PIERES: A Playground for Network Interrupt Experiments on Real-Time Embedded Systems in the IoT

Franz Bender
franz.bender@icloud.com
Technische Universität Berlin
Berlin, Germany

Jan Jonas Brune
jan.j.brune@campus.tu-berlin.de
Technische Universität Berlin
Berlin, Germany

Nick Lauritz Keutel
keutel@campus.tu-berlin.de
Technische Universität Berlin
Berlin, Germany

Ilja Behnke
i.behnke@tu-berlin.de
Technische Universität Berlin
Berlin, Germany

Lauritz Thamsen
lauritz.thamsen@tu-berlin.de
Technische Universität Berlin
Berlin, Germany

ABSTRACT

IoT devices have become an integral part of our lives and the industry. Many of these devices run real-time systems or are used as part of them. As these devices receive network packets over IP networks, the network interface informs the CPU about their arrival using interrupts that might preempt critical processes. Therefore, the question arises whether network interrupts pose a threat to the real-timeness of these devices. However, there are few tools to investigate this issue.

We present a playground which enables researchers to conduct experiments in the context of network interrupt simulation. The playground comprises different network interface controller implementations, load generators and timing utilities. It forms a flexible and easy to use foundation for future network interrupt research. We conduct two verification experiments and two real world examples. The latter give insight into the impact of the interrupt handling strategy parameters and the influence of different load types on the execution time with respect to these parameters.

CCS CONCEPTS

• **Computer systems organization** → *Embedded hardware*; • **Computing methodologies** → *Simulation tools*; • **Hardware** → *Testing with distributed and parallel systems.*

KEYWORDS

internet of things, real time, interrupts, load simulation, cyber physical systems, benchmarking

ACM Reference Format:
Franz Bender, Jan Jonas Brune, Nick Lauritz Keutel, Ilja Behnke, and Lauritz Thamsen. 2021. PIERES: A Playground for Network Interrupt Experiments on Real-Time Embedded Systems in the IoT. In *Companion of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21 Companion), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3447545.3451189

1 INTRODUCTION

Many processes in industrial settings have real-time constraints. Engineers need guarantees for these processes, e.g. that an action A takes at most N seconds. Real-time operating systems (RTOSs) have proven to be a suitable platform for implementing software by providing such guarantees. RTOSs offer, in contrast to regular operating systems, special interfaces for precise timing and scheduling of time critical tasks [9]. These RTOSs have therefore found application in the field of embedded systems such as sensor systems, industrial control systems and many other specialized devices. For researchers and engineers using RTOSs it is essential that the real-timeness of their device is maintained.

With the rise of the Internet of Things (IoT) many of these devices get connected to the internet. This trend has been described as a new era [10]. It enables features such as remote monitoring, remote debugging and a more intelligent management of devices by combining data from multiple devices for decisions. You could take smart navigation systems as proposed in [3] as an example for the latter.

Modern network interface controllers (NIC) such as the Intel©82574 GbE Controller Family use interrupts to inform the CPU about new packets. This implies that other network devices have the ability to invoke interrupts at the target. As interrupts are handled by interrupt service routines (ISRs), which have a very high execution priority, these interrupts may interfere with user code. As the rate of network interrupts increases, more time is spent in the ISR instead of the user code [1].

This behavior is sound as ISRs reside in a higher priority space to minimize the I/O delay of the embedded system. However, while being a valid phenomenon, this behavior may not be intended by the engineer of the device, as network packets might not be as important as the critical user code. Note that the high number of interrupts may be caused by a malicious attacker or by other non-malicious conditions such as a bad network configuration or other similar conditions. All in all it means that adding network capabilities to real-time systems adds a per-packet workload which may drown critical tasks and break time guarantees.

In this paper we present a playground which enables researchers and engineers to tackle these issues. The playground

- can simulate network interrupts on a microcontroller to help analyze the impact of network traffic on applications running on IoT devices.
- offers simulation of continuous and Poisson-distributed[1] network interrupts, as well as replays of captures of actual network traffic

The remainder of this paper is structured as follows. Section II gives an overview of the related workSection III details the approach we took to design the playground. Section IV details the experiments we conducted using our playground. Finally, Section V summarizes our work.

2 RELATED WORK

Available literature focusses mostly on mitigation strategies.

As a general example, *interrupt moderation* or *coalescing* [8] is used by the authors of [2] to reduce energy consumption in a system for high speed networks. Multiple interrupts are grouped to invoke a single interrupt at a later time.

Other approaches rely on the optimization of the interrupts themselves. The authors of [6] use that the interrupt mechanism can be split up into preprocessing, the ISR and postprocessing. They exploit that the preprocessing and the postprocessing are independent of the interrupt itself and "reuse" these phases for multiple interrupts and only calling the ISR for every interrupt.

While these authors worked on mitigation strategies, others conducted research on the interrupts themselves. This resulted in different models for network interrupts that differ in their approaches: The authors of [4] present a complex model for the firing and execution of interrupts. They use an extension of stochastic Petri nets to model the system as this allows them to combine the probabilistic aspects caused by the randomness of the interrupts with the stateful aspects of prioritized interrupt handling.

A more basic model was presented in [7]. There, a double interrupted poisson process is employed for their calculations. This is a poisson process that can be can be in either a high or a low state which determines the poisson parameters for the load. They conclude by giving formulas for blocking probabilities, i.e. the probability that a link is blocked and packets are dropped.

To the best of our knowledge, there is no work that focusses on providing a testing capability for the impact of network interrupts on real-time systems.

3 APPROACH

This section describes the goal of the playground, how to enable different NIC implementations and the choice of network traffic scenarios as test loads, which are detailed in the following.

3.1 Goal of the Playground

The goal of this playground is to enable researchers to conduct experiments and to enable engineers to test their code regarding network interrupt simulation. This shall be done on an actual device used for IoT applications. Therefore, the playground has to fulfill the following requirements: It has to

- run on real IoT hardware.

[1]A Poisson distribution is commonly used to model traffic in the literature, e.g. [7]

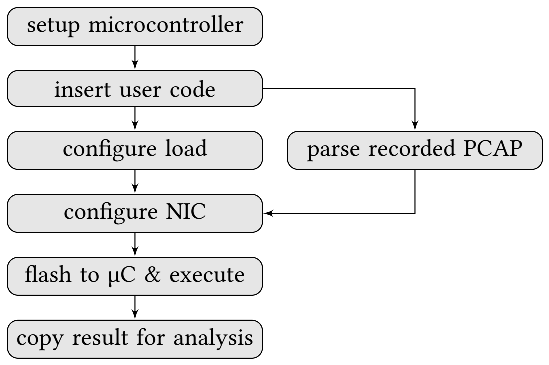

Figure 1: General procedure of a user operating the playground.

- be capable of simulating multiple NIC implementations.
- be able to simulate multiple network traffic scenarios.
- be easily configurable.
- have minimal performance impact on the tested process.

3.2 Operation of the Playground

The operation of the playground is shown in Fig. 1. The playground is setup on the microcontroller. The user then inserts the code of the critical process under test and either chooses a configuration for a Uniform, a Random Load, or parses a recorded network trace. Afterwards, the user configures the NIC and flashes the playground onto the device. Measured metrics can be configured by the user and range from execution time and number of interrupts to more complex metrics such as the ratio between interrupt sources (see Sec.4).

3.3 NIC Implementations

The playground has to model different NICs, that can be configured by the user. The simplest NIC would inform the CPU at the arrival of every packet by triggering an interrupt for each. Alternatively, some modern NICs offer interrupt moderation. To incorporate this into the playground, we offer the choice between a simple NIC model without any interrupt moderation and several smarter NIC implementations.

For the simple NIC, the duration $d(l)$ of an interrupt is dependent on the packet length l and plainly modeled as a length dependent delay d_l which is evoked l times plus constant length independent delay d_c which is an overhead evoked for every interrupt:

$$d(l) = d_l \cdot l + d_c$$

Different simple NIC implementations can be characterized by the user through setting the values for the length dependent delay d_l and the length independent delay d_c.

For smarter, more elaborate NIC simulations with interrupt moderation, we support NICs to be defined with a counter mode, a timer mode or a combination out of the box. Here, the parts of the interrupt duration model, packet length and corresponding dependent and independent delay, are modeled twice each, once for the simulated ISR and once for a simulated receiver task.

A NIC with the counter mode does not trigger an interrupt for every received packet, but counts the arriving packets, stores them in a buffer and – after a specified number of packets – evokes one interrupt for them all. Afterwards, the counter and buffer are reset. Another option is a NIC with the timer mode. In this case, for an arriving packet a delay timer of specified duration is set. Upon expiration, one interrupt will be triggered. If further packets arrive before the timer has run out, the timer will be reset without evoking an interrupt. However, problems may arise if the timer is constantly being reset by arriving packets, never allowing an interrupt to be triggered. The combination of both modes circumvents this problem.

3.4 Network Traffic Scenarios as Load

The arrival of packets with corresponding time stamps over some observed time constitutes a load scenario. As we want to simulate different scenarios, the playground offers uniform loads, random loads and user defined/recorded loads.

Uniform loads have a constant receive frequency. For the randomized loads, a Poisson distribution is used to model the arrival of new packets.

The Poisson distribution is achieved by inverse transform sampling with a uniform distribution. Assuming the number of incoming packets per interval p_i are Poisson distributed, the inter-arrival time d_i is exponentially distributed:

$$\begin{aligned} & p_i \sim Poisson(\lambda),\ d_i = p_{i+1} - p_i \\ \Rightarrow\ & d_i \sim Exp(\lambda) \\ & \hat{d_i} = F^{-1}(u_i) = -\frac{1}{\lambda}\ln(1-u_i) \hat{=} -\frac{1}{\lambda}\ln(u_i). \end{aligned}$$

By inverse transform sampling (as described in [5] Section 23.2) we determine the empirical delays between packets $\hat{d_i}$ by sampling $u_i \sim U(0,1)$ and calculating $\hat{d_i}$. By setting the parameter λ, different randomized Poisson distributed loads can be specified.

As a third option, the playground allows for recorded network scenarios to be replayed.

4 EXPERIMENTS

Two validation and two demonstration experiments were performed for which a large summation (in a loop) with a conditional statement at each step of the iteration was used as the user code.

4.1 Prototype Implementation

The playground is implemented on the ESP32 [2], a dual core CPU, and written in C and C++, matching the system of the microcontroller. The parsing script for the recorded network scenarios (PCAPs) is written in Python and generates C++ code. The last playground requirement is fulfilled by using both cores to separate the computational load of the playground code from the tested user code.

[2]ESP32 Series Datasheet Version 3.4, https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf, accessed 2021-01-07

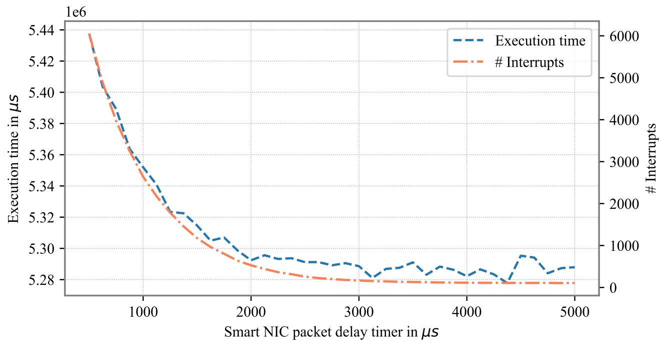

Figure 2: Execution time and number of interrupts decrease inversely with the increase of the times threshold when using the Smart NIC packet delay timer for interrupt moderation.

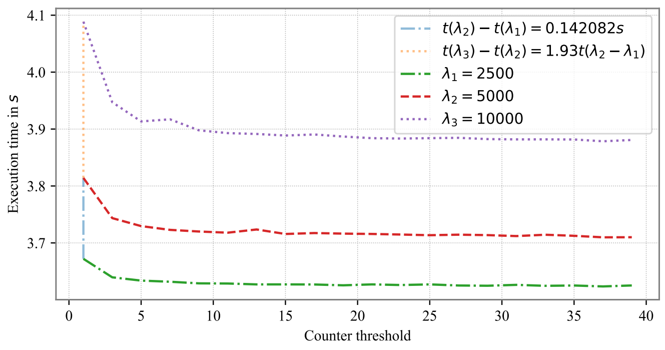

Figure 3: Comparison of the execution time of three Poisson-distributed random network load scenarios for different λ in dependence of the counter threshold.

4.2 Validation

All validation testing was performed using a Poisson-distributed random load. The following results merely show a selection of all the experiments that have been carried out with the playground.

First, the impact of the packet delay timer on the execution time and the number of interrupts was measured, as seen in Fig. 2. This test was performed using the combined interrupt moderation mode. Both, the execution time and the number of interrupts decrease with an increasing packet delay timer. As the number of packets is constant, the execution time is reduced by using interrupt moderation. This gain from increasing the packet delay timer comes with the caveat of a higher packet latency.

In the second validation experiment three different parameters λ for the Poisson-distributed random load generation were compared, as seen in Fig. 3. This test was performed using the counter mode. A larger parameter value corresponds to a higher load. In the graph, the counter threshold is plotted against the execution time. λ_1 is half as big as λ_2 and λ_3 is twice as big as λ_2. As shown in the legend, it can be observed that the difference between the execution times of λ_3 and λ_2 is twice as big as the one between λ_2 and λ_1. This

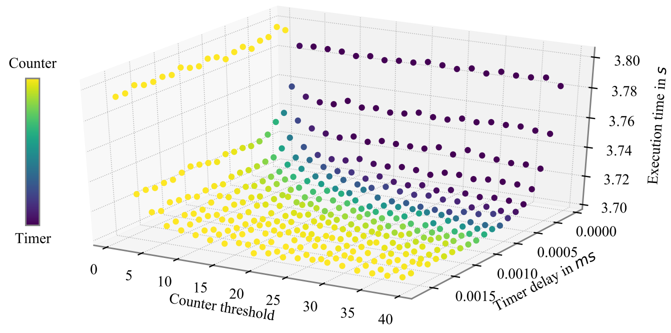

Figure 4: Ratio of the reasons for triggering interrupts, shown in relation to counter threshold, timer delay and execution time when using the smart NIC with mixed mode.

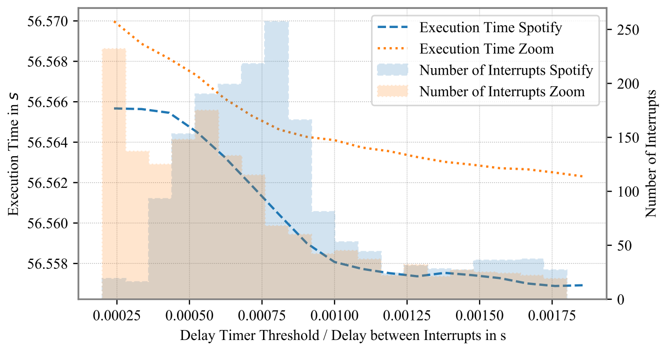

Figure 5: Execution time of the test code in two replayed network scenarios from prerecorded PCAPs is shown for diverse delay timer thresholds.

ratio stays roughly the same along the x-axis. This shows that the counter mode scales linearly with load.

4.3 Practical Examples

In the first practical example the reasons for interrupts using the combined interrupt moderation mode of the smart NIC model were investigated, as seen in Fig. 4. A Poisson-distributed random load was used for this test. The coloring of the data points indicates the reason why an interrupt was triggered. An area in the plane of the data points where the coloring indicates an equilibrium between the two reasons can be observed. The area extends in both counter threshold and timer delay directions but drifts towards the direction of the counter threshold axis, indicating that with increasing counter threshold, it plays less of a role in causing interrupts than the timer delay does.

In the second practical example we take a look at recorded loads. We compare the execution time of the user code when using a mixed mode NIC with a Spotify network load to a Zoom conference load that have been prerecorded. The load is a lot less intense compared to the previous experiments. We use these two loads because they have two different packet arrival patterns: the Spotify load is a bursty load, while the Zoom load is a rather continuous load. Note that we use a longer running user code (more iterations) here to allow for a longer measurement.

Fig. 5 is a combination of two diagrams: the lines show the execution time for different delay timer thresholds while the filled area is a histogram which shows in what intervals packets arrive in the two load scenarios. We see that the execution time in the Spotify scenario benefits more from the relaxing of the timer timeout delay while in the Zoom scenario the behavior matches the behavior of a Poisson load more closely.

When comparing the results from both scenarios it becomes more obvious that the expected load progression can be used to find suitable interrupt moderation parameters. [3]

5 CONCLUSION AND FUTURE WORK

We presented a playground which enables researchers to conduct experiments in the context of network interrupt simulation in real-time scenarios. It offers multiple load generators including random and custom prerecorded settings as well as logging capabilities. The playground was validated through a series of tests. We also presented two practical use cases, highlighting the ability of the playground to simulate desired network characteristics and analyze the results.

Further steps include a broader range of more complex NIC models and random load sources. Additionally, a repository of PCAP files could be created by playground users.

REFERENCES

[1] Ilja Behnke, Lukas Pirl, Lauritz Thamsen, Robert Danicki, Andreas Polze, and Odej Kao. 2020. Interrupting Real-Time IoT Tasks: How Bad Can It Be to Connect Your Critical Embedded System to the Internet?. In *IPCCC 2020: 39th International Performance Computing and Communications Conference*. IEEE, 1–6.

[2] Jaeil Han and Young Man Kim. 2016-10. Interval-Based Adaptive Interrupt Coalescing in High-Speed Networks. In *2016 International Conference on Information and Communication Technology Convergence (ICTC)* (Jeju). IEEE, 68–70. https://doi.org/10.1109/ICTC.2016.7763437

[3] Marcus Handte, Stefan Foell, Stephan Wagner, Gerd Kortuem, and Pedro Jose Marron. 2016-10. An Internet-of-Things Enabled Connected Navigation System for Urban Bus Riders. 3, 5 (2016-10), 735–744. https://doi.org/10.1109/JIOT.2016.2554146

[4] Gang Hou, Weiqiang Kong, Kuanjiu Zhou, Jie Wang, Xun Cao, and Akira Fukud. 2018-07. Analysis of Interrupt Behavior Based on Probabilistic Model Checking. In *2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI)* (Yonago, Japan). IEEE, 86–91. https://doi.org/10.1109/IIAI-AAI.2018.00026

[5] Kevin Patrick Murphy. 2012. *Machine Learning: A Probabilistic Perspective.* MIT Press.

[6] K. Nakashima, S. Kusakabe, H. Taniguchi, and M. Amamiya. 2002. Design and Implementation of Interrupt Packaging Mechanism. In *International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems* (Big Island, HI, USA). IEEE Comput. Soc, 95–102. https://doi.org/10.1109/IWIA.2002.1035023

[7] M. Rajaratnam and F. Takawira. 1996. Network Modelling in Circuit-Switched Networks Using the Double Interrupted Poisson Process Model. In *Proceedings of 8th Mediterranean Electrotechnical Conference on Industrial Applications in Power Systems, Computer Science and Telecommunications (MELECON 96)* (Bari, Italy), Vol. 2. IEEE, 971–975. https://doi.org/10.1109/MELCON.1996.551371

[8] Khaled Salah. 2007-04. To Coalesce or Not to Coalesce. 61, 4 (2007-04), 215–225. https://doi.org/10.1016/j.aeue.2006.04.007

[9] John A. Stankovic. 1994. Real-Time Operating Systems. In *Real Time Computing*, Wolfgang A. Halang and Alexander D. Stoyenko (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 65–82.

[10] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. 2017-03. The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. 11, 1 (2017-03), 17–27. https://doi.org/10.1109/MIE.2017.2649104

[3] code and details on https://github.com/dos-group/pieres_playground