues for Android apps, in: Proc. 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), 2016.

[17] Z. Wan, D. Lo, X. Xia, L. Cai, Bug characteristics in blockchain systems: a large-scale empirical study, in: Proc. 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), 2017.

[18] A. Di Franco, H. Guo, C. Rubio-González, A comprehensive study of real-world numerical bug characteristics, in: Proc. Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, 2017.

[19] J.-A. Clow, Comprehensive Study of Portability Bug Characteristics in Desktop and Android Applications, 2018.

[20] X. Wu, Y. Wen, L. Chen, W Dong, Data race detection for interrupt-driven programs via bounded model checking, in: Proc. 2013 IEEE Seventh International Conference on Software Security and Reliability Companion, 2013.

[21] Y. Sun, S. Guo, S.-C Cheung, Analyzing and Disentangling Interleaved Interrupt-driven IoT Programs, IEEE Internet of Things Journal, 2019.

[22] J. Regehr, Random testing of interrupt-driven software, in: Proc. Proceedings of the 5th ACM international conference on Embedded software, 2005.

[23] Z. Lai, S.-C. Cheung, W.-K. Chan, Inter-context control-flow and data-flow test adequacy criteria for nesC applications, in: Proc. Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, 2008.

[24] G.-M. Tchamgoue, K.-H Kim, Y.-K Jun, Dynamic race detection techniques for interrupt-driven programs, in: Proc. 2012 International Conference on Future Generation Information Technology, 2012.

[25] J. Yang, M.-L. Soffa, L. Selavo, K Whitehouse, Clairvoyant: a comprehensive source-level debugger for wireless sensor networks, in: Proc. Proceedings of the 5th international conference on Embedded networked sensor systems, 2007.

[26] Tinyos-help, Failed task-posting. <http://tinyos-help.10906.n7.nabble.com/Two-questions-about-tasks-td17846.html>, 2019 (accessed 10.09.19).

[27] Tinyos-help, Long task-running. <http://tinyos-help.10906.n7.nabble.com/Tasks-dilemma-td1585.html>, 2019 (accessed 10.09.19).

[28] W. Archer, P. Levis, J. Regehr, Interface contracts for tinyos, in: Proc. 2007 6th International Symposium on Information Processing in Sensor Networks, 2007.

[29] Tinyos-help, Array access out-of-bound. <http://tinyos-help.10906.n7.nabble.com/Two-bugs-in-Dip-Protocoltd1570 0.html>, 2019 (accessed 10.09.19).

[30] Tinyos-help, Memory leak. <http://mail.millennium.berkeley.edu/pipermail/tinyos-devel/2009-January/003594.html>, 2019 (accessed 10.09.19).

[31] Tinyos-help, Erroneous split-phase synchronization. <http://www.mail-archive.com/tinyos-help@millennium.berkeley.edu/msg07568.html>, 2019 (accessed 10.09.19).

[32] B.-L. Titzer, D.-K. Lee, J. Palsberg, Avrora: Scalable sensor network simulation with precise timing, in: Proc. 2005 International Symposium on Information Processing in Sensor Networks, 2005.