ATLANTIS PRESS
Advances in Computer Science Research, volume 44
3rd International Conference on Wireless Communication and Sensor Network (WCSN 2016)

A Light-weight Multilevel Recoverable Container for Event-driven System: A Self-healing CPS Approach

Peng Zhou[1,2]
[1]School of computer science and technology, Harbin Institute of Technology
Harbin, Heilongjiang, China.15000
[2] LIMOS, UMR 6158 CNRS, Blaise Pascal University Clermont-Ferrand II
63173, Aubière CEDEX, France.
zhoupeng@ftcl.hit.edu.cn

De-Cheng Zuo
School of computer science and technology, Harbin Institute of Technology
Harbin, Heilongjiang, China.15000
zdc@ftcl.hit.edu.cn

Kun-Mean Hou
LIMOS, UMR 6158 CNRS, Blaise Pascal University Clermont-Ferrand II
63173, Aubière CEDEX, France.
kun-mean.hou@isima.fr

Zhan Zhang
School of computer science and technology, Harbin Institute of Technology,
Harbin, Heilongjiang, China.15000
zz@ftcl.hit.edu.cn

Hong-Ling Shi
School of computer science and technology, Harbin Institute of Technology,
Harbin, Heilongjiang, China.15000
hongling.shi@isima.fr

Abstract—Cyber Physical Systems (CPS) is regarded as a new technological revolution, which tightly integrates computing, communication, and control technologies, to build a kind of smart networked distributed embedded control system. CPS is designed to interact autonomously with the volatile external environment, which implies that the requirements is constantly changing during run-time. So guaranteeing the reliability of system becomes extremely difficult. Flexible self-healing mechanisms are needed urgently to improve the reliability and availability of CPS. This paper presents a light-weight container-based virtualization for event-driven CPS. By providing a unique run-time stack for each application, the container isolates faults and limits the effect of failures. Furthermore, a multilevel fault detection and recovery method is integrated to protect applications and to limit the fault propagation. And the analysis shows the container has very low memory footprint (939 bytes) and constant performance overhead. Also the testing manifests that the multilevel recovery is high reliable on WCET violation failure recovery even if the application is not well designed or malicious.

Keywords-CPS; Container-based virtualization; Self-Healing reliability; Availability; Fault Detection

I. INTRODUCTION

Cyber Physical Systems (CPS) is a kind of smart distributed networked embedded control system that interacts with the physical world. Generally, CPS is large scale system and deployed in an open, complex and volatile environment. The requirements of CPS is constantly changing during run-time, so it's almost impossible to test all cases of the whole system before deployment, which leaves none of measure except self-healing on run-time. Lots of researchers highlight that the reliability or dependability is a big issue for CPS applications [1][5]. However as far as we know, only a few practical solutions are proposed for tiny systems [5]. The main reasons are limited resources (such as limited capacity of MCU, memory, network, etc.), critical constraints (e.g., power budget, real-time, global reference time, accuracy of data/decisions, etc.), and the variable complex environments (e.g., unpredictable emergencies, the effect of humans, etc.).

IBM describes a concept of autonomic computing system (ACS), it's a self-managing system that includes 4-Self characteristics: *Self-configuring*, *Self-healing*, *Self-optimizing*, *Self-protecting* [6]. The *Self-healing* here means that a system detects, diagnoses, and recovers automatically from damage that occurs during its life cycle. But unfortunately, the complex strategies of self-* increase the logical complexity which is harmful to system reliability. In a sense, failure detector/manager is also a kind of CPS application whose purpose is to guard common applications rather than physical world. It also has to face the same problems that common applications suffer from. To build a reliable CPS, reliable failure detection and recovery components are needed. Yet, to guarantee the reliability of such failure management components, should we build additional components? It

becomes an endless loop nightmare to engineers. Sha has made a conclusion that using simplicity to control complexity is the way to build a reliable system [7]. And also Jackson highlights the importance of decoupling and simplicity to system design [8]. So the problem can be refined to design a simple self-healing methods.

In our opinion, reliability of CPS is not an isolated issue, it should be taken into account with self-adaptivity and flexibility together. For CPS, some simple but universal mechanism of reliability should be devised to achieve self-healing. Isolation is the basis method to block fault propagation between components. An ideal solution to minimize interference between sensors/actuators is building each sensor and actuator with an isolated embedded system and sharing as few resources as possible. But the drawback is that such solution costs too much and seems unpractical, and the limited bandwidth of wireless network and energy supply also restrict the potential of such solution. Virtualization is a practicable technology with high universality to isolate applications and keep the flexibility of services [9].

The organization of this paper shows as follow. Section 2 discusses about related work on CPS reliability and virtualization. Section 3 introduces the model of CPS application with event-driven mealy Finite State Machine (FSM) and the difference between the existing container and our design at the model level, and also the framework of our container is proposed. Section 4 details the architecture of our light-weight container, the multilevel failure detection and recovery mechanism and the implementation. Section 5 shows the analysis of performance and reliability, and also testing results are presented. Finally, a conclusion is made in section 6, and also the further work is discussed.

II. RELATED WORK

With the development of information and communication technology, the applications of embedded system become more and more complex. Compared with WSN, CPS doesn't only sense the physical world but also analysis and interact with it. However, to physical system, what's done can't be undone, the physical changes can't be revoked, which makes reliability and safety extremely important in CPS. Redundancy and failure isolation are the two traditional basic mechanisms to enhance the reliability. And fault detection, failure prediction, fault diagnose and recovery are always implemented together. Lots of papers have analyzed the causes of failures and challenges of reliability/dependability in CPS [10][13]. Gunes V et al. make a detail investigation about CPS's attributes and requirements, discus dependability and its sub-attribute: maintainability, availability, safety, reliability one by one, and also lots of other X-abilities [12], but no solution is proposed. Lee et al. introduce a PTARM microarchitecture and a deterministic CPS modeling paradigm called Ptides [2], [13]. Lee's research is insightful on correctness, it tries to improve the reliability by making less mistakes, yet the research on reliability mechanisms seems feeblish.

A. *Reliability solution of CPS*

Redundancy is useful to detect faults and to build a fault-tolerance system. And a lot researches on redundancy are presented. Such as Kim J et al. build a reliable architecture with redundant components [3], and Hu Y-L et al. enhance the system with redundant data [14]. However, redundancy seems less effective for CPS than general system, because the failure of CPS is caused by the volatile environment. All the redundant nodes suffer the same problem and will get the same error result which means the redundancy mechanism is failed.

What's more, redundancy costs too high for a small battery-powered devices. However, it is still a good practice to apply redundancy on the node level because CPS always contains massive of redundant sensor nodes and actuator nodes. J.Valverde introduces a dynamically adaptable bus architecture called ARTICo [15]. It uses a reconfigure technology based on Double or Triple HW Module Redundancy (DMR or TMR).

Improving the reliability with modeling methods or designing new self-diagnose architecture are other popular ideas in CPS. Teodora Sanislav proposed an event-driven multi-agent model to explore the method of quantitative evaluation [16]. A data-centric runtime monitoring platform named ARIS is described in Ref.[17]. It detects abnormal behaviors through data thresholds checking, patterns matching and machine learning classification. In Ref.[18], both static and dynamic co-design methods are explored and evaluated, the authors take both energy and reliability into account. Sanislav T et al. describe an ontology-driven environmental monitor to assure system dependability; Failure Mode and Effects Analysis (FMEA) is used to quantify the risk level of failure [19]. It is a self-healing method which is flexible and scalable in theory, the only fly in the ointment is that no testing result has been proposed and it sounds that high memory and performance overhead will be introduced. S. Krishnamurthy et al. present a Bayesian network approach to learn, detect and isolate the abnormal behaviors [20]. It is an ideal (networked) system level solution with self-learning and self-healing without considering the network delay and data noise effect which is a main challenge of CPS.

B. *Fault detection and isolation*

Fault Detection and Isolation (FDI) has been well researched. J. Korbicz et al. detailed investigate the fault detection and isolation methods [21]. Those detection methods can be classified into 3 types: 1) system mode/state based, 2) (data) testing based, 3) rule/behavior based. The isolation methods also can be classified into 4 types based on the level where applied: 1) hardware level, 2) binary level, 3) components level, 4) system/node level. Wander. A et al. introduce FDI strategies on-board spacecraft and highlight the challenge of feasibility which mainly caused by complexity [22].

Time is important resource to CPS because the action of physical process should be taken at right time and finished in time. As Lee argued in Ref.[13], computation

time is closely related to the correctness. And the timing behavior becomes increasingly uncontrollable with the increasing complexity of system. Systems get stuck easily in the open and variable environment, especially without supervisor. Worst Case Execution Time (WCET) is a traditional strategies to handle this problem [23]. As a kind of rule/behavior based detection, it requires little the internal detail logic of applications which makes it widely used. There are two main ways lead to the WCET based detection, heartbeat-based and timer-based. Generally, heartbeat takes effect by sending heartbeat signals/packages each other, so it needs at least a pair of components. Yet, like watchdog, timer-based detector is designed for single component. It is simpler but should select the threshold of timer carefully.

C. *Virtualization based isolation on embedded system*

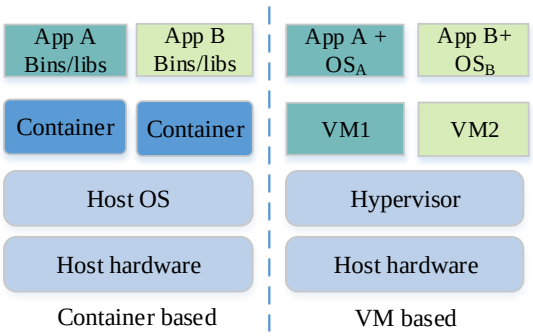

Figure 1. Two kinds of virtualization

Virtualization is a key technology of Cloud Computing. It can consolidate services, achieve load-balancing, and manage the power, firewall service and running different operating systems [24]. From fault isolation aspect, virtualization is an ideal isolation solution. It is universal and can provide a unique run-time environment for each application/subsystem, which can protect normal and healthy applications from the influence of a failed application and minimize the risk of system failure. Currently, there are two kinds of typical virtualization, one is virtual machine based (VM-based) and another is container-based. The frameworks of two virtualization are showed as Figure 1.

Most current researches on embedded system are VM-based virtualization. Heiser. G introduces two kinds of approaches, one is bare metal virtualization which is a typical VM-based technology. And another is hosted virtualization which needs a host operating system (OS) and the hypervisors execute on the host OS [24]. Compared with VM-based virtualization, container based virtualization is lighter. Borko Furht et al. did a detailed investigation on ARM-based mobile virtualization from KVM, to VMWare, and from BlackBerry to Android in book [25]. And a practical and lightweight domain isolation solution named TrustDroid are introduced which can achieve good isolation without duplicating any portion of operating system stack. In Ref.[26], a container based solution for Android by building a virtual binder IPC layer is described. It improves memory performance with service sharing and saves storage with a read-only filesystem sharing.

Felter et al. tested VM-based virtualization (KVM) and container based virtualization (Docker) on an IBM R System x3650 M4 server and Docker won all the performance competition [9]. The performance of KVM, Xen (VM-based) and Docker on ARM has been compared with the results obtained in Ref.[27]. The result shows that Docker still has the advantage except in file-system operations. Though Docker is much lighter than VM-based virtualization, it still too heavy to most of embedded systems especially to tiny system like AVR-based nodes. E.g. the size of core Docker source code is more than 30 Mbytes. To our best of knowledge, all of current virtualization are designed for the general system or for the high performance embedded platform. These solutions are too heavy for most of CPS nodes.

III. Time Bound Based Container Model

Event-driven system becomes popular because this kind of programming model fits the characteristic of CPS better, it describes the stochastic events in real world naturally [2], [16]. We use event-driven mealy FSM with recovery operation to model the application of CPS. We use Actor to donate the application/service which is defined as fallow:

$$Actor = (\Sigma, S, s_0, \theta, \Psi(IN, OUT), T, \Upsilon) \quad (1)$$

Where Σ is the input alphabets set and S is the state set, s_0 is the initial state, θ is the transition function, Ψ is the operation function, IN and OUT are the input and output of Ψ, Υ is the recovery operation, T is the set of time bound of $Max(\theta, \Psi)$. And the transition can be defined as fallow:

$$\theta_i : \zeta \times s_i - [\psi_i(in, out)] \rightarrow s_j \mid (t_i, \gamma_i);\ \zeta \in \Sigma \quad (2)$$

The actor switches from one state s_i to another state s_j when receives an event ζ, and executes operation ψ_i at the same time. The t_i is the WCET of this transition, it is the maximal time of transition θ_i and operation ψ_i. γ_i is the recovery function, it executes only if the transition violate the WCET constraint.

Actor is the abstraction of the atomic components/services of a node. Generally, one application contains several cooperative actors. These actors are belong to one same group and communicate with each other through message case. The communication in the same group is simplified as fallow:

$$Actor_i \xrightarrow{Message} Actor_j \quad (3)$$

Considering the limited resources of sensors and actuators, we prefer to apply the container-based virtualization rather than virtual machine based (VM-based) virtualization on our AVR-based embedded system. According the existing container solutions, container is the basic resource unit with running environment, so that the container can be executed on any node in the cloud system. And the unit of scheduling can be denoted with a tuple $<C_k,<Actor_k(\theta_i),(t_i,\gamma_i)_k>>$. Where C_k is the container that contains a group of Actors. The system has to create a container instance for each application $\{Actor\}$ and schedules at both the container and application level. It increases the memory footprint and logic complexity.

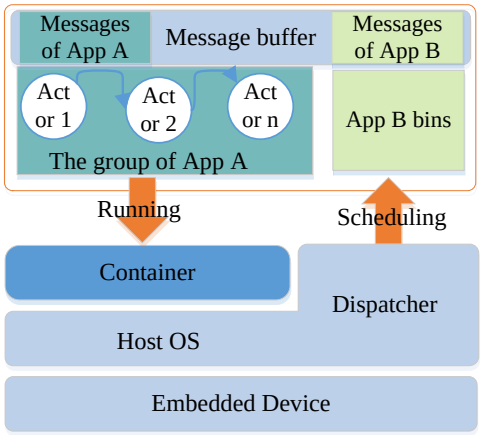

Figure 2. The framework of light weight container-based virtualization

To CPS, most application is location-aware, which means the meaning of data and the behavior of an application closely depends on the surrounded environment where the processing takes. For example, imaging that the plants in place A is thirsty, but the actuator transfers the watering process to another actuator located in B and water the plant in place B. It sounds ridiculous and the transferring is nonsensical. This characteristic inspires us that container migration is unnecessary in CPS and we can build the container as a packaging function $C(Actor_k(\theta_i),(t_i,\gamma_i)_k)$. $\{Actor\}$ are the parameters of container. In our model, container is still the basic resource unit but the unit of scheduling is Actor. And in order to distingue the Actors in different groups and isolate the relevant resources, we add a field of *group_id* for the each Actors, and the communication is modified as $Actor_i \xrightarrow{Message_{group\_id}} Actor_j$, every message sent should carry a *group_id*, both $Actor_i$ and $Actor_j$ have the same *group_id* with the message.

The framework of such container illustrated as Figure 2. The node instants only one container. Every messages should carry a *group_id*. The Actor cannot read the message if it doesn't share the same *group_id*. The scheduler dispatches the applications as normal, just like the system without container. But the applications should be executed within the container.

IV. Container Design and Implementation

The applications on sensors and actuators share the hardware resources together with almost none protection. For example, Contiki OS does not take addition actions on failure detection and diagnose, the applications on it should manage all the failures by itself, even though some failures are in common, which distracts developers from the normal function design, and also complicates the system.

In order to run the container on tiny embedded system, we should simplify the container step further. Generally, embedded control system is a single-user system, so it's unnecessarily to apply user policy and user group policy. And also the programs on sensors and actuators are statically linked, which greatly simplifies the container on the Dynamic Link Libraries (DLL) environment management. According those premises, a light weight recoverable container is designed.

A. *The architecture of container*

To achieve flexibility, the application is decoupled and made up of several actors. The container-based virtualization architecture is showed in Figure 2, the container deploys on Host OS. To host OS, the actor is a scheduling unit, it can dispatch the applications as before, so it can minimize the change of application development. Container generates the run-time environments for actors especially the run-time stack. To minimize the memory footprint, all the messages are stored in one same message buffer and marked with a *group_id*. Actors should read the message with a system function. And the function will prevent to read and write the message carrying a different *group_id*.

With this approach, the OS just needs to maintain only one container instance and one message buffer, which enables running container with a tiny memory overhead. And by handing the *group_id* over to buffer management function, it simplifies the maintenance of the running time environment in container.

B. *A multilevel failure detection and recovery*

Furthermore, we design a multilevel failure detection and recovery strategy. Firstly, a WCET-based failure detector is built in the container. At soon as putting an $Actor_k$ into the container, the detector starts monitoring and the timer starts counting. If the actor cannot finish its action θ_i within time t_i, a timer interrupt will be triggered. Then the failure counter of $Actor_k$ will increase and the state of actor will be set as failed. To minimize the influence to other actors, the recovery operation γ_i is only taken when system is free. If the recovery failed, the current instance of $Actor_k$ will be given up, and new instance will be appended to the ready queue. New turn of scheduling will start. The failure number of recovery will

be counted, if the failure counter increasing too quickly, the failure actor will be regarded unreliable and disabled.

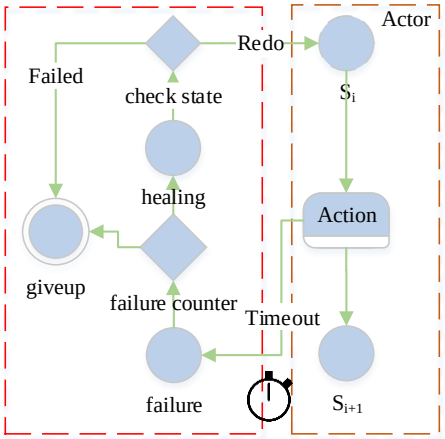

Figure 3. The process of failure detection and recovery

As the recovery function is user-defined, it may also fail. We use a watchdog to detect the time failure of recovery and build a multilevel failure detection solution shown in Figure 4.

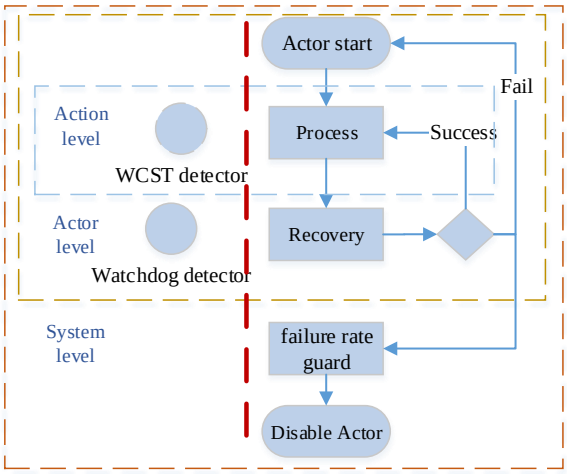

Figure 4. Multilevel failure detection and recovery

On system level, we use a global counter to record failure of actors. If the counter increases too fast which means the whole system is ill, a restart operation will be trigged. For example, Actor A may modify other actors' memory if the Program Counter (PC) register is error, so other actors also fail quickly and the whole system becomes untrusted. Restarting is the only low cost method.

C. *The detailed implementation*

The container is implemented as a function, and it is defined by the Figure 5. As soon as the container is executed, it firstly stores current running environment (the current field, the registers) with *setjmp (env)*. Secondly, a timer is started and the recovery operation are initialized. After that, the action of $Actor_k$ is processed. If the action finished successfully, timer will be stop and all resources for recovery will be released. For this implementation, the time threshold is the sum of normal WCET time, the time to store the current field, the execute time of functions *Daemon_Recover_Init*() and *Daemon_Stop*(). The detail source can be found in [28].

```
void Container (pfun_process process,pfun_action
recovery, uint16_t deadline_ms, void* args)
{
    int rev = setjmp (env);
    if(rev>0)
    { //do something if necessary; e.g. disable actors or
release the resource allocated by process()
        return;
    }else{
        Daemon_Recover_Init(deadline_ms,recovery,
&env, rev);
        process(args);
        Daemon_Stop();
  }
}
```

Figure 5. Implementation of container

SP ->			unused
10 data in registers	R31	R31	
	R30	R30	
	……	……	
	R1	R1	
Return address	ISR high	Rec_High byte	SP+11
	ISR low	Rec_Low byte	SP+12
	……	……	
Bottom	0X8E	0X8E	0X2200
	Original Stack	Recovery stack	

Figure 6. ISR return address redirecting (ATmega 1281)

And we use timer/counter4 in AVR as the timer for detection. In order to keep the interrupter function as simple as possible and don't block other interrupters, we take a little Foxy strategy on the return address of ISR by redirecting the address to the function recovery(), this function just sets the state of actor to failed and recovers the stack with the stored current field, the detail recovery operation of actor is done when system is free. The running stack and the return address show as Figure 6.

To a system with no supervisor, it's important to provide a mechanism to prevent the system from getting stuck. This container provides an isolated stack for every

actor, reactivates the system by recovering the stack when the time bound is missed, and protects the context of other actors with low cost.

V. PERFORMANCE AND RELIABILITY EVALUATION

A. *Overhead and performance analysis*

The memory footprint of recoverable container is 939 bytes with 904 bytes of Text and 35 bytes of BSS. The performance loss is no more than 294 instructions which costs for function calling of *Container*, *setjmp*, *Daemon_Recover_Init*, *Daemon_Stop*, and *longjmp*. And the size of protecting field also a MCU related constant which is 23 bytes for iLive board[29] with ATmega 128RFA1. The time complexity of these functions are $O(1)$. As the time complexity is constant, the container can be applied in the real-time system. The memory complexity is $O(1)$ for field protecting (the field data to protect in *setjmp*), and as all messages and actors have *group_id*, it takes $O(n)$ complexity of extra memory.

B. *Reliability analysis and testing*

To evaluate the reliability recoverable container, here, we assume that the recovery function preforms as exactly as designed. As AVR is a modified Harvard architecture, only SRAM can be rewritten in normal running mode. So the recovery mechanism will not fail except in these situations below.

1) The data of field is destroyed

E.g. application has random dangling pointer.

2) PC register is error, recovery is missed or failed

E.g. stack overflow and the return address is error.

The hardware of PC register is error. The probability that field is destroyed is

$$P_1 = size(field) / size(sram) \tag{4}$$

Assume no other resetting operation in system, and the error value of PC register can be regard as random value. We can get the probability of restarting before recovery as fallow

$$P_2 \le WCET * IPS / size(\text{Flash}) + rate(T(C) + T(OS)) \tag{5}$$

Where $WCET$ is the worst case execute time, the PC gets an error as soon as actor starts. IPS is the instructions per second. Because the reset interrupt is at the beginning of binary code (for AVR, the addresses are 0x00 to 0x70). $T(C)$ and $T(OS)$ are the time amount that container and OS take. $rate$ is a percentage function. $WCET * IPS / size(\text{Flash})$ is the possibility that actor fails and system restarts before $WCET$ violated $rate(T(C) + T(OS))$ is the rate that PC pointer gets error and the code just locates right in area of container and host OS.

And sum of failure rate of recovery is $P_{sum} \le P_1 + P_2$. So the reliability of recovery is $R_{rev} \ge 1 - P_{sum}$;

We test the recoverable container code on iLive series board and Arduino Mega2560 board. The MCU includes AVR ATmega 1280/1281/2560/128RFA1, we design 3 kinds of failures to test the reliability of container. These failures are shown as follow.

1) WCET Violation

Simulate with endless loop.

2) Random SRAM Modify

Randomly modify the SRAM data with a uint8_t pointer to simulate the field destroyed failures.

3) Random PC Error

Simulate with a function pointer with random address.

We test recoverable container on iLive node with more than 2 months. The WCET of all testing is 4 seconds. In the testing case 3, we ignore the part of $rate(T(C) + T(OS))$, as we just inject PC register error at application level. System restarts when the recovery mechanism based on timer is failed. And if the system failed to restart, it means the watchdog detector fails. The result is shown in Table 1.

TABLE 1 RESULT OF TESTING

Fault injected	Testing duration	System restart times	
		Theoretical value	Record value
WCET Violation	5281965 secs	0	0
Random SRAM Modify	About 7 days[a]	3628.8	3526
Random PC Error	550368 secs	< 34398	59113

[a.] it failed to restart automatically once

The result in Table 1 shows the recoverable container is highly reliable on WCET violation recovery, and even the application has defects or PC register gets transient fault (in test case 2 and 3), container can isolate the error and recover without restarting with high probability. The multilevel recovery mechanism is very effective on enhancing the availability of system.

The testing record of random PC Error is higher than theoretical value. The reason includes that 1) we use watchdog as the second detector which increases the trigging rate of resetting, 2) system restart in less than 4 seconds (WCET) and more testing is done than expected. 3) PC Error may also destroy the field data. Though the container is design for software failure detection, the testing also shows a method using software to protect application from transient hardware fault with probabilistic recovery. By the way, PC error rate generally is extreme low. According the Atmel Reliability Monitor Report [30], the Failure Rate of the whole MCU is just about 7 FITS in High Temperature Operating Life testing, which means the Mean Time to Failure (MTTF) is about 142857142 hours.

VI. CONCLUSIONS AND FUTURE WORK

CPSs suffer the volatile environments which lead to frequency failures. Isolation is a key method to minimize

interference between applications and to achieve reliability. Virtualization shows a practical direction to universal isolation solution. The multilevel self-healing solution presented in this paper is a light-weight recovery container which can be applied on AVR-based CPS nodes. It can isolate actors with container, protect applications with *group_id*. The analysis shows this solution has low overhead and can be applied in tiny system easily. The testing with fault injection proves that the multilevel recovery mechanism has high performance on defending WCET violation failure. As the time complexity of all function of container is constant, it can also be possible to apply the recoverable container in a real-time system. What's more, if we implement the failure information collection in failure state, it can be used to test the errors with low frequency or the errors that difficult to be repeated in laboratory environment, which is a problem that troubles the developers of WSN a lot.

Current work focuses on dynamic executing environment protection, the failure detection strategy only supports WCET based detection. As WCET method still container some drawbacks [31], we will continue the work on exploring common behavior properties which requires few details of application logic. And also we will do research on the I/O virtualization to provide an isolated operation on network and interaction with physical world.

ACKNOWLEDGMENTS.

This work was supported by a grant from National High Technology Research and Development Program of China (863 Program) (No. 2013AA01A205), and National Natural Science Foundation of China (No. 61370085).

REFERENCES

[1] Petroulakis N. E., Spanoudakis G, et al. A Pattern-Based Approach for Designing Reliable Cyber-Physical Systems. 2015 IEEE Global Communications Conference (GLOBECOM), IEEE. 2015

[2] Lee, E. A. The Past, Present and Future of Cyber-Physical Systems: A Focus on Models. Sensors 15(3): 4837-4869. 2015

[3] Kim J, Puraniket P, et al. Realizing a fault-tolerant embedded controller on distributed real-time systems. ACM Sigbed Review 10(4): 33-36. 2013

[4] Nageswara S.V Rao. On Undecidability Aspects of Resilient Computations and Implications to Exascale. Euro-Par 2014: Parallel Processing Workshops, Springer.2014

[5] Fouquet F, Olivier B, et al. A dynamic component model for cyber physical systems. Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering, ACM. 2012

[6] IBM White Paper. Autonomic Computing Concepts, Available at http://www-03.ibm.com/autonomic/pdfs/AC Concepts.pdf

[7] Sha, L. Using simplicity to control complexity. IEEE Software. vol.18(4): 20-28. 2001

[8] D. Jackson. A direct path to dependable software. Communications of the ACM, vol. 52, No. 4, 2009

[9] Felter W., et al. An updated performance comparison of virtual machines and linux containers. Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On, IEEE.2015

[10] Sanislav, T., et al. A new approach towards increasing cyber-physical systems dependability. Carpathian Control Conference (ICCC), 2015 16th International, IEEE. 443 – 447, 2015

[11] Leon Wu and Gail Kaiser, "FARE: A Framework for Benchmarking Reliability of Cyber-Physical Systems", Columbia University Computer Science Technical Reports, Columbia University, 2013

[12] Gunes, V., et al. A Survey on Concepts, Applications, and Challenges in Cyber-Physical Systems. KSII Transactions on Internet and Information Systems (TIIS) 8(12): 4242-4268. 2014

[13] Lee, E. A. Computing needs time. Communications of the ACM 52(5): 70-79. 2009

[14] Hu Y-L,Su W-B, et al. Dependable Architecture of RFID Middleware on Networked RFID Systems. Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, IEEE. 2013

[15] Valverde J, et al. A dynamically adaptable bus architecture for trading-off among performance, consumption and dependability in Cyber-Physical Systems. Field Programmable Logic and Applications (FPL), 2014 24th International Conference on, IEEE. 2014

[16] Sanislav T, and L. Miclea. An agent-oriented approach for cyber-physical system with dependability features. Automation Quality and Testing Robotics (AQTR), 2012 IEEE International Conference on, IEEE. 2012

[17] Wu L, and G Kaiser. An autonomic reliability improvement system for cyber-physical systems. High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on, IEEE. 2012

[18] Lin, M., et al. Scheduling co-design for reliability and energy in cyber-physical systems. Emerging Topics in Computing, IEEE Transactions on 1(2): 353-365.2013

[19] Sanislav T., et al. A new approach towards increasing cyber-physical systems dependability. Carpathian Control Conference (ICCC), 2015 16th International, IEEE. 2015

[20] S. Krishnamurthy, S. Sarkar, and A. Tewari, Scalable anomaly detection and isolation in cyber-physical systems using bayesian networks, in ASME 2014 Dynamic Systems and Control Conference, 2014

[21] J. Korbicz, J. M. Koscielny, Z. Kowalczuk, and W. Cholewa, Fault diagnosis: models, artificial intelligence, applications: Springer Science & Business Media, 2012. pp. 59-113

[22] Wander, A. and R. Förstner. Innovative fault detection, isolation and recovery strategies on-board spacecraft: State of the art and research challenges, Deutsche Gesellschaft für Luft-und Raumfahrt-Lilienthal-Oberth eV. 2013

[23] Wilhelm, R, Jakob E, et al. The worst-case execution-time problem—overview of methods and survey of tools. ACM Transactions on Embedded Computing Systems (TECS) 7(3): 36.2008

[24] Heiser, G. The role of virtualization in embedded systems. Proceedings of the 1st workshop on Isolation and integration in embedded systems, ACM.2008

[25] Jaramillo D., et al. Mobile virtualization technologies. Virtualization Techniques for Mobile Systems, Springer: 5-20. 2014

[26] Xu, L, et al. Condroid: A Container-Based Virtualization Solution Adapted for Android Devices. Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on, IEEE. 2015

[27] Raho M., et al. KVM, Xen and Docker: A performance analysis for ARM based NFV and cloud computing. Information, Electronic and Electrical Engineering (AIEEE), 2015 IEEE 3rd Workshop on Advances in, IEEE. 2015

[28] The source code of container and testing is available at: https://bitbucket.org/Teampp/rc/

[29] Hong-Ling Shi. Development of an energy efficient, robust and modular multicore wirelesssensor network. Université Blaise Pascal - Clermont-Ferrand II, 109-110, 2014.

[30] Atmel, Atmel Reliability Monitor Report. Available at http://www.atmel.com/Images/Relmtrq4-15.pdf

[31] Abella. J, Hernandez. C, Quiñones, et al. WCET analysis methods: Pitfalls and challenges on their trustworthiness. 10th IEEE International Symposium on Industrial Embedded Systems (SIES), 2015, pp 1-10