D2.7

Base Architectures for virtual/physical computing

Project title:	iDev40: Integrated Development 4.0
Project start date:	1st of May 2018
Duration:	42 months (prolonged due to pandemic situation)

Deliverable type:	Report
Activity and Work package contributing to the deliverable:	Task 2.1.4 / Objective 2.1 / Work Package 2
Due date:	28/02/2021
Actual submission date:	01/03/2021

Responsible lead organisation:	OTH
Lead Author:	Wolfgang Mauerer
Dissemination level:	Public
Reviewer:	Germar Schneider (IFD)
Revision Date:	23/02/2021
Document Reference	D19

This project has received funding from the ECSEL Joint Undertaking under grant agreement No 783163. The JU receives support from the European Union's Horizon 2020 research and innovation programme. It is co-funded by the consortium members, grants from Austria, Germany, Belgium, Italy, Spain and Romania.

Authors

Organization	Name	Chapters
OTH	Wolfgang Mauerer	all
OTH	Ralf Ramsauer	all

Publishable Executive Summary

The Ostbayerische Technische Hochschule Regensburg (OTH) contributed to deliverable D2.7 “Base architectures for virtual/physical computing”. We focused on building novel system architectures. They combine standard IT base systems that provide a wide range of capabilities and features, as present in user-centric standard systems, with the ability to execute trusted, certified and verified industrial operating systems or computational tasks like robot control and signal processing workloads side-by-side on standard hardware.
The emergence of smart factories requires seamless migration of existing legacy fab components (such as deeply-embedded systems, human-machine-interfaces (HMI), custom programmable logic controllers (PLC), embedded control units, and other cyber physical systems (CPS)) to new smart production environments. Edge computing techniques will be used for this purpose, as they not only allow for keeping business-critical aspects within the own premises, but also enable computing real-time and production critical applications with safety relevant aspects on the same highly integrated single control units that leverage modern standard feature-complete IT approaches.

Virtualization technologies play a superordinate role and are a key enabling factor that needs to be well understood on a very low embedded level to make quantitative guarantees about dependability, robustness and integrity of safety-critical system properties. Therefore, we work on reliable software solutions that allow for edge-consolidation of multiple pre-existing system components.
Additionally, safety requirements of many of those system properties require strict conformance to development and engineering processes and certification regulations. By analysing different stages of software-centric development processes, we gain solid, verifiable evidence of their structural characteristics. These efforts ease the conversion from legacy fabs to smart production systems on a functional as well as on a non-functional level.
To deliver results that are significant and useful for the semiconductor industry, we developed practical examples and showcases that demonstrate how to migrate existing legacy solutions (for instance based on Matlab/Simulink) onto our new system.

Our results contribute to reducing R&D costs and R&D time, and allows for increased flexibility in consolidating mixed-criticality workloads in IT appliances for semiconductor industry. Additionally, our contribute reduce hardware and software costs, and improve open innovation collaboration culture by exploiting open source software and mindsets.

Key Words
Cyber-physical systems, edge computing, digital twin, fog computing, mixed criticality, open source, open innovation

Contents

List of Figures

1 Introduction

This report summarises the results of the research teams at OTH Regensburg, as they were contributed to deliverable D2.7, “Base architectures for virtual/physical computing”. D2.7 is part of Task 2.1.4, “Modular standardized toolbox for fab virtualization. This task is mainly embedded in work package 2 of the iDev40 project. Within Task 2.1.4, we focused on building novel system architectures that combine standard IT base systems that offer a wide range of capabilities and features present in user-centric standard systems, with the ability to execute trusted, certified and verified industrial operating systems or computational tasks like robot control and signal processing workloads side-by-side on standard hardware. To ascertain that results significant are for the semiconductor industry, we developed practical examples and showcases that demonstrate how to migrate existing legacy solutions (for instance based on Matlab/Simulink) onto our new system.

Section 2 of this report focuses on the aspect of building mixed-criticality systems by implementing a novel, static partitioning hypervisor based on open source technologies, and guided by the principles of open innovation. Section 3 reports on our results in developing certification technologies that guarantee appropriate quality standards for using our architectures in safety-critical settings. Section 4 discusses measurement-based approaches to quantify the temporal and behavioural guarantees that can be achieved with our platform. Section 5 summarizes our results and concludes this report. The bibliography at the end of this document recaptures our dissemination efforts by listing our publications and presentations written and performed in the context of the iDEV40 project, and also shows how external media reported on our efforts. The publications document the technical details of the results and technological achievements summarised in this report. Note that during the project, considerable amounts of open source code were contributed back to the respective community projects, in accordance with their license criteria. The code repositories are available on www.github.com/lfd.

2 Mixed-Criticality Base Architectures

2.1 Problem Description

Many tasks in the semiconductor industry, especially in the labour-intensive backend business, require a safe, flexible coordination between human workers and automated, smart mechanical tools that are combined with highly flexible, customisable processes. What is common to these requirements is that software, in particular base system software, is required that provides very demanding and specific technical capabilities not found in standard IT products like Windows or MacOS, TODO. Such properties (real-time capabilities, safe operation) cannot be guaranteed by analysing the final outcome of a development process (and be justified and proven by, for instance, extensive testing of the end product), but must be taken into account during the whole engineering lifecycle. Consequently, our solution must address two very disjoint aspects: The provision of appropriate technical capabilities (and a proof of their availability and correctness in the final engineering outcome artefact), and the ability to analyse and assess the development of said artefact from both, a technical and social (with regards to human interaction and cooperation during development) side.

2.2 Solution Overview

Our envisioned solution is a modular standardized toolbox for fab virtualization. We have proposed and implemented a system to combine a standard IT basis that provides a wide range of capabilities and features present in user-centric standard systems with the added ability to execute trusted, certified and verified industrial operating systems or computational tasks like robot control and signal processing workloads side-by-side on standard hardware. This enables substantial integration of systems-of-systems onto a single (or few) individual systems. To ascertain the utility of our approach for semiconductor manufacturing, we have ported our system to the smart workbench described as part of deliverable D2.5 (use-case UC7) in a separate document by OTH Regensburg. The resulting reduction in complexity is illustrated in Figure 1 and Figure 2: The number of required systems could, without any loss in functionality or performance, be reduced by more than 60%. This does not only imply a massive reduction in complexity [4, 8], but also enables substantial cost saving (hardware + software licenses) and a reduction of energy consumption and carbon dioxide footprint in operation. Likewise, administration costs decrease.

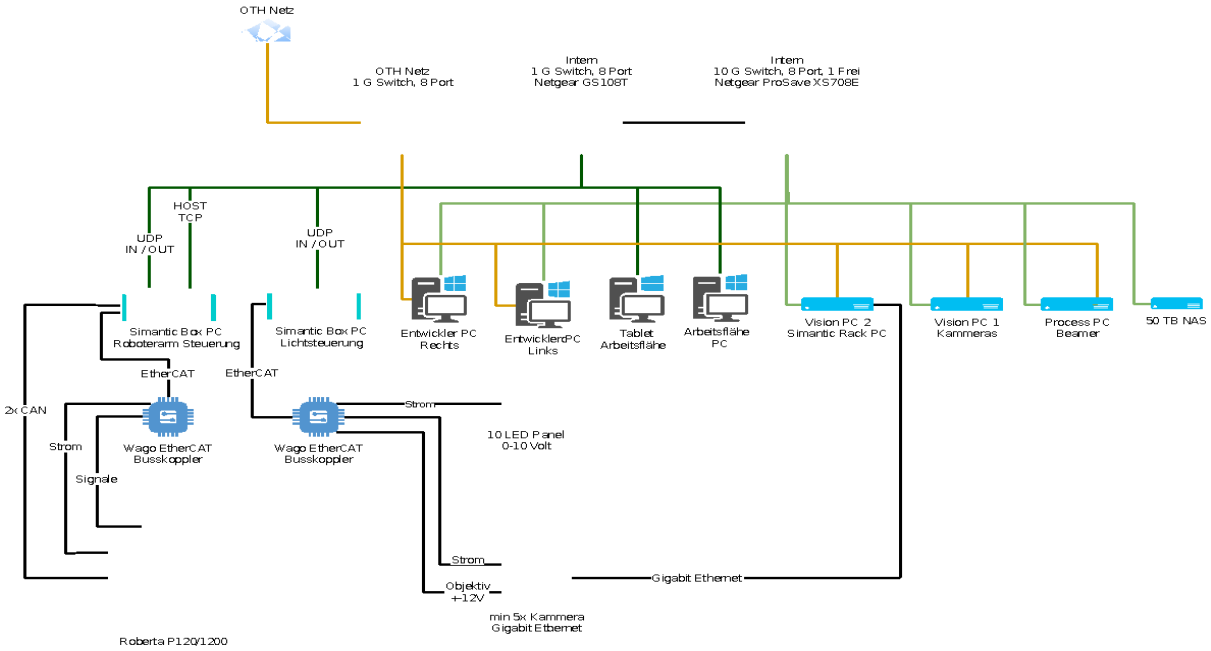

Figure 1: Architectural Diagram of the Smart Workbench of D2.5 (UC7) before deploying the mixed-criticality base system.

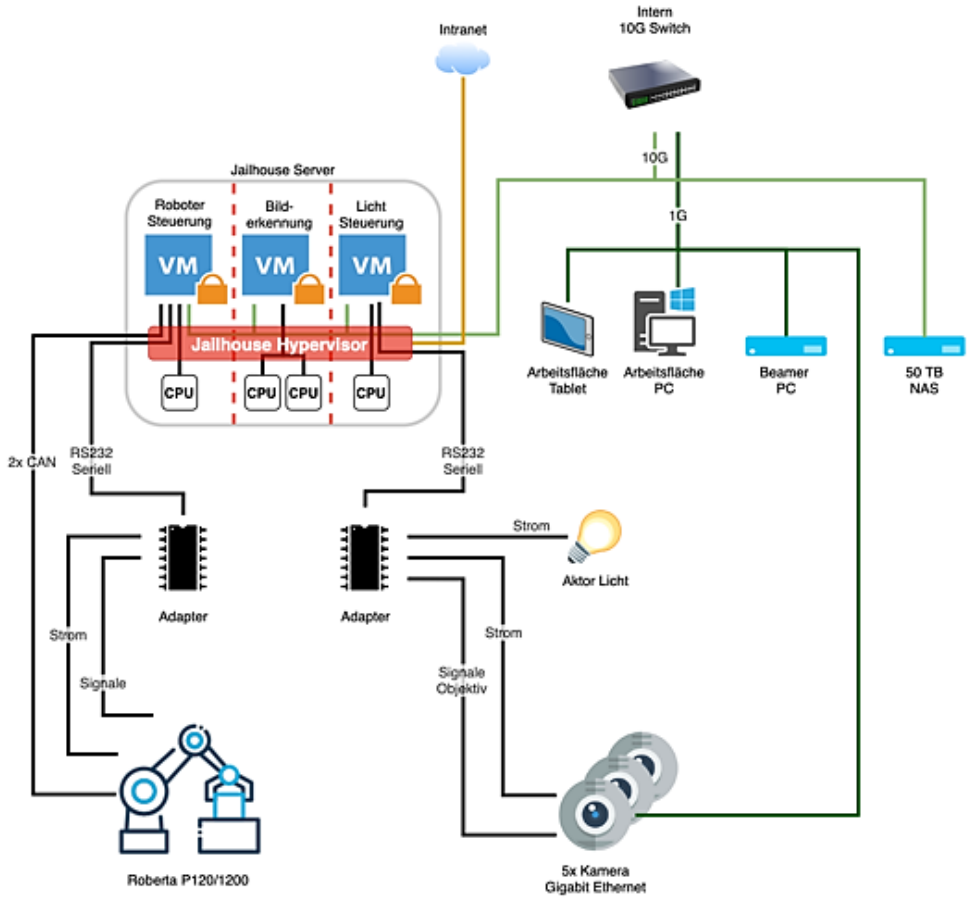

Figure 2: Architectural Diagram of the Smart Workbench of D2.5 (UC7) after deploying the mixed-criticality base system.

Additionally, it is possible to run a digital twin of any appliance as a guest of our virtualised edge-computing system. Real-time production data can be obtained during runtime, for instance to perform runtime performance optimisations. The cloudless approach mitigates security concerns of regular state-of-the-art cloud-based appliances [4, 6, 11] (this aspect is also discussed in Section 4).

2.3 Architecture

For cost effectiveness, many industrial applications cannot give up on the capabilities and feature-richness of Linux in their systems, yet they face increasing demands to simultaneously cope with safety or other certification requirements that are difficult to achieve with Linux. Our architectural approach fulfils these needs. The next figure shows how Jailhouse partitions a quad-core embedded system into multiple disjoint parts. Red lines indicate safety and criticality boundaries enforced by hardware virtualisation mechanisms. Hypervisor intervention is only necessary when illegitimate resource requests are made by payload code, and do not interfere with standard operation.

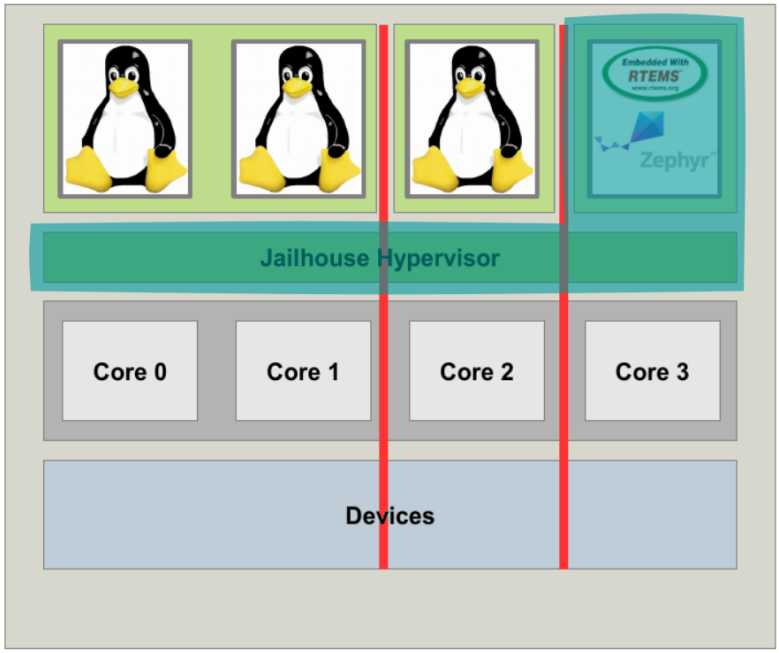

Figure 3: Overview of how Jailhouse partitions a quad-core embedded system into multiple disjoint parts.

Furthermore, consolidation of otherwise separated components becomes important in semiconductor industry: dedicated systems complicate the establishment of a single source of truth, and software maintenance efforts in massively distributed production lines become a major challenge [8].

Multi-core CPUs are a standard component in many modern embedded industrial systems. Their virtualisation extensions enable the isolation of services, and gain popularity to implement mixed-criticality or otherwise split systems. Our approach Jailhouse, a Linux-based, OS-agnostic partitioning hypervisor that uses novel architectural approaches to combine Linux, a powerful general-purpose system, with strictly isolated special-purpose components (cf.

Figure 2). Our design goals favour simplicity over features, establish a minimal code base, and minimise hypervisor activity.

Despite the omnipresence of multi-core CPUs, manufacturers of safety critical and uncritical products still tend to split components with different levels of criticality to separate hardware units. In such traditional mixed criticality environments, single logical control tasks are strongly bound to dedicated physical control units. Consolidating such systems to single hardware units is an architectural trend that does not only improve the maintainability of substantial and growing amount of software, but also reduces the overall hardware costs. The following figure illustrates the boot sequence of the Jailhouse hypervisor. The system requires a fully-set up Linux system. This unusual sequence saves several hundred thousand lines of code required for the complex bring-up of contemporary systems, and allows us to focus certification efforts to less than 10,000 lines of code, as compared to 50 million lines of code that make up a typically configured Linux kernel (or likewise for other systems like Windows, MacOS, or iOS). This drastic reduction of the "attack surface" for safety criticality is a precondition to integrate computationally intensive tasks like big data mining or artificial intelligence-based processing with industrial control tasks on the same hardware.

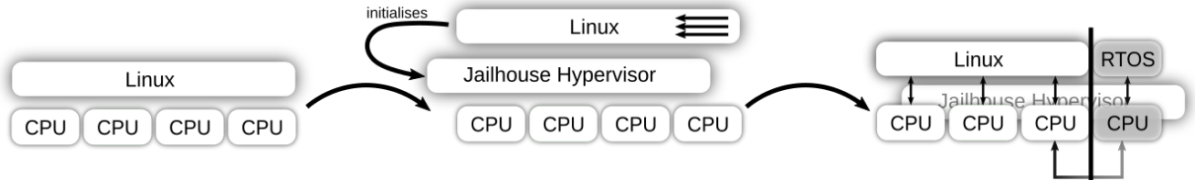

Figure 4: Illustration of the boot sequence of the Jailhouse hypervisor.

Direct assignment of hardware to guests, together with a deferred initialisation scheme (cf. Figure 4), offloads any complex hardware handling and bootstrapping issues from the hypervisor to the general-purpose OS. The hypervisor establishes isolated domains that directly access physical resources without the need for emulation or paravirtualisation (cf. Figure 5). This retains, with negligible system overhead, Linux's feature-richness in uncritical parts, while frugal safety and real-time critical workloads execute in isolated, safe domains.

The minimalist design approach of Jailhouse results in a manageable amount of source lines of code (SLOC). This is a crucial factor for both, formal verification from an academic point of view and system certification from an industrial point of view.

Jailhouse does intentionally not follow classical virtualisation approaches, but its design does not generally eliminate the use of those techniques. This opens the possibility to exploit Jailhouse as an experimental systems platform that allows for keeping focus on the actual problem instead of re-implementing fundamentals from scratch. Jailhouse is an ideal platform for investigating hardware and software behaviour under AMP workloads. Furthermore, it provides a convenient and comfortable environment for executing digital signal processing (DSP)-like workloads on raw hardware.

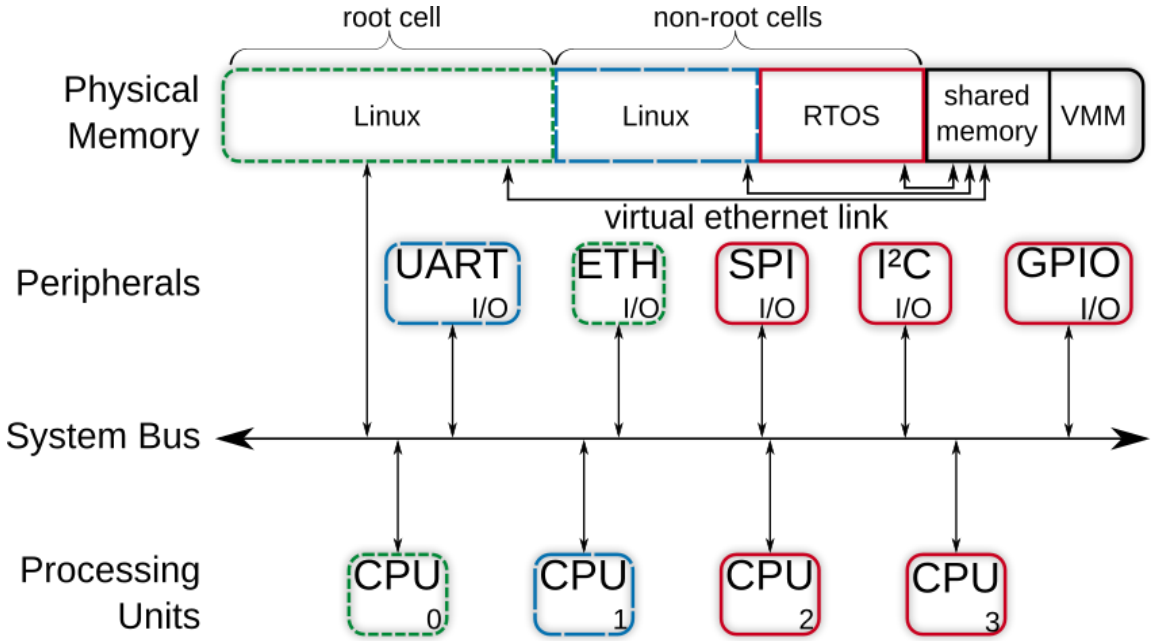

Figure 5: Illustration of how system resources of a single hardware platform are partitioned into multiple disjoint and strictly decoupled criticality domains.

2.4 Conclusion

The static partitioning hypervisor technique is a promising approach for embedded real-time virtualisation, as their ultimate goal to minimise the interaction with guests defers all issues that are introduced by typical paravirtualisation approaches back to the operating systems of the guests, where they already existed before. The driverless approach tries to fill the gap between academic research systems and industrial practicability.

In comparison to paravirtualisation techniques, direct hardware assignment to guests allows for running unmodified legacy payload applications with no active hypervisor overhead. The minimalist hypervisor core simplifies certification efforts. By executing standard operating systems as guests, we also minimised the effort that is required for porting existing legacy payload applications. By implementing a complex demonstration platform, we successfully showed the practicability of hardware partitioning.

While standard virtualisation extensions provided by current hardware seem to suffice for a straight forward implementation of our and many other approaches, real hardware presents a number of limitations that can completely undermine the advantages and guarantees of partitioning and virtualisation-based approaches. Other work performed in the context of the project has investigated how to quantify real-time performance of the hypervisor solution (see Chapter 3), and how it is possible to ascertain the any development performed on the hypervisor satisfies the stringent criteria required for safety-critical systems (see Chapter 4).

3 Development Process Analysis and Certification

Undesired interference of real and virtual systems as presented in the previous chapter needs to be prevented by technical means. However, such isolation is only partially measurable in a functional sense, and is ultimately also dependent on trust in the result of the outcome of an engineering process [1, 3].

Such non-functional system qualification aspects have traditionally been satisfied by manual analysis of processes, and by placing constraints on the development process, which is not feasible with the current complexity of al artefacts that rely on open base systems, as we argue in detail in Ref. [7]. Instead, the increase of software complexity requires careful and targeted deployment of machine learning algorithms to identify strengths and weaknesses of non-functional safety aspects. This enables us to extract proper process models with high quality and accuracy. Decentralized, world-wide developments create huge amounts of communication artefacts (i.e., spatio-temporal big data) that form socio-technical networks [1]. AI and machine learning algorithms help us to understand and interpret these data, in particular to ascertain the possible conformance of the development process with existing regulatory requirements imposed by norms like SIL/ASIL, and other domain-specific safety regulations [3, 4]. A considerable corpus of research on software evolution focuses on mining changes in software repositories, but omits their pre-integration history [5, 7]. Especially this crucial omission has been handled and fixed by us in the context of iDev40 [2, 5].

3.1 Ex-Post Reconstruction of Effective Development Processes

Software patches may have come a long way before their final integration into the official branch (known as mainline or trunk) of a project. There are many possible ways of integration. Among others, the origin of a patch can be a merge from other developers' repositories (i.e., integration of branches or patches from foreign repositories), pull requests on web-based repository managers such as Github or Gitlab, vendor specific patch stacks, or mailing lists (MLs). Especially MLs have been in use for software development processes for decades, and continue to be used for the most crucial components of mixed-criticality systems. Mailing lists form the backbone of their development processes. They are not only used to ask questions, file bug reports or discuss general topics, but implement a patch submit-review-improve strategy for stepwise refinement that is typically iterated multiple times before a patch is finally integrated to the repository (see Figure 6). Compared to conventional, orthodox proprietary industrial software, OSS exhibits different dynamics, and often requires fundamentally different development processes because of project size and a high number of massively geo-dispersed stakeholders [10]. Because of this nature of OSS, projects do not necessarily meet certification criteria. Vendors across different industrial sectors share similar concerns on the use of OSS components, and they are also used as the basis software components of the smart workbench described in iDev40 deliverable D2.5. Hence, their established processes can only be applied to a certain degree. Quantitative ex-post analyses of processes are required to investigate conformance. Statistical methods are necessary to judge the applicability of OSS components in different scenarios. This makes it possible to reconstruct process operations, and use them to draw conclusions on processes with quantitative software-engineering techniques

(Mauerer, et al., 2020). However, how to do this was an unsolved issue in industry that we have rectified with our research.

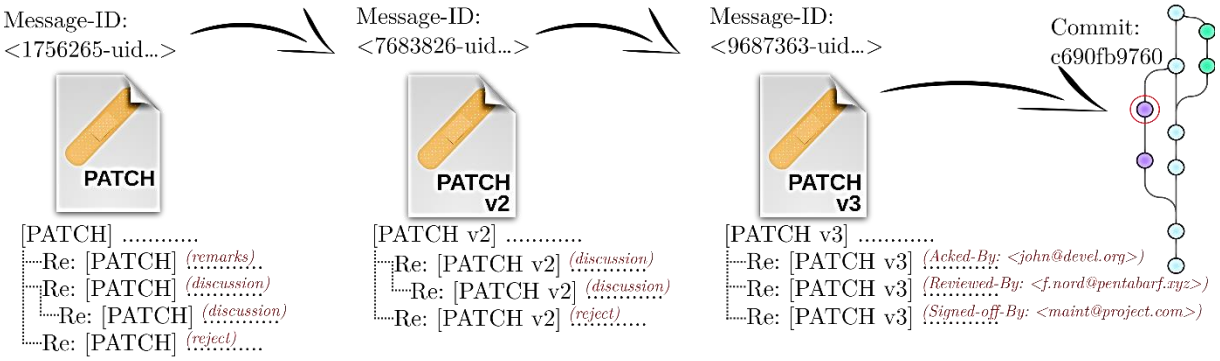

Figure 6: Development and step-wise refinement of a feature in a peer-reviewed, ML-based software development process, and eventual integration of the final artefact into a software repository.

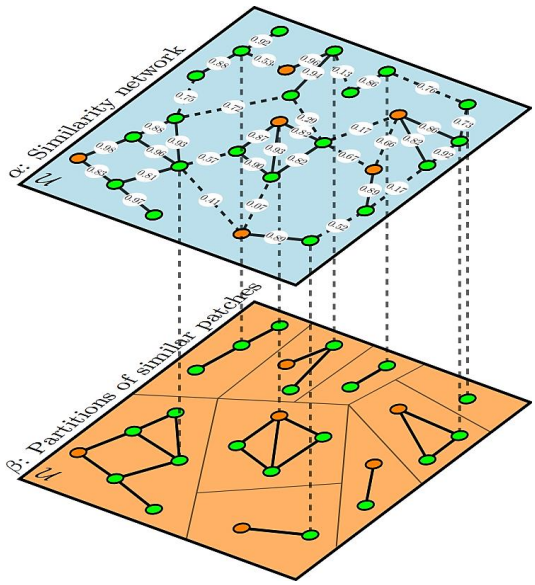

Figure 7: Mapping between a similarity network of patches as obtained by our approach using machine-learning techniques.

3.2 Techniques and Results

Independent of the type of submission, a patch p is formally defined as a 2-tuple that consists of a commit message and a diff. While the commit message informally describes the changes, the diff annotates the actual modifications (insertions and deletions) surrounded by a few lines of context. Context lines ease the understandability of the patch for human review. Patches can also include meta information, such as the author of a patch or the timestamp of its creation (Author Date). Not all types of patches contain the same set of metadata. Emails with patches contain several mail headers, while those headers are removed when the patch is applied to the repository. Repositories, in contrast, contain information on the exact spatial location of the patch. A patch series is a cohesive set of mails that contain several logically connected patches that, in the big picture, introduce one logical change that is split up in fine granular steps. Figure 8 (a) and (b) show two successive mails in a patch series. The submission of a patch or patch

series is typically tool-assisted by the version control system. After patches are submitted, reviewers or any subscriber of the list may comment on them.

Metadata may also change over time; even the author of a patch may change. Therefore, we intentionally do not consider metadata in our similarity analysis. Mapping patches on mailing lists to commits in repositories requires to understand common workflows in projects: When the author of a patch wants his or her patches to be integrated in the project, they need to send their patch or patch series to the mailing list of the project. This process is illustrated in Figure 11. To identify and track such patch relationships, we use the method described in detail in Ref. [7]. An illustration of the result of our machine-learning algorithm can be found in the following figures. These show the genesis of a patch, form the initial proposal via intermediate improvements to the final accepted version. It is obvious to a skilled human observer that the three patches address the same topics, but they differ in many details that make machine-based tracking of similarities challenging.

```
Message-ID: <1338734589-11512-4-git-send-email-tias@ulyssis.org>
Date: Sun,  3 Jun 2012 16:43:05 +0200
To: Discussion and development of BusyBox <busybox.busybox.net>
From: Tias Guns <tias@ulyssis.org>
Subject: [PATCH 3/6] android: fix 'ionice', add ioprio defines

patch inspired by 'BusyBox Patch V1.0 (Vitaly Greck)'
https://code.google.com/p/busybox-android/downloads/detail?name=patch_busybox

Signed-off-by: Tias Guns <tias@ulyssis.org>
---
 include/platform.h |    2 ++
 1 file changed, 2 insertions(+)

diff --git a/include/platform.h b/include/platform.h
index f250624..ba534b2 100644
--- a/include/platform.h
+++ b/include/platform.h
@@ -336,6 +336,8 @@ typedef unsigned smalluint;

 #if defined(ANDROID) || defined(__ANDROID__)
 # define BB_ADDITIONAL_PATH ":/system/sbin:/system/bin:/system/xbin"
+# define SYS_ioprio_set __NR_ioprio_set
+# define SYS_ioprio_get __NR_ioprio_get
 #endif



1.7.10
```

Figure 8: Genesis of a patch initial proposal

```
Message-ID: <1338734589-11512-3-git-send-email-tias@ulyssis.org>
Date: Sun,  3 Jun 2012 16:43:04 +0200
To: Discussion and development of BusyBox <busybox.busybox.net>
From: Tias Guns <tias@ulyssis.org>
Subject: [PATCH 2/6] android: use BB_ADDITIONAL_PATH

Signed-off-by: Tias Guns <tias@ulyssis.org>
---
 include/platform.h |    4 ++++
 1 file changed, 4 insertions(+)

diff --git a/include/platform.h b/include/platform.h
index d79cc97..f250624 100644
--- a/include/platform.h
+++ b/include/platform.h
@@ -334,6 +334,10 @@ typedef unsigned smalluint;
 # define MAXSYMLINKS SYMLOOP_MAX
 #endif

+#if defined(ANDROID) || defined(__ANDROID__)
+# define BB_ADDITIONAL_PATH ":/system/sbin:/system/bin:/system/xbin"
+#endif
+

 /* ---- Who misses what? ----------------------------------- */

--
1.7.10
```

Figure 9: Genesis of a patch with intermediate improvements

```
commit 3645195377b73bc4265868c26c123e443aaa71c6
Author: Tias Guns <tias@ulyssis.org>
Date:   Sun Jun 10 14:26:32 2012 +0200

    platform.h: Android tweaks: ioprio defines, BB_ADDITIONAL_PATH

    Signed-off-by: Tias Guns <tias@ulyssis.org>
    Signed-off-by: Denys Vlasenko <vda.linux@googlemail.com>

diff --git a/include/platform.h b/include/platform.h
index d79cc97..ba534b2 100644
--- a/include/platform.h
+++ b/include/platform.h
@@ -334,6 +334,12 @@ typedef unsigned smalluint;
 # define MAXSYMLINKS SYMLOOP_MAX
 #endif

+#if defined(ANDROID) || defined(__ANDROID__)
+# define BB_ADDITIONAL_PATH ":/system/sbin:/system/bin:/system/xbin"
+# define SYS_ioprio_set __NR_ioprio_set
+# define SYS_ioprio_get __NR_ioprio_get
+#endif
+

 /* ---- Who misses what? ----------------------------------- */
```

Figure 10: Genesis of a patch final accepted version

The extensive use of string metrics for measuring the similarity of different parts of a patch opens a wide spectrum for different thresholds of similarity. Additional parameters investigate the structure of the patch and control the sensitivity of the comparison. Without describing the computational details in this summary, let us just describe outcome and benefits: With optimised settings for prediction accuracy, our approach achieves a Fowlkes-Mallows index of 0.911. This considerably exceeds previous work: Jiang and colleagues also present a method for mapping patches on mailing lists to repositories, which has been previously deemed best compared to other approaches. Their Plus-Minus-based approach assigns each tuple of changed line and filename to a set of ids, where the id can either be a message ID or a commit hash. They then search for patches that contain sufficient identical changes. A threshold between [0, 1] determines the fraction of the number of identical changes that needs to be exceeded if patches are considered similar. We used their original implementation to evaluate it against the time window of our ground truth, and vary their threshold setting in the range [0,1]. The best Fowlkes-Mallows index of 0.743.

3.3 Discussion and Summary

The industrial deployment of OSS is often hindered by required certification of their non-formal development processes according to relevant standards, such as IEC 61508 for safety-critical industrial, or ISO 26262 for safety-critical automotive software. Even though the open and community- driven development process of OSS provides full traceability of its development, most of the information is not explicitly contained in the repository, but implicitly hidden in semi-formal discussions on mailing lists.

We presented a method that is able to reliably link emails with patches to commits in repositories with high accuracy. Additionally, we formalised the mathematical background of the problem and identified it as a clustering problem. Based on this, an elaborate evaluation built upon a solid ground truth quantifies the high accuracy of our approach. The ground truth and our framework can be used to evaluate the accuracy of other approaches, and the fully published framework allows for independent (industrial) evaluation required in certification efforts.

The evaluation verified that the presented approach performs better than existing work. For Linux and the LKML, we achieve a 22% larger Fowlkes-Mallows index of 0.911 than the best score achieved by the (previously best) plus-minus-line–based approach.

We have applied our method to the software components used to implement the smart workbench as delivered in D2.5, and have provided a basis for formal certification efforts that are required to put the system into industrial production use (see Ref. [7] and [2]). This allows us to achieve substantial reductions in software license and hardware costs, while at the same time gaining many additional functional capabilities, as compared to state-of-the-art solutions.

4 Measurement-Based Quantification of Real-Time and Isolation Guarantees

4.1 Overview

An important industrial requirement on real-world systems is that it must be possible to guarantee (under a reasonable definition of assurance) that partitioning implies the maintenance of determinism within a computing domain and freedom of interference between the partitioned domains [4, 8].

Not only on partitioned or virtualised, but also on conventionally scheduled systems, the surface of potential cross-domain interference is determined by the degree of interaction between different computing domains. This includes interactions between tasks, tasks and operating systems, operating systems and an underlying hypervisor, and interference of the system's low-level firmware.

As mentioned before, the aim of Jailhouse is to minimise the activity of the hypervisor. Though this would be possible in theory, the sole existence of a hypervisor introduces additional latencies that do not exist without a VMM. For example, shadow page tables may introduce additional memory access latencies because of additional steps in the page table walk in case of TLB misses.

To evaluate and determine the (real-time) performance of the hypervisor, several environmental conditions must be considered. It is hard or even impossible to quantify the hypervisor overhead with one single scalar. This results in a set of microbenchmarks that serve as a basis for a specific decision on the qualification of the system's architecture.

For all benchmarks, single-shot measurements do not allow to draw any conclusions on the behaviour of the system [9]. Microbenchmarks should be repeated under certain environmental conditions, such as the actual existence of a hypervisor, and the particular frequency of a certain measurement together with variations of the utilisation of other guests.

In the following, we describe considerable measurements and present three quintessential microbenchmarks and compare different platforms: the hypervisor-induced interrupt latency on an ARM platform, the moderation of the ICR register of the interrupt controller of x86 platforms, and the impact of Spectre mitigations on virtualised systems. These are three sources of workload-dependent hypervisor activity and hence, hypervisor noise.

4.2 Measurements

The following paragraphs give a brief overview of potential sources of interference and noise. It is important to remark that benchmarks do not measure the overhead of the hypervisor, but the overhead of the hypervisor when running on a specific hardware platform. Still, those measurements allow us to derive a trend of the performance of the hypervisor. Determining latencies relevant for real-time control processes as they are found in semiconductor

manufacturing, which may involve reaction and cycle times that go as low as 10-100 us, requires us to observe systemic delays at a much lower level than usual for throughput performance measurements. We find that several factors are significant sources of delays:

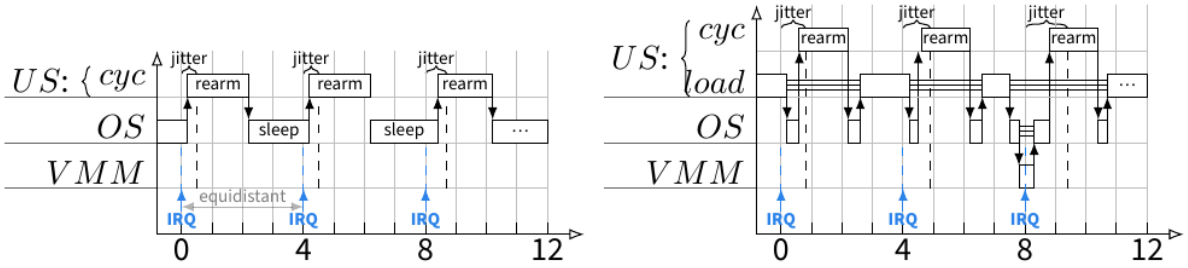

Figure 11: Synthetic measurement setup to determine cycle time variation based on responses to external stimuli.

- One typical benchmark for hypervisors is the cost of hypercalls, as hypercalls are often use to implement paravirtualisation of devices. In case of Jailhouse, hypercalls do not need to be considered, as they are only used for cell management purposes, and never occur in the operational state or in hot paths.
- Jailhouse implements an AMP system. Different guests asynchronously access memory, and memory or I/O access may be serialised by hard- ware. Though starvation does not occur on supported architectures, heavy utilisation of memory or I/O busses may lead to significant slow-downs of guests. While this problem is well-known for SMP applications, its impact must be evaluated when asynchronously executing multiple payloads that were designed for single-core platforms.
- While on x86 architectures, VT-d, for example, allows for direct trap-free remapping of MSI-X interrupts to guests, many ARM platforms miss equivalent extensions. While some ARM64 platforms support Software Delegated Exception Interface (SDEI), an extension that can be exploited to implement trap-free interrupt injection to guests, it is practically not available for 32-bit ARM platforms; interrupts must be reinjected by the hypervisor. A measurement of the interrupt latency is shown in Table 1.
- Because of architectural limitations, Jailhouse needs to emulate devices that are essential for a hardware platform and that cannot be virtualised in hardware, such as the interrupt distributor as part of the GIC in ARM architectures, or the ICR, as part of the interrupt controller on modern Intel x86 platforms. Depending on the utilisation of those devices, the impact of the hypervisor must be analysed. In Figure 12, we quantify the latency overhead that is caused by hypervisor activity that is required to moderate accesses to the ICR on a modern Intel platform.

Table 1: Interrupt reinjection delays

VMM	Freq	Stress	μ	σ	Max
off	10Hz	no	0.45	0.02	0.50
off	50Hz	no	0.45	0.02	0.50
on	10Hz	no	1.26	0.07	2.81
on	50Hz	no	1.25	0.04	2.94
on	10Hz	yes	1.36	0.34	5.56
on	50Hz	yes	1.35	0.34	5.38

Another considerable but unexpected factor that came into existence during the iDev40 project is a class of CPU-based information security leaks spearheaded by the now infamous Spectre attack. Since safety implies security as a precondition, we had to meet these challenges in our system architecture. Refer to [4], [11] and [12].

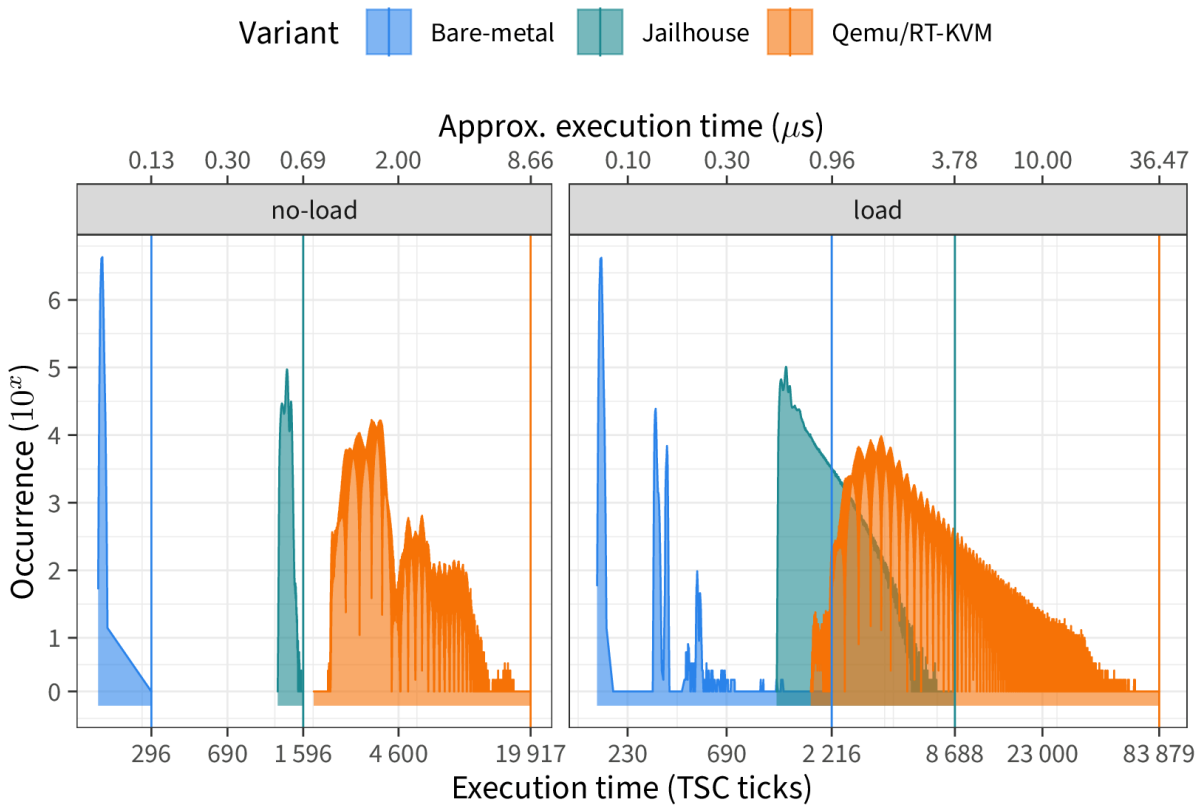

Figure 12: Histogram of the execution time of WRMSRs to the ICR register of the x2APIC in TSC ticks and approx. transformed execution time in us. Vertical lines denote the maximum latency of a measurement. Both axes are scaled logarithmically.

Mitigations for Spectre-like attacks heavily rely on measures by system software and firmware, as well as updates for CPU microcode. For example, Meltdown enforces operating systems to isolate user and kernel address space to separate page tables, which is a cost-intensive change, as it requires TLB flushes on context switches. As mentioned before, in case of TLB misses, the existence of a hypervisor introduces additional paging layers, which, in turn, leads to higher latencies for page table walks. Other platforms require assistance of the system-firmware to

mitigate the attack surface. To mitigate, for example, CVE-2017-5715 (Spectre v2), ARM64 platforms implement a firmware-based workaround. The OS conducts a pseudo firmware call (i.e., Secure Monitor Call (SMC)) that, as a side effect, results in a speculation barrier. However, a firmware call by a guest OS is trapped by the hypervisor which needs to moderate and forward the call to the firmware interface. In Jailhouse, we implement a fast dispatcher path for these kind of SMC calls. Nevertheless, those traps introduce additional latencies when speculation barriers are conducted. As a base-line, we have considered the impact of Spectre mitigations on conventional non-virtualised real-time systems, as well as their impact on virtualised, partitioned environments. Three quintessential microbenchmarks for four different platforms in total are provided as an overview; for exact technical details and more comprehensive measurements, we refer to Ref. [4]. We will measure specific aspects of the overhead that is introduced by the hypervisor on modern Intel x86 platforms (Xeon E5-2683 v4 and a Xeon Gold 5118) platform (see Figure 13) a 32-bit ARMv7 platform (Nvidia Jetson TK1) and a 64-bit ARMv8 platform (Nvidia Jetson TX1, see Figure 14).

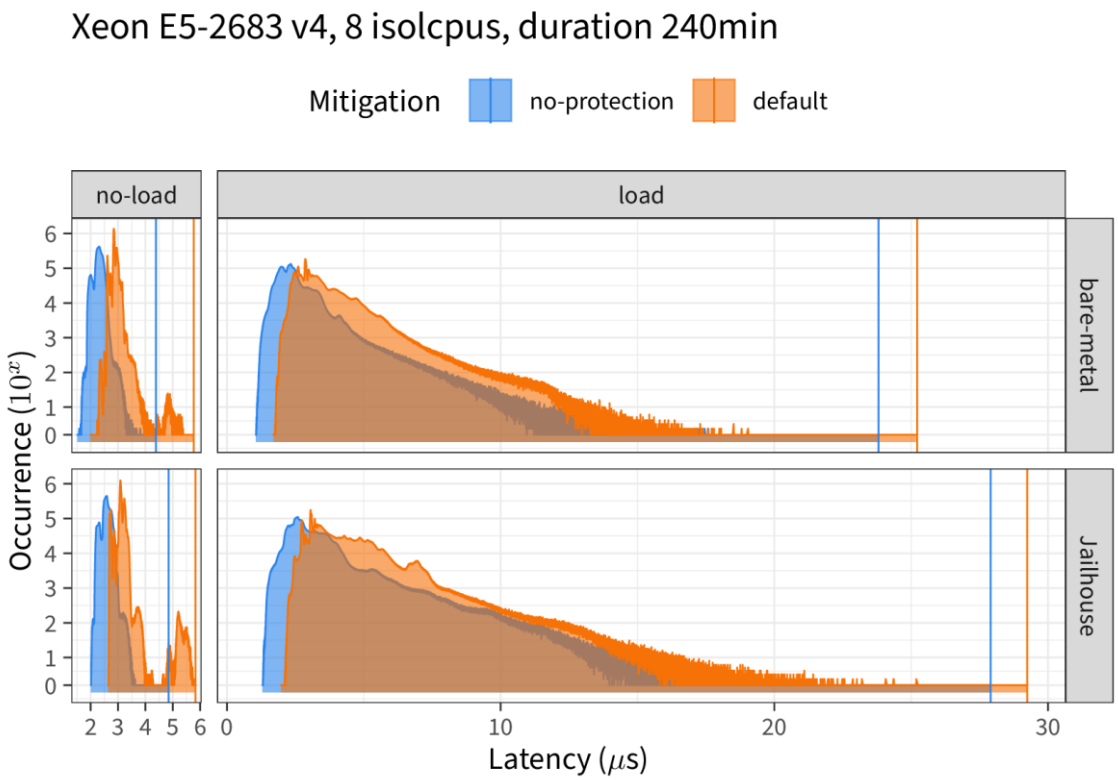

Figure 13: Intel Xeon E5-2683 v4: Latency histogram of the influence of mitigations against speculation attacks on determinism and response latencies of a Preempt-RT extended Linux based system. Left side: without additional system load. Right side: measuring CPUs are put under additional non-real-time load.

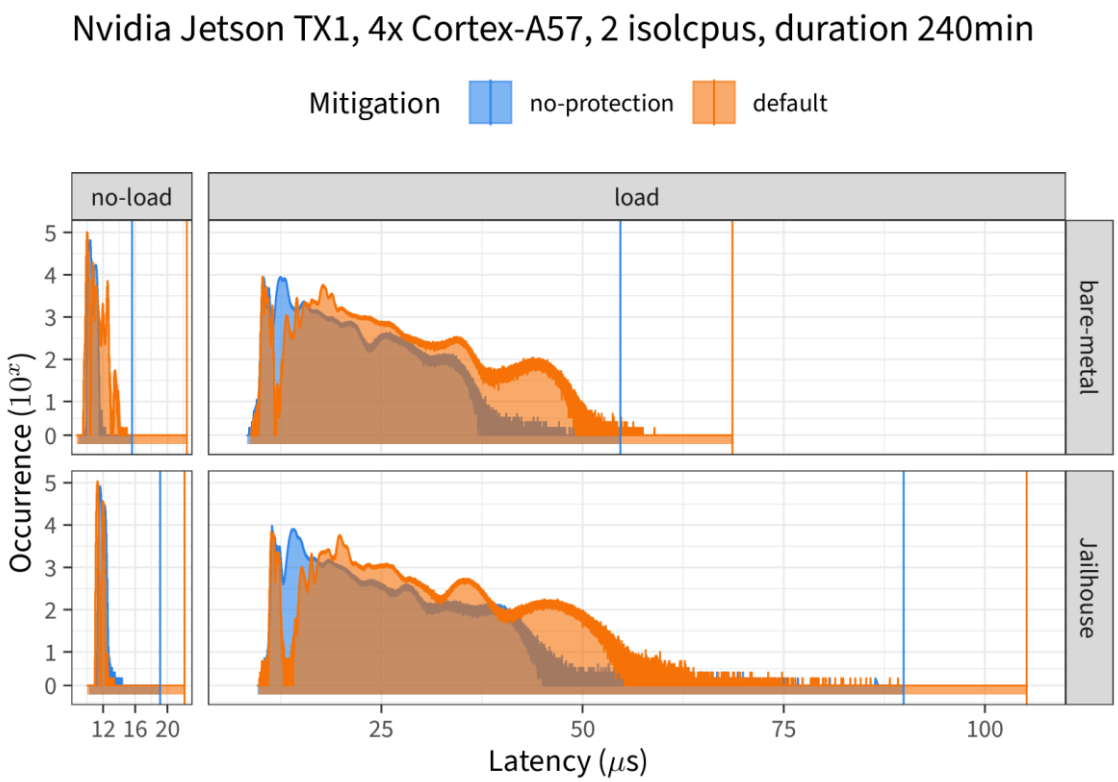

Figure 14: Nvidia Jetson TX1: Latency histogram of the influence of mitigations against speculation attacks on determinism and response latencies of a Preempt-RT extended Linux based system. Left side: without additional system load. Right side: measuring CPUs are put under additional non-real-time load.

4.3 Evaluation and Conclusion

The implementation of mixed-criticality systems requires safe isolation. Primitives that can be used to achieve safe isolation, in turn, depend on mechanisms provided by hardware. On the one hand, uncritical domains of a system demand for high flexibility with respect to their field of application, on the other hand, safety critical applications demand for strict isolation from other domains of the system. The one extreme is the individual conception of tailored hardware-processor architectures that precisely fulfil the needs of a specific system. This approach is accompanied by high hardware development costs and complicates future reconfigurations of the payload software. The other extreme is to purely rely on software-based isolation primitives - OSs use standard isolation mechanisms and shall guarantee a sufficient level of isolation.

Yet, this approach is accompanied by lower hardware costs, but also by lower isolation guarantees. Virtualisation extensions of COTS hardware allow to introduce strong isolation barriers that are otherwise not used by standard system software.

We presented the concepts of Jailhouse, a real-world Linux-based static partitioning hypervisor. Static hardware partitioning is a promising approach to fill the gap between both extremes: it enables dynamic reconfiguration of complex systems, and exploits strong hardware-based isolation mechanisms that are provided by economic COTS hardware components. For embedded real-time virtualisation, its ultimate goal is to minimise the interaction with guests.

All issues that are introduced by conventional (para-)virtualisation approaches are deferred back to the operating systems of the guests, where they already existed before. Furthermore, the driverless approach tries to fill the gap between academic research systems and industrial practicability. In comparison to paravirtualisation techniques, direct hardware assignment to guests allows for running unmodified legacy payload applications with no active hypervisor overhead. The minimalist hypervisor core simplifies certification efforts. By executing standard operating systems as guests, we also minimised the effort that is required for porting existing legacy payload applications. By implementing a complex demonstration platform, we successfully showed the practicability of hardware partitioning.

While standard virtualisation extensions provided by current hardware seem to suffice for a straight forward implementation of our and many other approaches, real hardware presents a number of limitations that can completely undermine the advantages and guarantees of partitioning and virtualisation-based approaches. Hence, we showed in quintessential benchmarks that are required to quantify the performance of the hypervisor and to assess the suitability of the approach for specific use cases.

The measurements evaluate the impact of Spectre mitigations on real-time systems. We show that for many real-world use cases that require Linux to run side by side with real time operating systems, static hardware partitioning is a viable alternative to classical OS-based isolation approaches, especially when legacy workloads must be protected against hardware weaknesses like Spectre-class speculation attacks without modifying certified components.
Finally, we defined the concepts of ideal VMMs, ideal partitions and ideal partitioned systems with the goal of establishing zero-trap hypervisors on real-world systems that only need to account for setting up partitions, but do not interact with the content of any partitions in the operational phase. Experiments with an implementation of the concept on multiple hardware platforms showed limitations inherent in current hardware. We discussed necessary improvements in future virtualisation techniques to facilitate a realisation of our approach on realistic systems.

5 Conclusion

The implementation of base architectures for virtual/physical computing systems necessitates a proof of concept implementation in a small-scale. This was realised in cooperation with UC7, and our results form a building block for future industrial virtual computing platforms.
Real and virtual production data can be seamlessly interchanged in realistic industrial setups; co-simulation/optimization performed on the virtual digital twins can be fed back to the physical system during online operation with best possible accuracy in real time; integration of new techniques and components into existing appliances can be tested and optimized on production appliances. Cost and complexity of cyber-physical edge computing systems and digital twin-based platforms are substantially reduced by re-using feature-complete IT mechanisms and the associated development environments (e.g., App mechanisms) and certified industrial codes (e.g., existing specialized operating systems), as we have shown with the explicit construction of a working demonstrator of realistic complexity.

Our work has involved many facets of modern IT systems, from high-level end-user applications to low-level hypervisors and operating systems, which are primarily based on Open Source Software (OSS) —the complete vertical software stack had to be adapted and engineered to satisfy our ambitious goals. It turned out that the planned approach of using open solutions was a successful and technologically reasonable choice to reach our goals.

We have continuously and stepwise refined and improved incorporated OSS components. Our impact on OSS communities on a social and human level is accounted by numerous contributions to essential projects in the affected domains.

Being participant and part of many OSS communities, we regularly presented our work on both, highly-ranked international industrial as well as academic conferences to several hundreds of participants. Our active contributions to emerging technologies allowed us to shape the definition of future industrial de-facto standards for industrial edge computing

6 List of Abbreviations

Abbreviation	Meaning
COTS	Commercial off-the-shelf
OS	Operating System
OSS	Open-Source Software
OTH	Ostbayerische Technische Hochschule
MC	Mixed Criticality
RT	Real-Time
VM	Virtual Machine
VMM	Virtual Machine Monitor

8 List of References

External Reports

[1] Wolfgang Mauerer, Mitchell Joblin, Damian A Tamburri, Carlos Paradis, Rick Kazman, Sven Apel (2021): In Search of a Socia-Technical Congruence: A Large-Scale Longitudinal Study, in: IEEE Transactions in Software Engineering.

[2] Ralf Ramsauer, Lukas Bulwahn, Daniel Lohmann, Wolfgang Mauerer (2020): The Sound of Silence: Mining Security Vulnerabilities from Secret Integration Channels in Open-Source Projects, In: CCSW'20: Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop, DOI: https://doi.org/10.1145/3411495.3421360.

[3] Ralf Ramsauer, Sebastian Duda, Wolfgang Mauerer (2019): The list is our Process: An analysis of the Kernel's email- based Development Process, In: Embedded Linux Conference Europe / Lyon, France, Link: https://www.youtube.com/watch?v=YCfU-2dXDq0.

[4] Ralf Ramsauer, Jan Kiszka, Daniel Lohmann (2021): Principles and Practice of Hypervisor-Assisted Real-World Hardware Partitioning, In: IEEE (under review).

[5] Ralf Ramsauer, Wolfgang Mauerer, Lukas Bulwahn (2019): The list is our process: An analysis of the Kernel's email- based Development Process, In: Linux Plumbers Conference, Portugal, Lisbon, Link: https://www.youtube.com/watch?v=QG1YDQ1HOKE

[6] Ralf Ramsauer, Jan Kiszka, Wolfgang Mauerer (2018): Spectre and Meltdown vs. Real-Time: How Much do Mitigations Cost?, Link: https://www.youtube.com/watch?v=nqU4j2M_Ul4

[7] Ramsauer, Ralf and Lohmann, Daniel and Mauerer Wolfgang (2019): The list is the process: Reliable pre-integration tracking of commits on mailing lists, In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE).

[8] Ramsauer, Ralf and Kiszka, Jan and Lohmann, Daniel and Mauerer, Wolfgang (2017): Look mum, no VM exits! (almost), In: Proceedings of the 13th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT '17).

[9] Mauerer, Wolfgang and Ramsauer, Ralf and Lucas, Edson R. F. and Scherzinger, Stefanie (2021): Silentium! Run-Analyse-Eradicate the Noise out of the DB/OS Stack, In: Fachtagung für Datenbanksysteme für Business, Technologie und Web (BTW '21).

Thesis

[10] Pia Eichinger, 2020, Maintainers Expectations vs. Maintainers Reality. An Analysis of Organisational and Maintainence Structures of the Linux Kernel, Bachelorarbeit OTH Regensburg. Link: https://www.youtube.com/watch?v=YDNzKGTl_PY (presented at LinuxCon Australia)

[11] Nathan Hausmann, 2020, Erstellung und Evaluierung von Benchmarks für Echtzeitanalysen von Hypervisoren unter Linux, Masterarbeit OTH Regensburg.

[12] Andrej Utz, 2020, Einfluss der Spectre-Mitigations auf Echtzeitfähigkeit des Jailhouse-Hypervisors auf modernen AMD Systemen, Bachelorarbeit, OTH Regensburg (In Zusammenhang mit [6] enstanden) Link: https://www.youtube.com/watch?v=nqU4j2M_Ul4; Link to Thesis: https://cdn.lfdr.de/ba-utz.pdf