An Experience Report on Exploring CPU Usage Spike Monitoring in Linux Electronic Control Units Using Extended Berkeley Packet Filter

Larissa Pestana
Stellantis
Recife, Brazil
larissapestana13@gmail.com

Breno Miranda
Federal University of Pernambuco
Recife, Brazil
bafm@cin.ufpe.br

Abstract

The increasing consolidation of functionalities into high-performance Electronic Control Units (ECUs) has increased the variability of computational workloads in modern automotive systems, especially in architectures based on the AUTOSAR Adaptive (AUTomotive Open System ARchitecture Adaptive) platform. In critical scenarios, transient peaks in Central Processing Unit (CPU) usage may prevent the *watchdog* from being serviced within the expected time interval, leading to ECU resets and temporary loss of availability of embedded functions. This work proposes a proactive CPU usage spike monitoring system based on *extended Berkeley Packet Filter* (eBPF), through *bpftrace*, targeting Linux-based ECUs. The approach collects per-process execution samples at runtime, enabling the identification of high-load situations without modifying the kernel, scheduler, or monitored applications. The feasibility of the proposal was evaluated through a proof of concept on Arm Virtual Hardware (AVH), using Ubuntu 24.04 on a 64-bit Arm architecture, considering three workload levels representative of a consolidated ECU. The results indicate that the monitor distinguishes normal operating regimes from sustained peak-load situations, with low observed overhead. The approach shows potential as an observability mechanism to support the early detection of overload conditions associated with watchdog-triggered resets in Linux-based automotive platforms.

Keywords

Automotive Embedded Systems, Linux-based ECUs, CPU Usage Spikes, eBPF, AUTOSAR Adaptive, Watchdog Monitoring

1 Introduction

The intensive use of computational resources in Electronic Control Units (ECUs) is not a new problem in the automotive domain. In traditional architectures based on AUTOSAR Classic (AUTomotive Open System ARchitecture Classic), scenarios have already been observed in which the load of the Central Processing Unit (CPU) varies according to system behavior. In powertrain applications, for example, industrial patents document a direct relationship between the increase in engine speed (Revolutions Per Minute, RPM) and the growth of the ECU computational load [4, 5]. This behavior shows that, even in more static architectures, CPU load may affect the execution of critical tasks.

With the increasing complexity of embedded functionalities in modern vehicles, such as autonomous driving, Advanced Driver Assistance Systems (ADAS), and connectivity, this problem becomes more significant. The consolidation of multiple functions into high-performance ECUs, supported by platforms such as AUTOSAR Adaptive, general-purpose operating systems, and multicore processors, has become a central characteristic of Software-Defined Vehicles (SDVs) [6]. In this context, load variability is no longer associated only with physical events, but also emerges from concurrent service execution, function consolidation, and communication bursts.

In critical scenarios, transient CPU spikes may prevent the *watchdog*, a supervision mechanism periodically reset by the software to indicate correct system execution, from being serviced within the expected interval, leading to an ECU reset [8, 11]. Although the *watchdog* is a recovery mechanism, it does not prevent the conditions that lead to the failure. A reset during vehicle operation represents an immediate loss of availability of embedded functions and may simultaneously affect multiple critical services in consolidated ECUs. Therefore, monitoring only the post-reset state is insufficient; it is necessary to observe, at runtime, the signals that precede this behavior.

In industrial practice, the investigation of *watchdog* resets is often hindered by the loss of the execution context preceding the failure. Consequently, integration, verification, and diagnostic engineers frequently depend on incomplete logs, trace captures, or costly laboratory reproductions to perform root cause analysis. In consolidated Electronic Control Units, where multiple services share the same computational resources, this task becomes significantly more challenging.

Recent studies have associated the *extended Berkeley Packet Filter* (eBPF) with efficient observability applications in embedded systems, including automotive platforms [7, 12]. As a continuation of this line of work, this paper proposes the use of eBPF, through *bpftrace*, for proactive monitoring of CPU spikes in Linux ECUs, with a focus on AUTOSAR Adaptive architectures. The proposal was evaluated through a proof of concept on Arm Virtual Hardware (AVH), considering different computational load levels, and the results indicate that the approach distinguishes normal operation from spike situations with good temporal resolution and low overhead.

The main contributions of this work are: (i) the characterization of the problem of resets associated with transient CPU spikes in consolidated ECUs; (ii) a proactive monitoring architecture based on eBPF; (iii) a proof of concept with quantitative evaluation; and (iv) a discussion on the applicability of the approach to AUTOSAR Adaptive and the challenges of extending it to AUTOSAR Classic.

The remainder of this paper is organized as follows. Section 2 presents the technical background. Section 3 discusses the related

work. Section 4 describes the experimental methodology. Section 5 presents the results, discusses limitations and future work.

2 Technical Background

2.1 AUTOSAR Classic and AUTOSAR Adaptive

AUTOSAR Classic was designed for Electronic Control Units (ECUs) with predominantly static configurations, targeting embedded functions with deterministic requirements and strong hardware integration. In this model, software components, tasks, signals, and communication mechanisms are mostly defined at design time, favoring predictability and control over execution.

AUTOSAR Adaptive, in turn, was conceived for more dynamic and computationally intensive automotive applications. These platforms run on general-purpose operating systems, such as Linux or QNX, and adopt a service-oriented communication model. This architecture is suitable for functionalities such as Advanced Driver Assistance Systems (ADAS), connectivity, and applications associated with Software-Defined Vehicles (SDVs), but it also introduces greater concurrency among processes, shared resource usage, and variability in temporal behavior [6].

2.2 Watchdog and Relationship with CPU Load

The *watchdog* acts as a supervision timer that must be periodically reset by the software. When this reset does not occur within the defined interval, the system interprets that an execution failure has occurred and forces the reset of the Electronic Control Unit (ECU). In the context of this work, the relevant point is that this *timeout* may occur not only due to permanent hangs, but also due to transient overload situations, in which the task or context responsible for servicing the *watchdog* does not receive enough execution time.

Industrial platforms, such as OpenECU, indicate CPU load analysis as part of the investigation of *watchdog*-related resets, while industrial studies point out limitations in the quantitative evaluation of the conditions that lead to its activation [8, 11]. This relationship motivates the use of observability mechanisms capable of recording overload signals before the reset occurs.

2.3 Observability with *Extended Berkeley Packet Filter*

The *extended Berkeley Packet Filter* (eBPF) is a Linux kernel technology that allows verified programs to run in kernel space without modifying the source code of the operating system or applications. Through kernel instrumentation points, such as scheduling events and system calls, eBPF programs can collect execution and performance metrics with low overhead.

bpftrace is a high-level tool for creating scripts based on eBPF, enabling dynamic instrumentation at runtime. In the automotive domain, recent studies have explored eBPF and *bpftrace* for latency measurement on 64-bit Arm platforms and for optimizing the processing of *Scalable service-Oriented MiddlewarE over IP Service Discovery* (SOME/IP-SD) packets [7, 12]. These results indicate that the technology is suitable for observability in Linux-based automotive platforms.

3 Related Work

Related work associated with this study can be organized into two main lines. The first includes approaches focused on load management and temporal predictability in multicore automotive systems. Techniques such as adaptive scheduling, static task allocation, *runnable* optimization, and isolation policies among applications have been proposed to reduce interference, balance load, and improve deadline compliance [1, 3, 9, 10]. Although these approaches are relevant for the design and sizing of automotive systems, they usually operate at the scheduler level, depend on design-time decisions, or require modifications to the system configuration. Therefore, they do not directly address the continuous runtime observation of transient CPU spikes in already integrated Electronic Control Units.

The second line includes studies that explore eBPF in Linux-based automotive platforms. Thomeczek et al. use *bpftrace* for latency measurement on 64-bit Arm platforms, while Lim et al. apply eBPF and *eXpress Data Path* (XDP) to the processing of *Scalable service-Oriented MiddlewarE over IP Service Discovery* (SOME/IP-SD) packets [7, 12]. These studies demonstrate the feasibility of using eBPF in automotive contexts, mainly for latency characterization and communication optimization. Building on this foundation, this work investigates the use of eBPF for proactive per-task load monitoring, aiming to identify transient overload conditions that may precede resets in consolidated Linux Electronic Control Units.

4 Experimental Methodology: Proof of Concept

This section describes the proof of concept developed to evaluate the feasibility of the proposed approach. The evaluation considers the experimental environment, the generation of representative workloads for a consolidated Electronic Control Unit, the eBPF-based monitoring mechanism, and the data collection and analysis procedure.

4.1 Experimental Environment

The experiments were conducted on Arm Virtual Hardware (AVH) [2], a cloud-based 64-bit Arm emulation platform. AVH was selected because it enables the execution of a complete Linux system on an Arm architecture, approximating prototyping environments used in embedded platforms, such as Raspberry Pi boards employed in AUTOSAR Adaptive experiments. This configuration allows the approach to be evaluated in a controlled and reproducible environment compatible with the Linux-based monitoring proposal, without requiring physical automotive hardware.

The instance used an Arm Neoverse-V2 processor with 2 cores, 4 GB of RAM, and Ubuntu 24.04 LTS with kernel 6.8.0-71. Monitoring was performed with *bpftrace* 0.20.2, and auxiliary workload generation used *stress-ng* when necessary. Ubuntu 24.04 was adopted because it provides eBPF support and offers a Linux environment on Arm architecture suitable for an initial evaluation of the proposed approach.

4.2 Representative Workloads for a Consolidated Electronic Control Unit

To represent workload scenarios compatible with a consolidated Electronic Control Unit, a Python script named `outburst_sim.py`

was developed. The script comprises three computational functions inspired by automotive applications: camera processing, sensor fusion, and *Vehicle-to-Everything* (V2X) communication. The camera function uses matrix multiplication as a CPU-intensive workload; the fusion function performs floating-point operations; and the V2X communication function performs data manipulation through bit counting.

The functions were organized into three operating modes, each lasting 10 seconds. The first mode represents a light workload, with only the camera function active. The second mode represents an intermediate workload, combining camera processing and sensor fusion. The third mode represents a heavy workload, with camera processing, sensor fusion, and V2X communication running simultaneously. This last scenario approximates a transient spike situation, in which the concurrent activation of multiple functions may significantly increase CPU utilization, characterizing a condition close to the *point outburst situations* described in industrial performance analysis tools [13].

These workloads are not intended to fully reproduce the complexity of a real automotive Electronic Control Unit. Instead, they create a controlled progression of CPU usage. Therefore, the proof of concept allows us to evaluate whether the proposed monitor can distinguish different load regimes and identify execution windows associated with transient spikes.

4.3 Proposed Monitoring

Monitoring was implemented with *bpftrace*, using periodic kernel sampling events. The `profile:hz:10` event was used to collect execution samples at a frequency of 10 Hz, corresponding to one sample every 100 ms. Every second, the `interval:s:1` event prints the sample count per process and clears the counter map.

```
sudo bpftrace -e 'profile:hz:10 { @[comm] = count(); }
                  interval:s:1 { print(@); clear(@); }'
```

The collected metric corresponds to the number of times each process was observed executing within a 1-second window. The process of interest is `python3`, which runs the `outburst_sim.py` script. Since the sampling frequency is 10 Hz, the per-second count ranges, in principle, from 0 to 10 samples. Higher values indicate that the process was observed executing in a larger fraction of the samples, serving as an approximation of its relative CPU occupancy during the monitored window.

This strategy was adopted because it is simple, non-intrusive, and compatible with the objective of the proof of concept. The monitor does not require modifications to the application code, scheduler, or kernel, acting only as a runtime observability mechanism.

4.4 Experimental Procedure and Metrics

The procedure consisted of starting the *bpftrace*-based monitor, sequentially executing the three modes of the `outburst_sim.py` script, and recording the output in a log file. Each mode was executed for 10 seconds, with a 1-second interval between modes. The counts associated with the `python3` process were then extracted to compare the light, intermediate, and heavy workload scenarios.

The main metric analyzed was the number of samples of the `python3` process per second. Since the monitor operates at 10 Hz, the alert criterion was defined as more than 7 samples per second, equivalent to more than 70% of the samples in the window. This threshold was empirically adopted to distinguish high-load situations in the proof of concept. The overhead was observed during execution and remained below 2% of CPU usage, which is consistent with previous studies on the use of *bpftrace* on 64-bit Arm platforms [12].

Table 1: Sample counts of the `python3` process by workload mode

Mode	Functions	Samples/s	Spike
1	Camera	2–3	No
2	Camera + fusion	1–8;	Occasional
3	Camera + fusion + V2X	9–10	Yes

5 Results and Future Work

This section presents the results of the proof of concept described in Section 4, considering the sample counts of the `python3` process and the overhead observed during execution.

5.1 Workload Behavior and Spike Detection

Table 1 summarizes the per-second sample counts of the `python3` process for the three evaluated workload modes. The values were extracted from the log generated by *bpftrace* after the sequential execution of the light, intermediate, and heavy modes.

In Mode 1, the `python3` process was observed in 2 or 3 of the 10 possible samples per second, indicating low relative CPU occupancy. In Mode 2, the count ranged from 1 to 8, with sporadic peaks above the alert threshold, but without being sustained throughout the entire interval. In Mode 3, the process was observed in 9 or 10 samples per second during almost the entire execution, consistently exceeding the defined threshold and characterizing a sustained spike condition.

The matrix sizes used in the camera function were 50×50, 100×100, and 150×150 in Modes 1, 2, and 3, respectively. The sensor fusion and V2X communication functions had their number of iterations adjusted according to the mode intensity, with a `sleep` inversely proportional to the intensity to modulate the workload.

5.2 Monitor Overhead

The additional CPU usage associated with *bpftrace* was observed using the `top` tool and remained below 2%. This value is consistent with previous studies on the use of *bpftrace* on 64-bit Arm platforms [12] and indicates low monitoring impact on the evaluated environment. Thus, the proof of concept demonstrated that the eBPF-based monitor distinguished the evaluated workload regimes and consistently identified the sustained spike scenario.

5.3 Industrial Applicability

From an industrial perspective, the information collected by the monitor can support both post-failure diagnosis and preventive mechanisms. Before a reset occurs, sustained processor usage spikes may be used to trigger diagnostic event logging or notify platform management services. After a *watchdog* reset, the collected data may

assist in identifying tasks or services associated with the overload condition and support root cause analysis.

In practice, such information may support software integration, validation, and diagnostic engineers during the investigation of intermittent *watchdog* resets, particularly in consolidated electronic control units where multiple services share the same computational resources. In an industrial deployment, the monitoring mechanism could be integrated with state management services, onboard diagnostics, or graceful degradation strategies, enabling the platform to react to overload conditions before a reset occurs.

5.4 Future Work

As next steps, we intend to validate the approach on real automotive hardware, using workloads closer to AUTOSAR Adaptive applications, service-based communication, and multiple processes monitored simultaneously. This validation should also consider different sampling frequencies and the calibration of the alert threshold according to the platform, task criticality, and *watchdog* supervision period. We also intend to evolve the monitoring approach to support diagnostics and proactive actions by recording information before reset, such as the active task, CPU usage, and scheduling events, or by signaling management and controlled degradation mechanisms. Extending the concept to AUTOSAR Classic remains a possible research direction, but it requires alternative strategies, since these platforms typically run on real-time operating systems and do not natively support eBPF.

In an industrial deployment, this mechanism could be integrated with state management services, onboard diagnostic frameworks, or graceful degradation strategies, enabling the platform to react to overload conditions before a *watchdog* reset occurs.

Artifact Availability

The artifacts of the proof of concept are available in the repository https://github.com/LarissaPestana/CBSOFT26, including the workload simulation, monitoring, and automation scripts, as well as the raw execution log. These materials allow the experiments to be reproduced in the AVH environment or on any Linux ARM64 platform with eBPF support.

References

[1] Waqar Ali and Heechul Yun. 2019. RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*. IEEE, 143–155. doi:10.1109/RTAS.2019.00021 Framework de gang scheduling que evita interferência entre tarefas de tempo real em multicore.

[2] Arm Ltd. [n. d.]. Arm Virtual Hardware (AVH). Platforma de emulação ARM64 na nuvem. https://www.arm.com/products/development-tools/simulation/virtual-hardware Plataforma utilizada para a prova de conceito, com emulação de hardware ARM64 e suporte a Linux.

[3] Yunhao Bai, Li Li, Zejiang Wang, Xiaorui Wang, and Junmin Wang. 2022. Performance optimization of autonomous driving control under end-to-end deadlines. *Real-Time Systems* 58, 4 (2022), 509–547. doi:10.1007/s11241-022-09379-6 Framework AutoE2E de escalonamento adaptativo de duas camadas para controle autônomo.

[4] Hyundai Motor Company. 2019. Usage amount monitoring method and monitoring unit of electronic control unit for vehicle. Patente que descreve um método para registrar picos de uso da CPU e o RPM do motor.

[5] Toyota Motor Corporation. 2017. Control apparatus and control system. Patente que estabelece a relação direta entre o aumento da rotação do motor (RPM) e o crescimento da carga computacional da ECU.

[6] ETAS. 2024. Efficiency via AUTOSAR Multicore Distribution. Whitepaper. https://www.etas.com/ww/en/about-etas/newsroom/overview/maximizing-efficiency-in-modern-vehicles-unlocking-the-power-of-autosar-multicore-distribution/ Documento da ETAS sobre consolidação de funções e otimização de carga em ECUs multicore.

[7] Seong Hyeon Lim, Sung Bihon Oh, Syng Keun Cha, Young Soo Do, Se Jeong Lim, Hyeok Jun Choi, and Jae Wook Jeon. 2025. An eBPF/XDP-Based Architecture for Efficient SOME/IP Service Discovery. In *Proc. 11th Int. Conf. on Mechatronics and Robotics Engineering (ICMRE)*. IEEE, 1–6. doi:10.1109/ICMRE64970.2025.10976318 Aplicação de eBPF/XDP para processamento eficiente de SOME/IP-SD no kernel, reduzindo contexto e carga de CPU.

[8] OpenECU Support. [n. d.]. ECU Reset. Documentação online. https://support.openecu.com/tips/ecu_reset.html Documentação da plataforma OpenECU listando as causas de reset, incluindo watchdog_reset por alta carga de CPU.

[9] Sneha Paranjape and Anju S. Pillai. 2017. Optimal workload allocation for performance evaluation on multi-core automotive ECUs. In *Proc. Int. Conf. on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*. IEEE, 684–689. doi:10.1109/ICICICT1.2017.8342781 Algoritmo de alocação estática de tarefas com balanceamento de carga e task splitting.

[10] Salah Eddine Saidi, Sylvain Cotard, Khaled Chaaban, and Kevin Marteil. 2015. An ILP approach for mapping AUTOSAR runnables on multi-core architectures. In *Proc. Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO)*. ACM, 6:1–6:8. doi:10.1145/2694333.2693439 Formulação ILP para mapeamento de runnables AUTOSAR em multicore, minimizando comunicação inter-núcleo e balanceando carga.

[11] Charles Shelton and Christopher Martin. 2007. Using Models to Improve the Availability of Automotive Software Architectures. In *Proc. IEEE Int. Conf. on Software Architecture (ICSA)*. 1–6. doi:10.1109/ICSA.2007.5 Modelo da Bosch para avaliação de disponibilidade de watchdog em ECUs.

[12] Ludwig Thomeczek, Andreas Attenberger, Johannes Kolb, Vaclav Matousek, and Jurgen Mottok. 2019. Measuring Safety Critical Latency Sources using Linux Kernel eBPF Tracing. In *Proc. ARCS Workshop*. VDE, 71–78. Demonstra o uso de bpftrace para medir latências em plataforma automotiva ARM64, com overhead de 10-23 µs.

[13] Vector Informatik. [n. d.]. TA Tool Suite: Timing and performance analysis in AUTOSAR Adaptive systems. Whitepaper. https://www.vector.com/br/pt/produtos/products-a-z/software/ta-tool-suite/autosar-adaptive/ Documento da Vector que documenta "point outburst situations" (picos transitórios de CPU em AUTOSAR Adaptive).