# Enhancing the Robustness of Linux Kernel Modules that Rely on Netlink-based Interfaces

José Flauzino
*Department of Informatics*
*Federal University of Paraná*
Curitiba, Brazil
jwvflauzino@inf.ufpr.br

***Abstract*—Linux is a mature and widely adopted operating system. The robustness of the Linux kernel has been addressed in several works in the literature. Typically, those works focus on system calls, which represent the main interface between user-space applications and the kernel. However, Linux also provides an interface between the user space and the kernel space based on message exchanges via Netlink sockets. This work proposes the development of a comprehensive methodology for addressing the robustness of Linux kernel modules that rely on Netlink-based interfaces. The approach encompasses multiple stages, including the development of a fault injection technique tailored for Netlink interfaces and the systematic robustness assessment of well-known kernel modules. The research plan also encompasses the investigation of the side effects of any failures revealed and the corresponding mitigation strategies.**

***Index Terms*—robustness testing, dependability, networking**

## I. Introduction

Linux is an operating system widely adopted around the globe and is relevant in a variety of contexts. This widespread adoption is partly due to the high performance and flexibility of its monolithic kernel with module support [1]. The Linux kernel is present in a variety of contexts, from mobile devices [2] to the world's most powerful supercomputers [3].

Despite Linux being considered a mature software system, developed over three decades, the vast extent of its source code (currently exceeding 28.8 million lines of code) makes it significantly challenging to ensure the absence of bugs, which may lead to robustness issues. Several works have already assessed the robustness of Linux [4]–[7]. Those works generally propose robustness assessment methods based on the execution of specific tests on the main interface between user applications and the system kernel, known as system calls.

Although system calls are the primary means of interaction between user space and kernel space, Linux also has a secondary (but fundamental) interface for this type of interaction: Netlink sockets [8]. More specifically, Netlink sockets allow processes (i.e., running programs) in user space to communicate with Linux kernel modules that rely on this interface. Therefore, since modules are linked to the Linux kernel at runtime [9], it can be said that Netlink enables interaction with the kernel itself.

While system calls trigger an interrupt (or trap) instruction to execute a specific kernel routine with a high privilege level, Netlink communication is carried out through the exchange of messages. Currently, several Linux kernel modules employ a Netlink interface, including notable subsystems such as Netfilter (iptables), the module implementing the IEEE 802.11 (Wi-Fi) standard, and the devlink tool, a framework for managing and configuring network devices.

Besides the relevance of such subsystems, as far as is known, no prior work has proposed a methodology specifically dedicated to systematically evaluating the robustness of Linux kernel modules over Netlink. The present doctoral research aims to fill this gap by establishing a comprehensive methodology comprising methods and techniques designed to evaluate and enhance the robustness of Linux kernel modules that rely on Netlink-based interfaces.

The proposed approach comprises multiple stages, including the development of a fault injection technique specifically designed for Netlink-based communication and the systematic robustness evaluation of widely used kernel modules. Additionally, the research plan involves analyzing the potential impact of the identified failures and formulating effective mitigation strategies to address them. The ultimate goal is to uncover unknown failures of kernel modules, as well as propose technical solutions to them, contributing to an even more robust Linux kernel.

## II. Related Work

The robustness of operating systems, and Linux in particular, has been extensively explored in the literature. Ballista [10], a portable method based on fault injection on system calls, was proposed to assess the robustness of POSIX (Portable Operating System Interface) systems. This method was used to evaluate the main POSIX operating systems of the 1990s, including FreeBSD, SunOS, and Linux [4].

In [5], a state-aware approach was proposed to assess the robustness of operating systems. The solution, called SABRINE, automatically extracts state models from system call execution traces to generate a set of test cases that cover different states of the operating system. The approach was demonstrated by evaluating the robustness of FIN.X-RTOS, a Linux-based real-time operating system used in the aviation domain.

Existing literature has also investigated the robustness of Linux device drivers (which can be built directly into the kernel core or as loadable modules). In [11], a fault injection technique targeting the Driver Programming Interface (DPI)

was proposed to characterize how faulty drivers affect the kernel. Another approach was proposed in [12], in which the system behavior was evaluated in the presence of faulty drivers through machine-code-level fault emulation. A third work in this context proposed Devil [13], an Interface Definition Language that enhances driver robustness by generating code with built-in checks.

The Linux Test Project (LTP) [6] is a collaborative initiative maintained by organizations such as Red Hat, SUSE, IBM, Cisco, Oracle, and Fujitsu. Its primary goal is to support and facilitate testing efforts aimed at validating the reliability, robustness, and stability of the Linux kernel. In practice, LTP offers a comprehensive suite of tools and test cases designed to evaluate the kernel and associated components.

Fuzzing is another widely used approach to assess the robustness of the Linux kernel. This encompasses Google's Syzkaller tool [14] as well as several enhancements and tools developed from it [15]–[18].

While all these works assess the robustness of Linux with an emphasis on system calls, efforts focused specifically on the robustness assessment of Linux kernel modules via Netlink interfaces remain scarce. This work aims to fill this gap by proposing a methodology not only for assessing but also for enhancing the robustness of the Linux kernel in the context of Netlink-based communication.

## III. Linux Kernel Modules and Netlink Sockets

This section introduces essential concepts relevant to the work. It begins with an introduction to Linux kernel modules in Subsection III-A, followed by a brief overview of Netlink sockets in Subsection III-B.

### A. A Brief Introduction to Linux Kernel Modules

Linux has a monolithic kernel but supports modules thus inheriting the many advantages of microkernels (such as flexibility) without sacrificing performance [1]. A kernel module is a binary object that extends kernel functionality and can be dynamically loaded or unloaded at runtime [9]. Modules run in kernel mode on behalf of user processes, similar to statically linked kernel functions. Common types include device drivers, file systems, and networking components.

### B. An Overview of Netlink Sockets

A socket serves as an abstraction for inter-process communication, enabling data exchange between processes that may reside either on the same host or across different machines. Each socket is characterized by three key attributes: (*i*) the Address Family (AF); (*ii*) the socket type; and (*iii*) the Protocol Family (PF). Although there is generally a one-to-one correspondence between AFs and PFs, some AFs can be associated with multiple PFs.

Netlink [8], [19], represented by `AF_NETLINK`, is a Linux address family that enables bidirectional communication between user space and kernel modules. In the latest stable version of the Linux kernel (6.15.8), there are 23 protocol families (PFs) registered for Netlink, including `NETLINK_NETFILTER` (used by `iptables`) and `NETLINK_ROUTE` (for managing network routes, IP addresses, link parameters, queueing disciplines, and more). Linux also provides a Generic Netlink family (`NETLINK_GENERIC`), a flexible interface for defining custom communication protocols.

In the Linux kernel, modules can register one or more Generic Netlink families, each representing a particular functionality. Messages sent to these families are automatically routed to the correct module without requiring a destination address. Each family specifies a set of commands and the attributes needed to execute them.

## IV. A Methodology for Enhancing the Robustness of Linux Kernel Modules with Netlink Interfaces

The purpose of this doctoral work is to establish a methodology that encompasses a set of methods and techniques aimed at enhancing the robustness of Linux kernel modules that rely on Netlink-based interfaces. The next subsections introduce the research proposal, describing the core research problems that motivate the study, and presenting the planned investigations.

### A. Robustness Assessment: Testing & Analysis Techniques

Robustness is defined as the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions [20]. Thus, robustness testing is essential to ensure that software components behave correctly under unexpected conditions. Therefore, it plays a crucial role in detecting and mitigating these issues before they impact users or systems in production environments.

In this sense, an essential part of this work is to develop a robustness testing technique suited to the characteristics of Netlink communication. First of all, it is necessary to consider the differences in message formats used by the Netlink and Generic Netlink families. In particular, it is important to carefully determine which header fields can be safely manipulated to evaluate robustness without compromising the validity of test results. Improper handling of these fields may result in communication failures (such as messages not delivered to the kernel module) being mistakenly interpreted as failures in the target kernel module.

A substantial challenge is the absence of detailed specifications for kernel modules. The official Linux kernel documentation for existing Netlink and Generic Netlink families [21] is often incomplete. For some families, there is a complete lack of documentation, while for the remaining, details such as the range of values accepted as valid for each attribute are not specified. This significantly makes it difficult to generate truly invalid input sets and, consequently, makes it more challenging to identify whether an output returned for a given input was actually appropriate or not (in other words, it makes it more challenging to identify failures).

To address these challenges, the investigation will focus on a fault injection technique in which input values are systematically generated from data type-driven rules – since the types of attributes are specified in the documentation or

can be identified in the source code. In particular, the rules will be defined in such a way as to explore the limits of each attribute's data types. This allows for the generation of reduced test cases (compared to other techniques, such as fuzzing, for example) that are nevertheless considerably representative. Table I presents the initial set of rules.

TABLE I
RULES FOR GENERATING INVALID VALUES FOR NETLINK ATTRIBUTES.

| | Rule Name | Description |
|---|---|---|
| String | StrNull | Set a null value |
| | StrEmpty | Sets an empty string |
| | StrNonPrintable | Sets a string with non-printable characters |
| | StrAlphanumeric | Sets an alphanumeric string |
| | StrOverflow | Sets characters that exceed the maximum length |
| Number | NumNull | Set a null value |
| | NumMinType | Sets the minimum valid number for the data type |
| | NumMaxType | Sets the maximum valid number for the data type |
| | NumUnderflow | Sets a negative number that exceeds the data type |
| | NumOverflow | Sets a positive number that exceeds the data type |
| Struct | StructNull | Sets a null value |
| | StructPrimitive | Sets a primitive value (Boolean, Int, Float, *etc.*) |
| | StructCommon | Sets a common data structure (List, Date, *etc.*) |

To overcome the challenges of identifying robustness failures (in cases where a response is returned), the initial focus will be on alternatives to the Netlink specification. Specifically, the proposal is to use the POSIX standard error specification instead of relying on the specification of accepted value limits for each attribute (more details are given in Section V-A). The final goal in this regard is to achieve an automatic fine-grained failure detection approach.

Another aspect to be investigated is the side effects of any failures revealed. In particular, efforts will concentrate on the analysis of failure propagation and vulnerability analysis.

### *B. Mitigation Strategies for Enhancing the Robustness of Kernel Modules*

This work aims not only to reveal failures in Linux kernel modules but also to propose solutions or, at the very least, mitigation strategies to address them. The main emphasis in this regard will be on improving the mechanisms for validating input data. The current approach used to validate Netlink attributes automatically is based on `nla_policy` contracts (also known as Netlink Attribute Policies). Therefore, investigations will focus on improving and proposing new policies, as well as alternative solutions.

## V. CURRENT STATUS AND NEXT STEPS

This section outlines the current progress of the doctoral research and the activities planned for its completion. First, the current status of the work is presented, followed by a description of the investigations planned for the next stages.

### *A. Current Status*

As an initial step in the research, an approach for assessing robustness via Netlink-based interfaces was developed, focusing on a mature and widely adopted Linux kernel module: the Open vSwitch datapath implementation. Open vSwitch is an open-source multilayer network switch designed for virtualized environments. Its architecture includes a kernel module dedicated to packet processing, which has been officially integrated into the Linux kernel since version 3.3, released in March 2012.

The proposed approach [22], employs a fault injection technique that systematically generates invalid input values (from data type-driven rules) and submits them to the Open vSwitch kernel module through Netlink attributes. As illustrated in Figure 1, the robustness tests are performed using a fault injector implemented as a user-space process. This injector generates both the workload (valid inputs) and the faultload (invalid inputs), which are sent to the Open vSwitch kernel module through a Netlink socket. The module processes these inputs and returns a response – unless a failure prevents it. Failures are identified by monitoring the target kernel module and manually analyzing the responses received (or their absence).

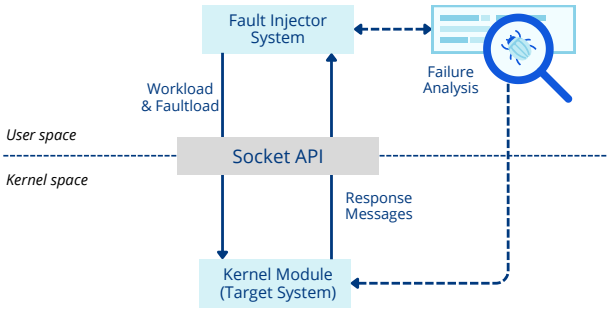


Fig. 1. An overview of the proposed approach to assess the robustness of Linux kernel modules with Netlink-based interfaces.

Due to the current limitations of the Linux kernel documentation (not providing complete specifications for all Generic Netlink families), a system error code specification-oriented analysis method was proposed to examine the returned responses. Thus, instead of relying only on the module specification, the proposed method allows the classification and interpretation of failures and inconsistencies based on POSIX-standard [23] error codes and the CRASH scale [24].

Experimental results considering four major kernel versions, spanning 10 years of Open vSwitch development, revealed a plethora of failures and inconsistencies. The identified failures include cases where incorrect error codes are returned for certain invalid input entries (*i.e.*, hindering failures – from the CRASH scale) and cases where a success code is returned when an error code should have been returned (silent failures).

Figure 2 presents a comparison of different kernel versions in terms of the failure rate (*i.e.*, the proportion of test cases that triggered failures). The observed behavior showed that, in most cases, the failure rate either increased over time or remained consistently high – often around or even exceeding 40%. This suggests that robustness has historically not been a priority in the development of the Open vSwitch kernel module. These findings also raise concerns that other Linux kernel modules may exhibit similar issues, highlighting the

relevance of this work and the importance of systematically assessing the robustness of additional modules.

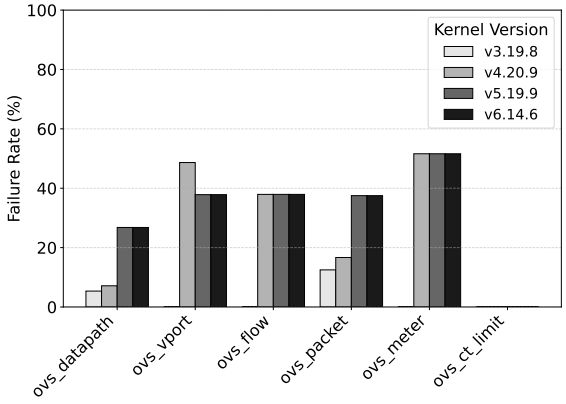


Fig. 2. Evolution of failure rates across kernel versions. The x-axis represents Generic Netlink families implemented by the Open vSwitch kernel module.

### B. Next Steps

The progress achieved so far demonstrates the potential of the proposed approach to assess the robustness of kernel modules via Netlink. However, further research efforts are required to address the challenges mentioned in Section IV.

Once the methodology is established, a representative set of Linux kernel modules will be systematically selected for robustness evaluation using the proposed approach. Insights and results obtained from these practical assessments will guide iterative refinements, ensuring the methodology is sufficiently comprehensive for application to other existing modules and future developments based on Netlink interfaces. Furthermore, any identified failures will be reported, and suggestions for improvements in the code and development practices will be presented to the community.

The final stage of the doctoral work will focus on writing and refining the dissertation. The tentative date for the thesis defense is scheduled for mid-2026.

## VI. Conclusion

This doctoral research proposal introduces a methodology for assessing and enhancing the robustness of Linux kernel modules that rely on Netlink-based interfaces. The proposed methodology to be developed is grounded in fault injection techniques and systematic test profiles to uncover failures that may affect kernel module robustness. Preliminary results from the evaluation of the Open vSwitch kernel module highlight both the feasibility and the relevance of the work. Moving forward, the research will refine the methodology, expand its application to other modules, and propose mitigation strategies aimed at contributing to an even more robust Linux kernel.

## Acknowledgments

The author thanks his advisors, professors Elias P. Duarte Jr. and Marco Vieira. This work has been supported by the Coordination for the Improvement of Higher Education Personnel (CAPES) - Program of Academic Excellence (PROEX).

## References

[1] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel: from I/O ports to process management*. "O'Reilly Media, Inc.", 2005.

[2] Statista, "Market share of mobile operating systems worldwide from 2009 to 2025," https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/, 2025, accessed in July 2025.

[3] TOP500, "June 2025," https://top500.org/lists/top500/2025/06/, 2025, accessed in July 2025.

[4] P. Koopman and J. DeVale, "Comparing the robustness of POSIX operating systems," in *Proceedings of the 29th IEEE International Symposium on Fault-Tolerant Computing (FTCS'1999)*. IEEE, 1999, pp. 30–37.

[5] D. Cotroneo *et al.*, "Sabrine: State-based robustness testing of operating systems," in *28th IEEE/ACM International Conference on Automated Software Engineering (ASE'2013)*. IEEE, 2013, pp. 125–135.

[6] LTP, "Linux Test Project," https://linux-test-project.readthedocs.io., 2025, accessed in July 2025.

[7] L. Wang, X. Wang, and D. Wu, "Correlation between complex network features and robustness in Linux kernel modules," in *Proceedings of the 2nd IEEE International Conference on Software Analysis, Testing and Evolution (SATE'2017)*. IEEE, 2017, pp. 80–89.

[8] P. Neira-Ayuso, R. M. Gasca, and L. Lefevre, "Communicating between the kernel and user-space in Linux using Netlink sockets," *Software: Practice and Experience*, vol. 40, no. 9, pp. 797–810, 2010.

[9] K. N. Billimoria, *Linux Kernel Programming: A comprehensive guide to kernel internals, writing kernel modules, and kernel synchronization*, 2nd ed. Packt Publishing Ltd, 2024.

[10] N. P. Kropp, P. J. Koopman, and D. P. Siewiorek, "Automated robustness testing of off-the-shelf software components," in *Digest of papers. twenty-eighth annual international symposium on fault-tolerant computing (cat. no. 98cb36224)*. IEEE, 1998, pp. 230–239.

[11] A. Albinet, J. Arlat, and J.-C. Fabre, "Characterization of the impact of faulty drivers on the robustness of the linux kernel," in *International Conference on Dependable Systems and Networks, 2004*. IEEE, 2004, pp. 867–876.

[12] J. Duraes and H. Madeira, "Characterization of operating systems behavior in the presence of faulty drivers through software fault emulation," in *2002 Pacific Rim International Symposium on Dependable Computing, 2002. Proceedings.* IEEE, 2002, pp. 201–209.

[13] L. Réveillère and G. Muller, "Improving driver robustness: an evaluation of the devil approach," in *2001 International Conference on Dependable Systems and Networks*. IEEE, 2001, pp. 131–140.

[14] Google, "Syzkaller - kernel fuzzer," https://github.com/google/syzkaller, 2025.

[15] Y. Lan *et al.*, "Thunderkaller: Profiling and improving the performance of syzkaller," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 1567–1578.

[16] B. Zhao *et al.*, "{StateFuzz}: System {Call-Based}{State-Aware} linux driver fuzzing," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3273–3289.

[17] D. Wang *et al.*, "{SyzVegas}: Beating kernel fuzzing odds with reinforcement learning," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2741–2758.

[18] M. Fleischer *et al.*, "{ACTOR}:{Action-Guided} kernel fuzzing," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 5003–5020.

[19] Kernel, "Introduction to Netlink – The Linux Kernel Documentation," https://docs.kernel.org/userspace-api/netlink/intro.html, 2025, accessed in May 2025.

[20] IEEE, "Systems and Software Engineering–Vocabulary," IEEE, Standard 24765:2017, 2017.

[21] Kernel, "Netlink Family Specifications," https://docs.kernel.org/networking/netlink_spec, 2025, accessed in July 2025.

[22] J. Flauzino, M. Vieira, and E. P. Duarte Jr, "Robustness Assessment of the Open vSwitch Kernel Module," in *36th IEEE International Symposium on Software Reliability Engineering (ISSRE, 2025)*, 2025.

[23] IEEE, "IEEE Standard for IEEE Information Technology - Portable Operating System Interface (POSIX)," IEEE, Standard 1003.1-2001, 2001.

[24] P. Koopman *et al.*, "Comparing operating systems using robustness benchmarks," in *Proceedings of SRDS'97: 16th IEEE Symposium on Reliable Distributed Systems*. IEEE, 1997, pp. 72–79.