A Priority-Aware Multiqueue NIC Design for Real-Time IoT Devices

Ilja Behnke, Philipp Wiesner, Robert Danicki, Lauritz Thamsen
Technische Universität Berlin
{i.behnke,wiesner,r.danicki,lauritz.thamsen}@tu-berlin.de

ABSTRACT

Low-level embedded systems are used to control cyber-phyiscal systems in industrial and autonomous applications. They need to meet hard real-time requirements as unanticipated controller delays on moving machines can have devastating effects. Modern developments such as the industrial Internet of Things and autonomous machines require these devices to connect to large IP networks. Since Network Interface Controllers (NICs) trigger interrupts for incoming packets, real-time embedded systems are subject to unpredictable preemptions when connected to such networks.

In this work, we propose a priority-aware NIC design to moderate network-generated interrupts by mapping IP flows to processes and based on that, consolidates their packets into different queues. These queues apply priority-dependent interrupt moderation.First experimental evaluations show that 93 % of interrupts can be saved leading to an 80 % decrease of processing delay of critical tasks in the configurations investigated.

KEYWORDS

Embedded systems, real-time operating systems, network interface controller, internet of things, cyber-physical systems

ACM Reference Format:
Ilja Behnke, Philipp Wiesner, Robert Danicki, Lauritz Thamsen. 2022. A Priority-Aware Multiqueue NIC Design, for Real-Time IoT Devices. In *The 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22), April 25–29, 2022, Virtual Event, .* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3477314.3507165

1 INTRODUCTION

In the context of cyber-physical systems, where software processes lead to physical actions, industrial processing systems and machines are controlled by microcontrollers. These devices usually provide only little processing power and run Real-Time Operating Systems (RTOSs) that implement guarantees regarding reaction times for sensory input and control commands [5]. With the advent of the Internet of Things (IoT) and Industry 4.0, many cyber-physical systems are being connected to IP networks for remote control, monitoring, and maintenance [3, 8]. While the overall architecture of these connected devices is similar to general-purpose microcontrollers, they also have to provide Network Interface Controllers (NICs) and network stack implementations.

SAC '22, April 25–29, 2022, Virtual Event,

To notify the CPU of newly available data, input devices like sensors and serial interfaces use interrupt requests that trigger Interrupt Service Routines (ISRs) on the controller. Due to the distinct priority spaces of ISRs and scheduled processes in RTOSs, interrupts preempt any currently running process independent of its priority [11]. On real-time devices this introduces a high level of unpredictability as the processing units are interrupted from outside the RTOS [9]. As a result, interrupts triggered by a NIC for incoming network packets and the load-dependent overhead of driver and networking tasks can alter the assumed time-line of running processes. This can result in breaking real-time guarantees for critical processes [2, 4].Consequentially, common IoT devices are easily overwhelmed by network packet floods, even when no subsequent processing of packets is performed.

Due to this problem, programmers have to resort either to turning off interrupts during critical executions or using separate resources for networking and processing in the typically resource-constrained environments of microcontrollers [10, 13]. However, a different solution has to be found in scenarios where IP networks are used for the control of cyber-physical systems and are necessary for critical machine functions.

In this paper, we present the design and preliminary evaluation of a priority-aware multiqueue NIC design that can reduce network-induced interrupt overloads of IoT devices without delaying packets for time-critical tasks. This is achieved by reducing the number of interrupts generated by packets related to low priority tasks. The acceptable rate of interrupts can be configured through the operating system to a per-process window.

2 MULTIQUEUE NIC DESIGN

To address the real-time violating effects of network-generated interrupts and their processing overhead, we propose a priority-aware network interface controller that handles network packets depending on their destination process: A configurable multi-queue NIC for real-time embedded systems. This section illustrates the NIC's design, configuration parameters, and operating system-side management.

2.1 Heterogeneous Interrupt Moderation

The issue of network generated interrupts impacting system performance can be addressed using interrupt moderation. However, traditional techniques have their disadvantages. While they increase the overall interrupt processing efficiency, they also increase the incurred packet delays and make them less predictable as packets are held back for a variable amount of time. Hence, our NIC design attempts at reducing the network overhead while guaranteeing low and constant latency for critical packets, i.e. packets used to control critical tasks.

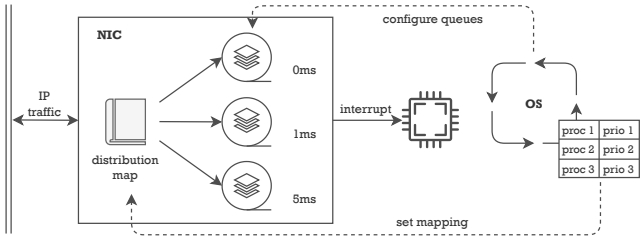

Figure 1: Multiqueue NIC: Traffic is organized into different queues with exemplary delay values attached.

As embedded systems run a fixed set of specific tasks that is seldomly altered, we can use their metadata to filter and manage incoming packets on the hardware layer before interrupting. Namely, these are the priorities of the packet-receiving processes and their associated IP flows. Interrupt moderation can thereby become the tool to enforce process priorities before entering the operating system domain.

2.2 NIC Adaptation

Our proposed NIC adaptation begins after the MAC-layer tasks. An illustration of the design can be seen in Figure 1. To accommodate incoming packets belonging to different real-time processes, the receive buffer is divided into multiple queues realized as ring buffers. This way, packet descriptors are assigned to different queues depending on their destination process and its priority.

When a data frame arrives from the network, it is validated and the packet metadata compared to a list of registered ports residing in a distribution map on the NIC. Here, packets are assigned to queues which hold packets of one process each. According to the process priority and expected packet load, different interrupt moderation configurations (e.g. delay timers and counter threshold) are applied to them via the operating system.

This way, packets for critical processes trigger interrupts immediately upon reception while less important packets (that is, packets with low priority receiving tasks) are held back before one interrupt is triggered for all packets in the respective queue, indicated by the millisecond specifications in Figure 1. Packets with no associated process can be dropped before an interrupt is triggered, preventing unnecessary processing. This is especially important with high unanticipated traffic loads targeting the device and potentially leading to a denial of service.

Relevant Parameters. The multiqueue NIC introduces four main parameters affecting packet delays and resource utilization:

- *Number of queues.* The number of queues the receive buffer is divided into is the number of processes accepting packets.
- *Size of queues.* The size of a queue corresponds to its expected packet load, available memory, and moderation parameters.
- *Absolute queue timer values.* Queue-specific periodic duration until an interrupt is triggered by the queue.
- *Packet timer values.* Queue-specific interrupt timer that is being reset reset by each incoming packet.

The timer values are used to span a time window of how long a packet remains in the queue. Depending on the packet rate, a variable number of packets is then coalesced for one interrupt. As these parameters have a high impact on the timeliness of incoming traffic and generated workload on the real-time device, the accuracy of their configuration is of high importance.

Configuration. As depicted in Figure 1, there are two interfaces necessary for NIC configurations. One for the tuning of the before mentioned queue parameters and secondly, the transfer of process-IP flow mappings. Both are performed when a socket is bound via the network stack API. To this end, the socket API is extended with driver calls performing the specific changes. Whenever a new process registers or frees a socket, the operating system transparently adjusts the number of queues and their parameters. Delay timers and queue size must be set to fit specific scenarios.

The system must be dynamically adjustable during runtime to facilitate changes in processes or IP flows. With the configuration process being linked to the socket API, all necessary tuning parameters can be passed at any point in time by the registering process.

As there is no explicit information about the receiving process in a network packet, there needs to be a mapping between packet meta data and processes. To this end, a map between IP flows and processes is created and placed on the NIC. In this design, the destination port is used to map a packet to a process and its priority.

3 EVALUATION

This section outlines a first set of experiments conducted under increasing network loads and preliminary results. The experiments have been conducted using a simulation of the presented NIC design in combination with an ESP32 IoT device running FreeRTOS.

Experiments under High Load. In all experiments the IoT device runs four worker processes of different priority. The processes are controlled over the widely used industrial communication protocol MODBUS/TCP[1]. As each of the processes binds its own socket, four queues with different interrupt moderation configurations are set up in the NIC.

One baseline experiment was performed without any interrupt moderation. To observe the system under high traffic, it is subjected to packet floods ranging from 0 to 15000 packets per second. All experiments were performed four times using different absolute delay timer values for the added packet floods. The values range from 800 μs to 3200 μs resulting in the designations *nomod* (for unmoderated traffic), *d800*, *d1600*, *d2400*, and *d3200*. Each experiment runs for a duration of 30 seconds. We observed the progression of interrupts generated in respect to packets received and the additional runtime of the processes incurred by the network traffic, and their consequential deadline misses.

Results. Figure 2 shows a comparison of packet and interrupt numbers for the baseline experiment without additional load. Queues 0 - 3 moderate interrupts in different time windows, so they generate fewer interrupts than the first queue, which is receiving packets for a critical task. The total rate of interrupts per packet ranged from

[1]MODBUS/TCP traces provided by [6].

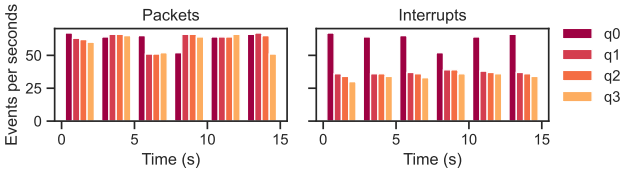

Figure 2: Packets and caused interrupts over time.

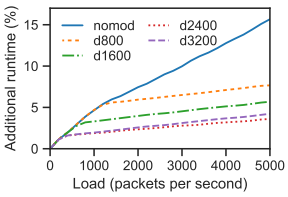

Figure 3: Additional runtime of critical process induced by increased load.

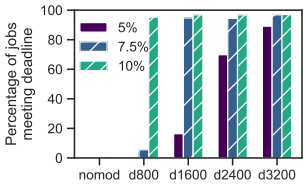

Figure 4: Share of critical packets meeting deadlines under load of 5000 pck/s.

70 % in the undisturbed experiment to 2 % with high additional load of 15000 packets per second and 3200 μs absolute timer value. The absolute moderation timer is an effective tool to moderate high loads as more packets are coalesced into interrupts while the critical task is unaffected.

Next, we observe the interrupt-induced runtime increase of the critical process. A significant mitigation of the malicious effects of packet floods can be obtained in all moderation configurations for the critical process. Processes of lower priority also benefit from the approach, as the CPU is freed up from unnecessary ISRs. Figure 3 shows the mitigating effects for the critical task under variable additional load. Further, it can be seen that there is a scenario-specific optimal configuration between *d2400* and *d3200*. By increasing the delay parameters, more packets are coalesced for each interrupt, meaning that the networking tasks are confronted with larger bursts of packets per notification. This has negative effects on CPU load starting at a critical packet count. For the maximum depicted packet load of 5000 packets per second the additional runtime could be decreased by 80 % resulting from the prevention of 93 % of interrupts.

Figure 4 shows the percentage of tasks for the critical process meeting their deadline with grace periods from 5 - 10 % compared to their baseline (median runtime without packet flood) under an additional load of 5000 per second.

4 RELATED WORK

The introduction of unpredictability in real-time environments through interrupts has been a long-standing research topic. In the following, we present past approaches to mitigate interrupt impact.

The Advanced Interrupt Controller [7] monitors the priority of the currently running process to determine if an interrupt should be triggered or held back by comparing it to the interrupt's priority. A simple extension of the interrupt controller unifies the priority spaces of attached interrupts and operating system processes.

Network Interface Controllers with multiple transmit and receive queues have been introduced by Intel as early as 2007. The goal is to make use of multicore systems by parallelizing network load on the different queues. The trend is to increase the number of queues to facilitate cloud computing as Zhu et. al. showed in 2020 [14].

The priority inverting impact of interrupts in real-time systems has been identified and tackled by Amiri et. al. by employing priority inheritance protocols for interrupt service threads [1]. In contrast, Multi-Sloth [12] presents an OS adaptation that treats all threads as interrupts, scheduling threads and ISRs in a unified priority space.

The issue of DoS attacks in industrial IoT environments has been addressed by Niedermaier et al. [13]. A dual microcontroller architecture is proposed to separate networking tasks from critical real-time processes.

5 CONCLUSION

Unexpected floods of network traffic can delay the process flow in real-time systems, which is a potential safety issue for many Industry 4.0 applications. To mitigate the effect of high traffic on real-time IoT devices, we propose a priority-aware multiqueue NIC that maps IP flows to processes when a socket is bound. We evaluated our design using a NIC simulation and an IoT device running a real-time operating system. The results of these experiments show that our approach significantly reduces the impact of traffic floods on critical process runtimes, saving 93 % of interrupts and 80 % of processing delay under packet rates of 5000 per second.

REFERENCES

[1] Javad Ebrahimian Amiri and Mehdi Kargahi. 2015. A predictable interrupt management policy for real-time operating systems. In *RTEST 2015.*
[2] 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.*
[3] Andreas Burg, Anupam Chattopadhyay, and Kwok-Yan Lam. 2017. Wireless communication and security issues for cyber–physical systems and the Internet-of-Things. *Proc. IEEE* 106, 1 (2017).
[4] Robert Danicki, Martin Haug, Ilja Behnke, Laurenz Mädje, and Lauritz Thamsen. 2021. Detecting and Mitigating Network Packet Overloads on Real-Time Devices in IoT Systems. In *EdgeSys 2021.*
[5] Norman Finn. 2018. Introduction to time-sensitive networking. *IEEE Communications Standards Magazine* 2, 2 (2018).
[6] Ivo Frazão, Pedro Henriques Abreu, Tiago Cruz, Hélder Araújo, and Paulo Simões. 2018. Denial of service attacks: Detecting the frailties of machine learning algorithms in the classification process. In *CRITIS 2018.*
[7] Tiago Gomes, Paulo Garcia, Filipe Salgado, João Monteiro, Mongkol Ekpanyapong, and Adriano Tavares. 2015. Task-aware interrupt controller: Priority space unification in real-time systems. *IEEE Embedded Systems Letters* 7, 1 (2015).
[8] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. 2014. Industry 4.0. *Business & information systems engineering* 6, 4 (2014).
[9] Luis E. Leyva-del Foyo, Pedro Mejia-Alvarez, and Dionisio de Niz. 2012. Integrated Task and Interrupt Management for Real-Time Systems. *ACM Transactions on Embedded Computing Systems* 11, 2 (2012).
[10] Anna Maria Mandalari, Daniel J Dubois, Roman Kolcun, Muhammad Talha Paracha, Hamed Haddadi, and David Choffnes. 2021. Blocking without Breaking: Identification and Mitigation of Non-Essential IoT Traffic. In *Proceedings on Privacy Enhancing Technologies.*
[11] Pedro Mejia-Alvarez, Luis Eduardo Leyva-del Foyo, and Arnoldo Diaz-Ramirez. 2018. *Interrupt Handling Architectures.*
[12] Rainer Müller, Daniel Danner, Wolfgang Schröder Preikschat, and Daniel Lohmann. 2014. Multi Sloth: An Efficient Multi-core RTOS Using Hardware-Based Scheduling. In *26th Euromicro Conference on Real-Time Systems.*
[13] M. Niedermaier, D. Merli, and G. Sigl. 2019. A Secure Dual-MCU Architecture for Robust Communication of IIoT Devices. In *MECO 2019.*
[14] Heqing Zhu. 2020. NIC-Based Parallellism. In *Data Plane Development Kit (DPDK): A Software Optimization Guide to the User Space-Based Network Applications.*