Embedded Virtualization for the Design of Secure IoT Applications

Carlos Moratelli
carlos.moratelli@pucrs.br

Sergio Johann Filho
sergio.filho@pucrs.br

Marcelo Veiga Neves
marcelo.neves@pucrs.br

Fabiano Hessel
fabiano.hessel@pucrs.br
PUCRS, Porto Alegre, Brazil.

ABSTRACT

Embedded virtualization has emerged as a valuable way to reduce costs, improve software quality, and decrease design time. Additionally, virtualization can enforce the overall system's security from several perspectives. One is security due to separation, where the hypervisor ensures that one domain does not compromise the execution of other domains. At the same time, the advances in the development of IoT applications opened discussions about the security flaws that were introduced by IoT devices. In a few years, billions of these devices will be connected to the cloud exchanging information. This is an opportunity for hackers to exploit their vulnerabilities, endangering applications connected to such devices. At this point, it is inevitable to consider virtualization as a possible approach for IoT security. In this paper we discuss how embedded virtualization could take place on IoT devices as a sound solution for security.

CCS Concepts

•**Security and privacy** → **Authentication;** *Software security engineering;* •**Computer systems organization** → **Embedded software;** •**Software and its engineering** → *Embedded software;*

Keywords

Embedded Virtualization, IoT, Security.

1. INTRODUCTION

The modern world is moving rapidly toward a connected model where billions of electronic devices will be interconnected through computer networks, especially the Internet. This has defined a concept called the Internet of Things (IoT). In this context, real objects are connected to the Internet, uniquely identified and accessible through the network. This results in a combination between the physical and the digital worlds expanding the Internet with new services and intelligence [8]. There are many possibilities for IoT applications, which include different domains like energy distribution, agriculture, healthcare, and transportation [9]. In light of the inevitable adoption of IoT in the future, some of its inherent problems will require attention. One of the main issues to be addressed is the security holes that may be introduced in private networks by the IoT devices. The Internet of the future, with billions of connected devices, is an attractive place for hackers to exploit security flaws and endanger life and business. Thus, building more secure IoT devices is a topic that requires further investigation.

A possible approach to improve security in the IoT field is the use of virtualization. Virtualization technology became a reality in modern embedded systems, motivated by the increasing challenges to comply with the requirements in this area. However, the intrinsic requirements of embedded systems, such as small memories, application footprint and tight real-time constraints motivated the appearance of hypervisors especially designed for embedded virtualization [13], [16] and [15]. This new generation of hypervisors was followed by the adoption of hardware-assisted virtualization, a set of processor features aiming to speed-up performance and simplify the design of hypervisors. The main embedded processor manufacturers have already designed virtualization extensions for their processors, such as the PowerPC [5], ARM [1], and MIPS [6] families of processors.

Among the goals specified during the design of embedded hypervisors, two can be highlighted: to keep low memory requirements and some level of support for real-time applications. These goals resulted in hypervisors with a low memory footprint and runtime overhead. Some of these hypervisors are small enough to fit IoT devices' memory and footprint requirements. Additionally, their capability to support real-time operating systems (RTOSs) and bare-metal applications make them a possible approach to build more secure IoT devices. Moreover, all advantages of general purpose virtualization will be inherited by IoT devices.

At first glance, virtualization on such resource constrained devices may appear to be an overkill solution. However, as we will show in this paper, the processing power and the amount of memory on some modern microcontrollers is enough to define virtualization as a feasible solution. We will also show that the strong spatial and temporal separation provided by hypervisors can considerably improve the security on IoT devices. The main contribution of this paper is the description of a flexible architecture which aims to improve security on IoT using virtualization.

The paper is organized as follows: Section 2 describes the main security flaws introduced by IoT devices and opportunities for improvements. Section 3 shows how hypervisors can improve the overall security on IoT devices. Section 4 describes a real-world application where a hypervisor is used

RSP'16, October 06-07, 2016, Pittsburgh, PA, USA

DOI: http://dx.doi.org/10.1145/2990299.2990301

to provide security by separation. Section 5 analyzes related works for IoT security solutions and some available hypervisors that are affordable for IoT virtualization. Finally, Section 6 presents our conclusions and future work.

2. IOT SECURITY FLAWS

Security is one of the major concerns regarding the development and adoption of the Internet-of-Things [9, 20, 7]. Due to the nature of some IoT applications (e.g, gathering of valuable personal/business information) and the potentially large number of connected devices, IoT may arguably become a major target for hackers' attacks. Apart from the security flaws present in today's Internet, additional ones are expected to be introduced with the massive use of IoT devices. This section will discuss the main security issues related to IoT as well as some opportunities for improvement.

The security of a IoT device can be compromised either remotely or locally. Remote flaws may include the exploitation of vulnerabilities via network. For geographically distributed IoT applications, the devices are expected to be accessible via Internet. Since today's Internet still uses the TCP/IP protocol suite, such applications will require additional mechanisms to ensure confidently, authenticity and availability. Local flaws may include modified/malicious software (e.g., malwares, virus) or even malfunctioning applications.

Depending on the application, a compromised IoT device can represent a serious danger for life and business. For example, it may involve stealing private/sensitive information (privacy/confidently); modifying information (authenticity); or causing the malfunctioning of the device itself or other devices controlled by it (availability and resilience to attacks). Moreover, it can even introduce security holes to get access to private networks.

Some IoT devices are expected to be deployed in unattended outdoor scenarios, which may allow one to have physical access to it. In this case, it is necessary to ensure that the device cannot be tampered in order to extract sensitive information or install any modified/malicious software.

2.1 Opportunities for Security Improvements

In this context, there are many opportunities for security improvements. In this paper, we argue that a large portion of the security issues in IoT can be solved by employing techniques of separation (spatial and temporal) combined to trust mechanisms.

The first step in order to improve the security of a embedded system for IoT is to ensure that all running code is cryptographically signed [14]. IoT devices must be designed to only boot up if the bootloader software is signed by an authorization entity (e.g., the device vendor). This may involve specialized hardware to verify that signature at the startup and put a trusted bootloader to run. This enables a Root of Trust (RoT) that can be used to ensure the integrity of running code.

At a higher level, a trusted hypervisor plays a key role in the IoT security. As mentioned before, a hypervisor for embedded systems consists on a small footprint piece of privileged code that manages the system-on-chip resources and allows for creating multiple isolated execution domains and defining access policies for them. In other words, it is possible to run multiple guest OSes and/or bare-metal applications that have limited privileges. The hypervisor protects guest OSes from each other and also protects itself.

One of the main benefits of using separation is to avoid that exploitable defects in one virtualized domain (e.g., guest OS) propagate to adjacent virtual domain, or to the physical platform, which makes the system more resilient to attacks. Spatial separation can be used to improve privacy by allowing IoT devices to securely collect and process sensitive data, even if one of the virtual domains is compromised (a example of use case will be presented in Section 4.1). Temporal separation can be used to ensure fair resource sharing among the virtualized domains, which may also improve availability and resilience. Take for example the case where one of the virtualized domains suffers a resource-intensive attack, such as a Distributed Denial-of-Service (DDoS) attacks – which is very common in today's Internet. In this case, the other domains will continue to work regardless the misbehavior of the compromised one. The Section 3 describes how security by separation can be implemented in IoT devices.

3. SECURITY BY SEPARATION FOR IOT DEVICES

The trend of multi-core processors being increasingly used in embedded designs reinforced the necessity of separation. The simplest way to guarantee separation is to use more than one processor and physically separate address spaces. The communication between processors would require an interconnection network and a message passing protocol, because they are separated by hardware. Yet, the adoption of modern hardware, including multi-core processors, means that everything is connected and sharing a common memory. However, developers still want to keep the separation between their applications. Virtualization is the best technology to guarantee spatial and temporal separation between applications. This separation is required even in single-core environments. The hypervisor can create this separation by making the software execute in the virtual machines (VMs) completely unaware of other software running on other VMs even if they are being performed on the same physical hardware. Hereafter, we explore how hypervisors can provide spatial separation (Section 3.1) and temporal separation (Section 3.2).

3.1 Spatial Separation

The most common way to provide memory isolation between processing elements in a system is through a memory management unit (MMU), a hardware block which provides virtual memory abstractions to the system. A typical MMU allows for the OS to implement memory isolation between processes. The OS translates virtual addresses (VAs) to physical addresses (PAs) using its page table to configure the processor's translation lookaside buffer (TLB). In a virtualized environment, the hypervisor must keep memory isolation between VMs while the OS is still able to keep isolation between processes. Thus, VAs are translated to intermediate physical addresses (IPAs) that are managed by the hypervisor. Without proper hardware-assisted virtualization, the management of the IPA needs to be emulated through a technique that makes use of a shadow page table. This table keeps the correct translations from IPA to PA in an intermediate page table, and its management increases the hypervisor overhead and complexity.

Hardware-assisted virtualization implements a second stage MMU translation in hardware. Essentially, the hardware performs the translation from IPA to PA without software intervention. The hypervisor still manages its page table by mapping an IPA to a PA in a second stage MMU. The guest OS manages the first-stage of address translation in exactly the same way as on a non-virtualized system. The result-

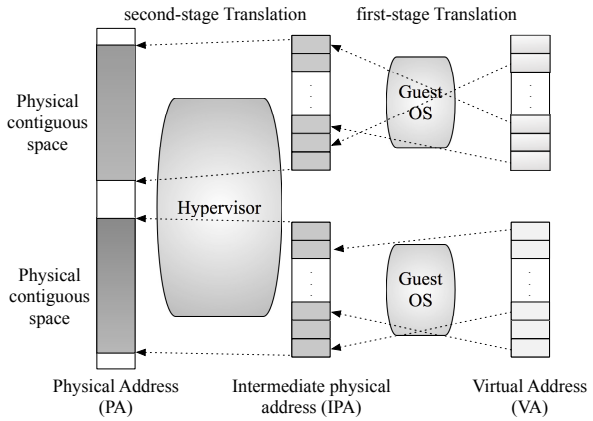

Figure 1: Virtual memory organization view on a virtualized system.

ing PA is generated by the hardware combining both TLBs. This mechanism drastically decreases the number of hypervisor exceptions and also the complexity of such hypervisor. Figure 1 depicts this scheme in case the hypervisor keeps the VM's in contiguous physical address space.

3.2 Temporal Separation

The temporal separation guarantees the correct distribution of processor time between the virtual CPUs (VCPUs) according their execution priorities. The hypervisor's scheduler implements algorithms that are used to schedule VCPUs. Many different authors have addressed the hypervisor's scheduler as a way to improve temporal separation and to honor real-time constraints, as seen in [3] and [19]. Some hypervisors distinguish between best-effort VCPUs (BE-VCPUs) and real-time VCPUs (RT-VCPUs). RT-VCPUs have priority over BE-VCPUs and follow the same policy of some known real-time scheduling algorithms, such as EDF. BE-VCPUs are scheduled by a best-effort scheduling algorithm that is invoked when there are no RT-VCPUs ready to execute. For example, RTOSs can be mapped to RT-VCPUs while Linux instances can be mapped to BE-VCPUs, because a general purpose OS aims to improve fairness over other metrics.

In a system with such hibrid characteristics, virtualization implies in the hierarchical scheduling problem, also known as two layer scheduling problem. A hypervisor schedules VCPUs, and a guest OS executing in a given VCPU schedules its processes or tasks. Even when the correct timing is ensured by the scheduling at the VCPU level, it is a difficult task to ensure timing constraints to the tasks which execute in a RTOS. However, if both VMs and the hypervisor make use of proper real-time algorithms, the system can be modeled as a hierarchy of schedulers, and real-time performance of each individual virtual system can be evaluated by using hierarchical scheduling analysis techniques [4].

System interrupts require attention since they interfere directly on the VCPU's execution. The interrupt subsystem must be designed as part of the hypervisor's scheduler. Thus, the hypervisor can implement different interrupt delivery policies according to the application needs. For example, when an interrupt occurs during the execution of another guest OS, the hypervisor must decide between: i) keep the interrupt masked and delay its delivery or; ii) intercept the interrupt and reschedule the guest OSs. This allows the implementation of different priority policies for interrupt control and making the temporal separation between guests possible.

4. PRPL-HYPERVISOR - A HYPERVISOR FOR IOT DEVICES

The Hellfire Hypervisor [18] is a small type-1 hypervisor specially designed for embedded devices. It was ported to an IoT platform (see Section 4.1) and renamed prpl-Hypervisor^{TM}. The prpl-Hypervisor^{TM} is part of a security framework maintained by the prpl Foundation, which is an open-source, community-driven, non-profit foundation targeting and supporting MIPS architecture. Thus, the prpl-Hypervisor^{TM} is open-source[1] and has a very permissive software license that allows for its deployment in both academic or commercial applications. The original Hellfire Hypervisor was developed targeting the MIPS M5150 processor with the MIPS-VZ architecture, which is the virtualization extension architecture for MIPS. The M5150 supports the second-stage MMU translation (as explained in Section 3.1) allowing the hypervisor to easily manage the memory separation between guests. Additionally, the hypervisor keeps spatial separation between the guest kernel and guest applications while protecting itself. This protection is possible because processors implementing the MIPS-VZ module support a third level of execution (supervisor mode) with higher privilege designed for the hypervisor. Therefore, the hypervisor is protected from the actions of malicious or misbehaving guest OSs, and the guest OSs are protected from their applications. It is important to highlight that embedded hypervisors designed for processors without proper virtualization assistance cannot distinguish the memory space between the guest kernel and applications, since the entire guest executes in the processorâĂŹs unprivileged mode [21]. The temporal separation is guaranteed through a round-robin scheduler algorithm and interrupt delivery policies as established in [11]. Finally, the hypervisor enables the implementation of the para-virtualization concept to provide extended services to the guest OSs. These services are useful to expand the virtualization functionalities, i.e., to implement functions that do not exist in a purely virtualized system, such as inter-VM communication.

This separation enabled the adoption of MIPS-VZ and resulted in security improvements. Yet, that is not the only advantage introduced by hardware-assisted virtualization. Hardware features, such as second stage MMU translation and the third level of execution simplifies the overall hypervisor design. As a result, the prpl-Hypervisor has a small footprint. The storage requirement is around 40 kilobytes while the RAM requirement is 20 kilobytes. Thus, the overall footprint is only 60 kilobytes, making the prpl-Hypervisor^{TM} a attractive choice for IoT virtualization. Section 4.1 presents proof of concept of virtualization for an IoT platform.

4.1 Proof Of Concept Of Virtualization for IoT Platorms

The PIC32MZ EF series is a microcontroller family targeting IoT and embedded markets. It is powered by the MIPS M5150 processor core. The PIC32MZ EF can run up to 200MHz resulting in 330 DMIPS and 3.28 CoreMarks/MHz. It supports up to 2 MB of flash memory (for program and data storage) and 512 KB of RAM. It has a wide diversity of connectivity peripherals including a 10/100 Ethernet MAC, Hi-Speed USB MAC/PHY, and dual CAN ports. An op-

[1]https://github.com/prplfoundation/prpl-hypervisor

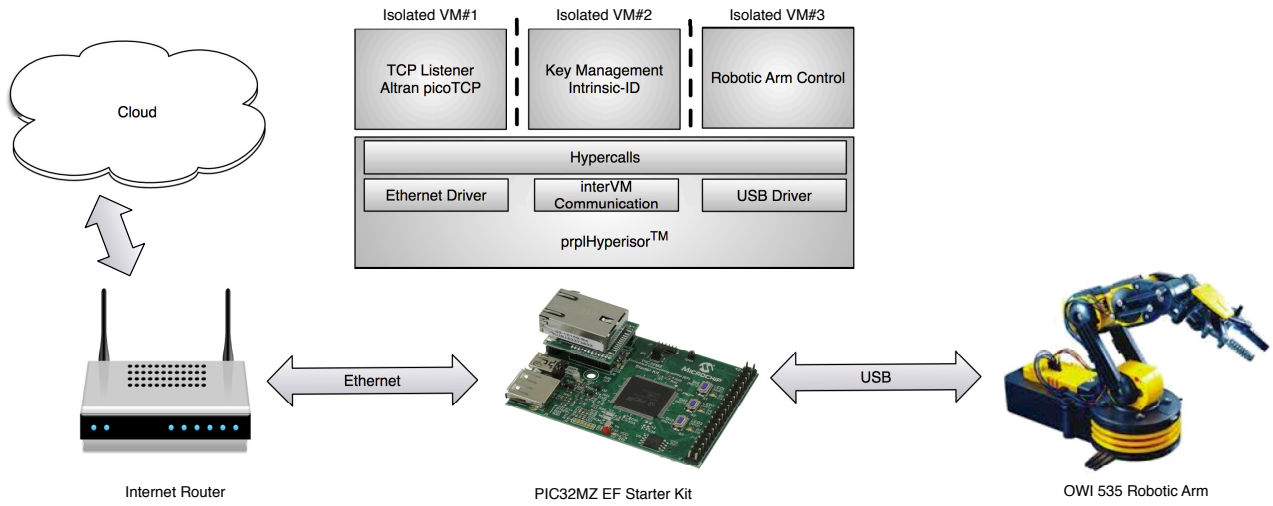

Figure 2: prplSecurity™ framework as a proof of concept of IoT virtualization.

tional crypto engine is also available with a random number generator for high-throughput data encryption/decryption. These features, associated with the hardware-assisted virtualization provided by the M5150 processor core, make the PIC32MZ EF microcontroller family a optimal choice for IoT virtualization. We ported the Hellfire Hypervisor to the PIC32MZ Embedded Connectivity with FPU (EF) Starter Kit development board (PIC32MZ2048EFM144) [10] renaming it prpl-Hypervisor™. This board has 2 MB of flash and implements the Ethernet and USB peripherals. Additionally, it exposes SPI, UART, and several I/O pins.

As proof of concept for IoT virtualization, an embedded application was implemented to control the movement of a robotic arm connected through the Internet. Beyond the cited PIC32MZ EF board, the application uses an OWI-535 robotic arm, which is controlled through the USB interface. The board must be connected through the Ethernet port to a router or directly to a laptop (for test purposes). Figure 2 shows this scenario.

The software architecture consists of three separate VMs, see Figure 2. Each VM runs a bare-metal application. To enable communication, a small open-source TCP/IP stack called picoTCP was developed and is maintained by Altran and supported on VM1. PicoTCP uses a hypercall interface to send/receive network datagrams through the hypervisor. The VM2 implements the prplPUF™ API developed by a company called Intrinsic-ID that authenticates incoming requests. The VM3 has the arm control software that sends/receives data through the PIC32MZ EF's USB controller to the OWI-535 robotic arm.

The system has various parallel functions. During boot time, the VM2 generates a unique key from the hardware fingerprint that can be recreated at every reboot and doesn't need to be stored in the system. Once connected to a network, a TCP server on VM1 keeps listening for incoming connections. Every client connection is verified using the VM2's authentication services. The VMs communicate with each other using the hypervisor's inter-VM communication tools. If a client is successfully authenticated, it can send start/stop commands to VM1 that are relayed to VM3 to control the arm's movement.

As the three VMs are completely separated by the hypervisor, even if VM1 (the only VM exposed to the network) security is compromised the attacker cannot take control of the robotic arm, since it needs the key on VM2.

5. RELATED WORKS

Puliafito et al. [2] propose using container virtualization. Their approach can be used as a lightweight alternative to hypervisors, while still addressing most embedded virtualization application needs. Although an effective approach, container virtualization is based on system calls. More specifically, the proposed implementation relies on Linux Container Virtualization (LCV), requiring a relatively powerful hardware platform in terms of processing power and memory.

The work presented in [12] defines the idea of temporal separation using virtualization for criticality mixed real-time systems. The authors explore the coexistence of different critical functionalities on a multicore platform for avionics, by using virtualization to tackle security and safety failures in the system. Their model is based on temporal isolation using PCIe I/O virtualization.

Several approaches regarding the potential of hardware-based IoT security are discussed in [17]. The authors perform a series of case studies that advocate the use of stable PUFs, hardware obfuscation, and digital PPUFs for a range of security protocols.

In [20] different emerging security threats and countermeasures for IoT are investigated. The authors evaluate threats in a large, unreliable, and pervasive computing environment, where sensitive and private information is exchanged between objects. They apply several conventional security approaches such as authentication schemes to ensure the confidentiality and integrity of data.

A virtualization approach for applications is presented in [16]. The work is based on the Xtratum hypervisor and aims to support the virtualization of applications with different levels of criticality on heterogeneous multi-processor embedded systems. The goal is to have these applications interacting with each other and co-existing in the same platform. Yet, guaranteeing that certification of such applications is still possible because of the isolation. The proposed hypervisor is based on a para-virtualization technique to accomplish spatial and temporal isolation among VMs, security, confidentiality, predictability, resource allocation, and fault isolation. The architecture is based on a shared memory heterogeneous multi-core composed of x86 and LEON3

cores. The use of para-virtualization and complex cores such as proposed in the work impose several limitations to the applicability of the model in the IoT field, where energy consumption, memory size, and processing power are limiting factors.

6. CONCLUSION AND FUTURE WORK

In this paper we showed that embedded virtualization is a valid approach to improve IoT security through separation. Additionally, it was demonstrated that specially designed embedded hypervisors can be small enough to fit IoT devices, enabling all of their virtualization advantages. A real-world application involving network communication and hardware control was developed as a demonstration of virtualization on IoT platforms. As future work, we will explore root-of-trust and secure-boot mechanisms for virtualized environments on IoT devices.

7. ACKNOWLEDGE

This work was partially supported by the prpl Fundation.

8. REFERENCES

[1] ARM. *ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition*, 2012.

[2] A. Celesti, D. Mulfari, M. Fazio, M. Villari, and A. Puliafito. Exploring container virtualization in iot clouds. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–6, May 2016.

[3] K. Cheng, Y. Bai, R. Wang, and Y. Ma. Optimizing soft real-time scheduling performance for virtual machines with srt-xen. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 169–178, May 2015.

[4] T. Cucinotta, G. Anastasi, and L. Abeni. Respecting temporal constraints in virtualised services. In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 2, pages 73–78, July 2009.

[5] Freescale. e500mc Core Reference Manual. Technical report, 03 2013.

[6] Imagination Technologies Ltd. MIPS32 Architecture for Programmers Volume IV-i: Virtualization Module of the MIPS32 Architecture. Technical report, 12 2013.

[7] D. Kozlov, J. Veijalainen, and Y. Ali. Security and privacy threats in iot architectures. In *Proceedings of the 7th International Conference on Body Area Networks*, BodyNets '12, pages 256–262, 2012.

[8] S. Kraijak and P. Tuwanut. A survey on iot architectures, protocols, applications, security, privacy, real-world implementation and future trends. In *11th WiCOM 2015*, pages 1–6, Sept 2015.

[9] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan. Internet of things (iot) security: Current status, challenges and prospective measures. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 336–341, Dec 2015.

[10] Microchip. Pic32mz embedded connectivity with fpu starter kit. http://microchip.wikidot.com/boards:pic32mz-ef, Accessed September 2, 2016.

[11] C. Moratelli, S. Filho, and F. Hessel. Hardware-assisted interrupt delivery optimization for virtualized embedded platforms. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 304–307, Dec 2015.

[12] D. Muench, M. Paulitsch, and A. Herkersdorf. Temporal separation for hardware-based i/o virtualization for mixed-criticality embedded real-time systems using pcie sr-iov. In *Architecture of Computing Systems (ARCS), 2014 27th International Conference on*, pages 1–7, Feb 2014.

[13] A. Patel, M. Daftedar, M. Shalan, and M. Watheq El-Kharashi. Embedded hypervisor xvisor: A comparative analysis. In *Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on*, pages 682–691, March 2015.

[14] prpl Foundation. Security guidance for critical areas of embedded computing. Web, Available at https://prpl.works/security-guidance/. Accessed at 10 ago., 2016.

[15] SysGO. Pikeos 3.3 datasheet, 02 2008.

[16] S. Trujillo, A. Crespo, and A. Alonso. Multipartes: Multicore virtualization for mixed-criticality systems. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 260–265, Sept 2013.

[17] T. Xu, J. B. Wendt, and M. Potkonjak. Security of iot systems: Design challenges and opportunities. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '14, pages 417–423, Piscataway, NJ, USA, 2014. IEEE Press.

[18] S. Zampiva, C. Moratelli, and F. Hessel. A hypervisor approach with real-time support to the mips m5150 processor. In *Quality Electronic Design (ISQED), 2015 16th International Symposium on*, pages 495–501, March 2015.

[19] D. Zhang, D. Liu, L. Liang, L. Yao, K. Zhong, and Z. Shao. Nv-cfs: Nvram-assisted scheduling optimization for virtualized mobile systems. In *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS)*, pages 802–805, Aug 2015.

[20] Z.-K. Zhang, M. C. Y. Cho, and S. Shieh. Emerging security threats and countermeasures in iot. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 1–6, New York, NY, USA, 2015. ACM.

[21] R. Zhou, Z. Ai, J. Yang, Y. Chen, J. Li, Q. Zhou, and K.-C. Li. A hypervisor for mips-based architecture processors - a case study in loongson processors. In *International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th*, pages 865–872, Nov 2013.