2025 IEEE 36th International Symposium on Software Reliability Engineering (ISSRE) | 979-8-3503-9302-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/ISSRE66568.2025.00032

Robustness Assessment of the Open vSwitch Kernel Module

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

Marco Vieira
College of Computing and Informatics
University of North Carolina at Charlotte
Charlotte, USA
marco.vieira@charlotte.edu

Elias P. Duarte Jr.
Department of Informatics
Federal University of Paraná
Curitiba, Brazil
elias@inf.ufpr.br

***Abstract*—Open vSwitch is a software implementation of a multilayer switch designed for virtualized environments. Its architecture includes components in both user and kernel space. Although Open vSwitch is considered to be a mature project and has been widely adopted, its robustness has never been publicly assessed. While previous works focused on performance, in this work, we investigate the robustness of a fundamental component of Open vSwitch, the kernel module. The approach is based on injecting faults into the control plane interface of the Open vSwitch kernel module – which is based on Netlink sockets. We systematically tested all Generic Netlink families implemented by Open vSwitch and their respective commands and attributes across four different Linux kernel versions. Results reveal a plethora of failures and clear indications of inconsistencies in the handling of faulty inputs.**

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

I. Introduction

The past two decades have witnessed a deep shift in networking technologies, with the widespread adoption of programmable, Software-Defined Networks (SDN) [1]. Those technologies have had a deep impact on the way networks are built and operated, allowing adaptable, flexible solutions. SDN decouples the control and data planes, improving the performance of the data plane as well as enabling a centralized and programmable control plane. The advance of virtualization technologies has not only boosted SDN, but also made other paradigms viable, such as Network Functions Virtualization (NFV) [2], [3], which allows specialized hardware appliances to be replaced by software-based network functions that run on commodity hardware.

This is the context in which Open vSwitch [4] emerged, providing an open-source software implementation of a multilayer network switch for virtualized environments. Open vSwitch is designed to be a flexible, general-purpose solution that still delivers competitive performance levels. First released in 2009, Open vSwitch is now maintained by the Linux Foundation and has had over 500 contributors along the way. The maturity of the project over ≈16 years of development and the wide range of features have contributed to Open vSwitch becoming the leading virtual switch solution today, being widely employed to support virtualized network services [5] and large-scale production environments [6].

The Open vSwitch architecture includes both user space and kernel space components. While command-line tools and daemons running in user space provide functionality related to the management and operation of the switch, a kernel module is responsible for packet processing. This module has been officially integrated into the Linux kernel since March 2012.

Several efforts have been made to improve, validate, and evaluate the performance of Open vSwitch [7]–[11]. However, flexibility and performance are not the only aspects to be considered in modern computer systems. Trustworthiness in the functioning of a system (*i.e.*, dependability) is another crucial factor [12], [13]. This concerns various attributes of the system, including robustness. The term robustness can be defined as the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions [14]. Although several works have assessed the robustness of different systems over the years [15], [16], to the best of our knowledge, Open vSwitch has not yet been evaluated in this regard.

In this work, we assess the robustness of the Open vSwitch kernel module by following a targeted fault injection approach. This approach is based on systematically crafted inputs, consisting of invalid and boundary values, submitted to the Open vSwitch kernel module. The injection is performed through the communication interface between user space and the kernel module, which is established via Netlink sockets. Thus, the robustness tests are focused on the control plane of the Open vSwitch kernel module.

Since the Linux kernel documentation does not specify the valid input values for each Netlink attribute, we propose a system error code specification-oriented analysis method. This method involves comparing the output of the kernel module against the error codes defined in the POSIX (Portable Operating System Interface) standard [17], and applying an adaptation of the CRASH scale [18] to identify system failures and inconsistencies.

We report results from four different versions of the Linux kernel, ranging from version 3 (in which Open vSwitch was initially integrated) to version 6 (the latest version available).

This work has been partially supported by the Coordination for the Improvement of Higher Education Personnel (CAPES) - Program of Academic Excellence (PROEX), Funding Code 001; and the Brazilian National Council for Scientific and Technological Development (CNPq) - grant 308959/2020-5.

979-8-3503-9302-6/25/$31.00 ©2025 IEEE
DOI 10.1109/ISSRE66568.2025.00032

The selected Linux versions were chosen to represent key stages in the evolution of the Open vSwitch kernel module and the Linux networking stack. Results reveal several failures in the Open vSwitch kernel module, including a total of 82 hindering failures and 5 silent failures in the latest stable release of the Linux kernel. The findings also indicate a degradation in the robustness of the Open vSwitch kernel module over the last 10 years, as the failure rate has increased significantly in some cases. Finally, the results also demonstrate that even with the widespread dissemination of software robustness assessment techniques, which began three decades ago, mature and stable software systems still present robustness issues.

In summary, the contributions of this paper are:

- **A robustness assessment of the Open vSwitch kernel module**. To the best of our knowledge, this is the first work that addresses the robustness of an Open vSwitch component (in this case, the Linux kernel module). Furthermore, it also represents a first effort to systematically evaluate the robustness of a kernel module that relies on a Netlink-based interface.
- **A system error code specification-oriented analysis method**. We leverage POSIX-standard error codes and apply an adaptation of the CRASH scale to classify and interpret failures and inconsistencies.
- **A comprehensive experimental study**. Our experiments span four major versions of the Linux kernel (from version 3 to 6), covering a timeline of over a decade of Open vSwitch developments.
- **Evidence demonstrating that robustness issues persist even in mature, widely adopted systems**. Results highlight the ongoing relevance of robustness evaluation in modern software systems.

The paper is organized as follows. Section II presents some of the most relevant related works. Section III presents an overview of Open vSwitch, providing a brief introduction to Netlink sockets and details on the Open vSwitch's Netlink interface, the robustness of which is evaluated in the paper. Section IV presents the proposed approach for evaluating the robustness of the Open vSwitch kernel module. The results of the evaluation are presented in Section V. In Section VI, we discuss the results. Section VII presents the limitations and potential threats to the validity of the work. Finally, Section VIII concludes the paper.

II. Related Work

This section presents the most relevant research efforts that are in some aspect related to the context of this work. That includes both works that evaluate Open vSwitch and works that evaluate the robustness of other software systems.

Due to its relevance in industry and academia, Open vSwitch has been evaluated from different points of view. Most existing studies primarily concentrate on performance, including works that evaluate [11], validate [10], and improve [7]–[9] the performance of Open vSwitch. These studies often explore enhancements such as the use of DPDK (Data Plane Development Kit), GPU-based acceleration, and even alternative architectural designs.

Assessing the robustness of software systems is an important topic that has been explored in various contexts [15]. A pioneering work in this area introduced Ballista [18], a method that generates tests for individual system functions (such as system calls) by combining valid and exceptional input values. This method was used to evaluate the robustness of 13 POSIX operating systems in the 1990s. *Arlat et al.* [19] proposed the MAFALDA tool, which enables the characterization of microkernel behavior in the presence of faults. In [20], a state-aware approach named SABRINE was introduced to assess operating system robustness by extracting state models from system call traces and generating state-coverage test cases. The method was applied to evaluate the robustness of FIN.X-RTOS, a Linux-based real-time OS used in aviation.

The literature also includes works evaluating the robustness of multiple and widely different systems, such as embedded systems [21], communication systems [22], web services [23], autonomous systems [24], and adaptive systems [25], among others. In the context of SDN, research has been conducted on the evaluation of both the security and robustness evaluation of SDN controllers, particularly when faced with failures in control plane components [26], [27].

However, to the best of our knowledge, no prior work has evaluated the robustness of Open vSwitch. The present work addresses this gap by designing an approach to assess the robustness of one of its core components, the kernel module. Furthermore, as far as we know, this is the first work to systematically evaluate the robustness of a system through a Netlink interface.

III. Open vSwitch and Netlink

This section presents an overview of the Open vSwitch architecture, followed by an introduction to Netlink sockets, and the Netlink interface of the Open vSwitch kernel module.

A. Open vSwitch Architecture

The Open vSwitch architecture is shown in Figure 1. Open vSwitch runs components in both user and kernel space, and also includes or interacts with remote components. A central component is ovs-vswitchd, which is implemented as a daemon (*i.e.*, a background process) that runs in user space. Another fundamental component is a kernel module that implements the so-called datapaths. This module has officially been part of the Linux kernel since version 3.3, released in March 2012.

Network packets are received and forwarded in kernel space by the Open vSwitch kernel module. When a packet arrives at the module, a query is made on the flow rules already defined (if any). These rules establish, for example, the physical or virtual ports through which the packets of a given flow should be transmitted. Therefore, if there is a rule for that packet's flow, the kernel module forwards it accordingly. Otherwise, it forwards the packet to ovs-vswitchd in user space.

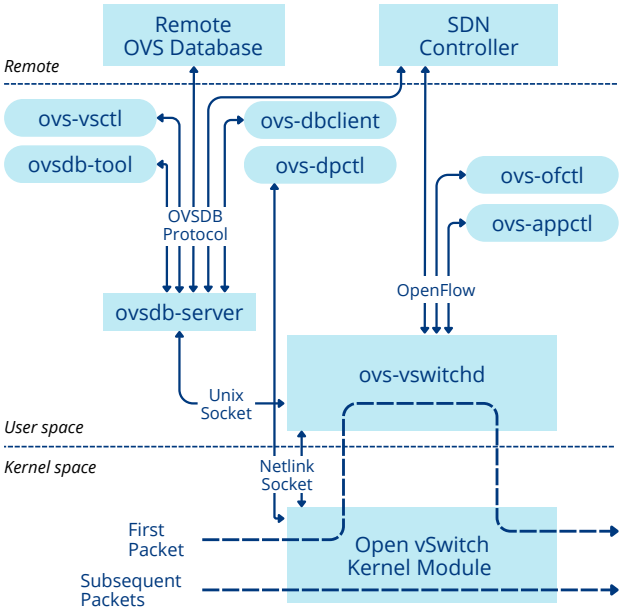

Fig. 1. The Open vSwitch architecture.

The ovs-vswitchd daemon is responsible for finding out if there is a flow rule set for that packet (this is done by interacting with ovsdb-server or the SDN Controller - both are described below). If ovs-vswitchd finds a matching rule, it returns the packet to the kernel module along with the actions to be taken to forward it appropriately. Otherwise, the packet is dropped. The actions received by the kernel module are stored in its (volatile) cache for handling future similar packets.

Figure 1 also presents the ovsdb-server (another daemon), which serves as the Open vSwitch database responsible for storing the switch's configuration. This includes, for example, the datapath settings, packet flow rules, virtual port configurations, and more. The database is managed via the OVSDB protocol, which was specifically designed for this purpose and is defined in RFC 7047 [28].

To enable the use of Open vSwitch in SDN environments, ovs-vswitchd provides an interface compatible with the OpenFlow protocol. Through this interface, SDN controllers can manage flow rules. However, OpenFlow is limited to flow control and does not handle broader operations, such as adding and removing ports on the virtual switch. To perform those kinds of operations, an SDN controller must also communicate with ovsdb-server using the OVSDB protocol. As OVSDB is formally defined in RFC 7047, most modern SDN controllers do provide support for this protocol.

Figure 1 also illustrates that the communication between ovs-vswitchd and the Open vSwitch kernel module occurs through Netlink sockets. The concept of Netlink sockets, as well as the Netlink interface implemented by the Open vSwitch kernel module, are described in the following subsections.

B. Netlink Sockets

A socket is an interface that abstracts inter-process communication. Processes that communicate through a socket can reside either on the same machine or on different machines, depending on the characteristics of the socket being used. A socket has three essential attributes: *i*) an Address Family (AF), or simply the family of the socket (*e.g.*, `AF_INET`, `AF_INET6`, and `AF_UNIX`); *ii*) a type (*e.g.* , `SOCK_STREAM`, `SOCK_DGRAM`, and `SOCK_RAW`), and *iii*) a Protocol Family (PF), such as `PF_INET`, `PF_INET6`, `PF_UNIX`, among others. Typically, each Address Family corresponds to a single Protocol Family, but in some cases, an AF may be associated with multiple PFs.

Netlink [29], [30] is an AF implemented in Linux, represented by `AF_NETLINK`. Using Netlink sockets, user-space processes can communicate with kernel modules that support this interface. That communication is bidirectional, in the sense that not only can user-space processes invoke kernel modules, but there are also upcalls through which the kernel can send notifications to user-space processes. In the current stable version of the Linux kernel (6.14.6), there are 23 PFs registered for `AF_NETLINK`. Notable examples of PFs include `NETLINK_NETFILTER`, used by the `iptables` tool to access Netfilter; `NETLINK_SELINUX`, which transmits SELinux (*Security-Enhanced Linux*) notification events, among others. Additionally, Linux also supports a Generic Netlink family, `NETLINK_GENERIC`, which allows developers to define and register custom protocol families for specialized communication needs.

A Linux kernel module can define one or more Generic Netlink families (*i.e.*, PFs), each corresponding to a specific functionality of the module. Messages sent through a socket associated with a Generic Netlink family are automatically routed to the appropriate module without requiring a destination address, unlike protocols such as IP. For each family, the module must also define a set of commands (*i.e.*, available operations) and associated attributes that must be provided when executing each command.

C. Netlink Interface of the Open vSwitch Kernel Module

The Linux kernel documentation provides specifications for the Netlink interfaces implemented by its modules. Unfortunately, those specifications do not cover all Generic Netlink families. To address this gap, we present in this subsection a didactic summary of the Generic Netlink families implemented by the Open vSwitch module, based on the available documentation and also identified by working with the Linux kernel source code.

As described in Table I, Open vSwitch defines a set of 6 Generic Netlink families. For each of these families a specific set of commands is also implemented, as detailed in Table II.

In addition, several attributes are defined and can be passed as parameters to the commands. However, the full list of existing attributes is too extensive to include in this document. As an illustration, Table III presents the first four attributes defined for the Generic Netlink family `ovs_flow`, which

TABLE I
GENERIC NETLINK FAMILIES IMPLEMENTED BY THE OPEN VSWITCH KERNEL MODULE.

Name	Description
ovs_datapath	Used to manage datapaths
ovs_vport	Used to manage the virtual ports
ovs_flow	Used to manage the flow table
ovs_packet	Used for packet-related notifications
ovs_meter	Used to manage packet flow metering
ovs_ct_limit	Used to limit entries in the connection tracking table

TABLE II
COMMANDS OF THE OPEN VSWITCH KERNEL MODULE.

	Command	ID	Description
ovs_datapath	OVS_DP_CMD_NEW	1	Adds a datapath
	OVS_DP_CMD_DEL	2	Removes a datapath
	OVS_DP_CMD_GET	3	Lists datapaths
	OVS_DP_CMD_SET	4	Modifies a datapath
ovs_vport	OVS_VPORT_CMD_NEW	1	Adds a virtual port
	OVS_VPORT_CMD_DEL	2	Removes a virtual port
	OVS_VPORT_CMD_GET	3	Lists virtual ports
	OVS_VPORT_CMD_SET	4	Modifies a virtual port
ovs_flow	OVS_FLOW_CMD_NEW	1	Add a flow rule
	OVS_FLOW_CMD_DEL	2	Removes a flow rule
	OVS_FLOW_CMD_GET	3	Lists flow rules
	OVS_FLOW_CMD_SET	4	Modifies a flow rule
ovs_packet	OVS_PACKET_CMD_MISS	1	Notifies a packet miss
	OVS_PACKET_CMD_ACTION	2	Notifies actions applied to a packet
	OVS_PACKET_CMD_EXECUTE	3	Applies actions to a packet
ovs_meter	OVS_METER_CMD_FEATURES	1	Lists supported features
	OVS_METER_CMD_SET	2	Adds or modify a meter
	OVS_METER_CMD_DEL	3	Removes a meter
	OVS_METER_CMD_GET	4	Gets meter statistics
ovs_ct_limit	OVS_CT_LIMI_CMD_SET	1	Creates/modifies a conntrack limit
	OVS_CT_LIMI_CMD_DEL	2	Removes a conntrack limit
	OVS_CT_LIMI_CMD_GET	3	Lists conntrack limits

allows managing the flow table used in packet classification. Some of these attributes are of primitive types, such as OVS_FLOW_ATTR_TCP_FLAGS, while others represent structured data with nested attributes, like attributes with IDs 1, 2, and 3 shown in the table.

Using these attributes (along with others not shown in the table), it is possible to define both the type of packet a rule should match and the actions to be performed. For example, a Layer 2 forwarding rule might specify that packets arriving on virtual port 15 with destination Ethernet address 02:42:07:5f:22:2c should be forwarded to virtual port 2. More complex rules can also be defined to perform Layer 3 routing, tunneling, and other advanced networking operations.

TABLE III
EXAMPLES OF ATTRIBUTES FOR CLASSIFYING PACKET FLOWS.

Attribute	ID	Type	Description
OVS_FLOW_ATTR_KEY	1	struct	Nested attributes specifying the flow key
OVS_FLOW_ATTR_ACTIONS	2	struct	Actions to take for packets that match the key
OVS_FLOW_ATTR_STATS	3	struct	Statistics for a given flow
OVS_FLOW_ATTR_TCP_FLAGS	4	uint8	The ORed value of all of the TCP flags seen on packets in a given flow
OVS_FLOW_ATTR_USED	5	uint64	The time (ms) at which a packet was last processed for a given flow

IV. ROBUSTNESS ASSESSMENT OF THE OPEN VSWITCH KERNEL MODULE

Our robustness assessment method relies on a fault injection technique that systematically introduces invalid and boundary inputs, tailored to the characteristics of Netlink communication. In essence, the approach involves generating both valid and invalid Netlink messages, sending them to the Open vSwitch kernel module, and monitoring its behavior to identify potential failures. This process is carried out within the scenario illustrated in Figure 2.

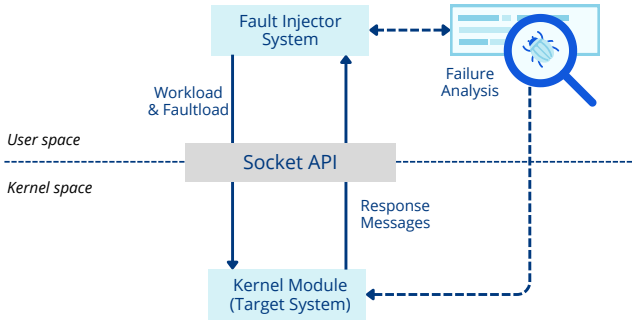

Fig. 2. An overview of the Open vSwitch robustness testing approach.

In this scenario, the robustness test is carried out by a fault injector, which was implemented as a user-space process. This fault injector is responsible for generating both the workload and the faultload, which are sent to the Open vSwitch kernel module via a Netlink socket. The kernel module processes the inputs and returns an output (whether correct or not) unless a failure prevents it from responding. Failures are detected based on the responses received (or their absence).

A. Robustness Testing Profile

A fundamental aspect of robustness testing lies in how the input sets are generated and how the tests are structured and executed; in other words, the robustness test profile. A well-defined test profile ensures that the system under test is exposed to a comprehensive and meaningful set of scenarios that reflect both typical and exceptional conditions. This includes the careful selection of input parameters, identification of input boundaries, and the injection of malformed or unexpected

values to observe system behavior. In this context, inspired by a methodology proposed in the literature [31], we follow an approach that is organized into three testing phases, as illustrated in Figure 3.

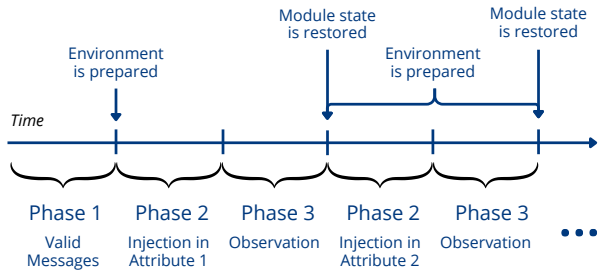

Fig. 3. Robustness testing profile.

The process starts with the selection of a Generic Netlink family to initiate the testing phases. While Phase 1 is executed only once, Phases 2 and 3 are repeated in cycles: each cycle targeting a different attribute from the selected Generic Netlink family. In Phase 1, a set of messages containing valid values is generated to form the workload.

In order to generate those messages, we first created a list of descriptors that indicate the attributes of each Netlink family (name, type, and the commands that support each attribute). The fault injector, which we implemented in Python, processes the list of descriptors and generates the sets of test cases as YAML (YAML Ain't Markup Language) files. During the execution of a test, those YAML files are used to generate each Netlink message that is sent to the module under test. In Phase 1, only valid messages are generated. This is achieved by assigning valid values to the attributes of the descriptors. These messages are sent to the Open vSwitch kernel module to observe its normal behavior in the absence of intentional faults. The purpose of Phase 1 is twofold. On the one hand, it allows the verification of the fault injector itself (is it generating correct messages?). On the other hand, it also allows to check whether the module under test is behaving correctly after receiving valid messages with the correct attributes.

Note that the environment itself has to be prepared to execute some of the fault injection experiments. This is shown in Figure 3. For example, if we are going to test an attribute in the command to add a virtual port to the datapath, we first prepare the environment by creating the datapath. Besides allowing the execution of the experiment, the purpose is frequently to ensure that any errors eventually returned are related to the invalid attribute value(s) submitted, rather than to unrelated environmental conditions.

In Phase 2, a single attribute is selected from the list of untested attributes within the chosen Generic Netlink family. Based on the attribute type, a range of invalid values is systematically generated according to the rules outlined in Table IV. These rules are designed to push the boundaries of Netlink attribute data types in an effort to uncover failures in the target module. The generated values are then used to construct Netlink messages that form the faultload, with each message representing an individual test case. Note that each of the messages generated in this phase are variations of the valid messages generated in Phase 1. These messages, containing invalid values, are subsequently sent to the Open vSwitch kernel module.

TABLE IV
RULES FOR GENERATING INVALID VALUES FOR GENERIC NETLINK ATTRIBUTES.

Type	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.*)

Afterwards, the module is reloaded into the Linux kernel (*i.e.*, it is disabled and then re-enabled) to restore it to a clean state. This ensures that any failure triggered during the test of one attribute does not interfere with the test of subsequent attributes. Finally, Phase 3 involves monitoring and analyzing the system to detect any failures caused by the previously injected faulty inputs. This is a manual procedure, described in Subsection IV-C. Phases 2 and 3 are then repeated for each attribute in the selected Generic Netlink family. Once all attributes in the current family have been tested, the next family is selected, and the process continues until all attributes across all Generic Netlink families implemented by the module have been evaluated.

B. Failure Modes

When conducting robustness tests, it is essential to differentiate failures and correct system behavior. Equally important is assessing the severity of any failures uncovered during testing. To this end, we adapted the CRASH scale [18] to suit the context of interactions with the Open vSwitch kernel module via Netlink. The CRASH scale is frequently used for robustness testing. The following failure modes were defined to categorize the observed behaviors.

- ***C**atastrophic*: The module becomes unresponsive to new inputs and must be reloaded to function correctly again. In more severe cases, the module failure may propagate and crash the entire operating system.
- ***R**estart*: A request to the module never returns an output, resulting in a stuck task that needs to be terminated (or restarted) by force;

- ***A**bort*: The transmission of either the workload or fault-load is abruptly interrupted;
- ***S**ilent*: No error or error code is returned by the tested module, when it should have been;
- ***H**indering*: An incorrect error code is returned by the tested module.

Correct behavior occurs when the system under test reacts according to its specification. In this case, the generation of an appropriate error code upon receiving an invalid input is considered correct behavior of the Open vSwitch kernel module.

C. Analysis Method

In robustness testing of software systems, the system specification typically serves as the foundation for characterizing failures (*i.e.*, for distinguishing correct from incorrect behavior). However, the current Linux kernel documentation [32] does not provide complete specifications for all Generic Netlink families. In the case of Open vSwitch, only the `ovs_datapath`, `ovs_vport`, and `ovs_flow` families are documented. Moreover, even for these families, the documentation specifies only the attribute types, without clearly defining the range of valid input values for each attribute. This poses a challenge when determining whether the module's behavior is correct in the face of an input and makes it difficult to identify failures.

In this sense, we propose an analysis method that focuses on the specification of the system's error codes. The goal is to enable the identification of system failures even in the absence of a detailed specification of the range of valid input values for each attribute. This approach relies on the assumption that the system under test implements a standardized set of error codes. In the case of the Linux kernel (and its modules), the error codes follow the definitions established by the IEEE Standard 1003.1-2001 (also known as POSIX.1-2001) [17].

The method consists of comparing the error code returned by the system with the expected error code specification, given an input generated based on one of the rules defined in Table IV. If the error code corresponds appropriately to the submitted input according to the specification, the system's behavior is considered correct. Conversely, if no error code is returned or the returned code is inappropriate, the behavior is classified as a system failure and categorized using the CRASH scale.

Another possible outcome is the return of a success code. However, such cases are expected to be rare if the module properly validates its input attributes, since at least one attribute in each request is deliberately invalid. When a success code is returned, it is further verified whether the requested operation was actually performed as intended; *e.g.*, whether a "remove" command successfully removed the correct item without affecting other data or functionalities. If the operation was not executed correctly, the case is classified as a failure.

Finally, an analysis of the consistency of the system's outputs is also carried out. For example, consider that the execution of a certain command by submitting an attribute x with a value y returns an error indicating that the attribute value is invalid. In this case, an inconsistency is identified if the same attribute-value pair ($x = y$) is considered valid when executing another command.

V. Results

In this section, we present the results obtained from the robustness tests performed. The fault injector was implemented in Python using the pyroute2[1] library, which implements Netlink communication. Some of the encoding methods in the library have been customized to allow invalid values in order to perform the robustness tests. The source code and associated artifacts are publicly available[2].

The kernel versions and Linux distributions used in the experiments are presented in Table V. We selected the latest releases of each major kernel version since the Open vSwitch module was integrated. Thus, the oldest is version 3.19.8, since Open vSwitch was integrated in the 3 series version, and the most recent is the current stable version of the Linux kernel at the time of writing, v6.14.6, released in May 2025. The Ubuntu distributions were chosen due to their proximity to the release date of each kernel being tested, in order to ensure better compatibility.

TABLE V
Summary of the environments used for the robustness tests.

Operating System	Kernel Version	Kernel Release Date
Ubuntu 15.04	v3.19.8	April 2015
Ubuntu 19.04	v4.20.9	February 2019
Ubuntu 22.04	v5.19.9	September 2022
Ubuntu 25.04	v6.14.6	May 2025

Initially, in Subsection V-A, we present the results obtained across all evaluated kernel versions, providing a comparative analysis. Subsection V-B highlights key results obtained across all the kernel versions. Following that, in Subsection V-C, we offer a detailed analysis of the robustness evaluation results for the Open vSwitch module in the most recent stable release of the Linux kernel.

A. Cross-Version Robustness Testing Results

Since it was first released, Open vSwitch has undergone continuous development, with new features and enhancements introduced with each new release. In this sense, the present work evaluates how the Open vSwitch kernel module has evolved over the years in terms of robustness.

Table VI shows the results of the robustness tests on four different versions of the Linux kernel. In the oldest of those kernel versions, v3.19.8, only the `ovs_datapath` and `ovs_packet` families were successfully tested due to technical reasons. In this version, we faced an "Invalid Argument" error when using the ovs-dpctl command line tool to add flow rules in order to prepare the environment for testing.

Unfortunately, even after consulting the official documentation and the community, it was not possible to find out

[1]https://pypi.org/project/pyroute2/

[2]https://doi.org/10.5281/zenodo.16883930

TABLE VI
RESULTS ACROSS DIFFERENT KERNEL VERSIONS.

Kernel Version	Generic Netlink Family	Expected Behavior (%)	Failure (%)	C	R	A	S	H
v3.19.8	ovs_datapath	94.64	5.36	–	–	–	–	3
	ovs_vport	∗	∗	∗	∗	∗	∗	∗
	ovs_flow	∗	∗	∗	∗	∗	∗	∗
	ovs_packet	87.50	12.50	–	–	–	–	3
	ovs_meter	∅	∅	∅	∅	∅	∅	∅
	ovs_ct_limit	∅	∅	∅	∅	∅	∅	∅
v4.20.9	ovs_datapath	94.64	5.36	–	–	–	–	3
	ovs_vport	51.35	48.65	–	–	–	–	18
	ovs_flow	62.07	37.93	–	–	–	1	32
	ovs_packet	83.33	16.67	–	–	–	–	4
	ovs_meter	48.39	51.61	–	–	–	4	12
	ovs_ct_limit	100	0	–	–	–	–	–
v5.19.9	ovs_datapath	75.00	25.00	–	–	–	–	14
	ovs_vport	62.16	37.84	–	–	–	–	14
	ovs_flow	62.07	37.93	–	–	–	1	32
	ovs_packet	62.50	37.50	–	–	–	–	9
	ovs_meter	48.39	51.61	–	–	–	4	12
	ovs_ct_limit	100	0	–	–	–	–	–
v6.14.6	ovs_datapath	73.21	26.79	–	–	–	–	15
	ovs_vport	62.16	37.84	–	–	–	–	14
	ovs_flow	62.07	37.93	–	–	–	1	32
	ovs_packet	62.50	37.50	–	–	–	–	9
	ovs_meter	48.39	51.61	–	–	–	4	12
	ovs_ct_limit	100	0	–	–	–	–	–

– Indicates that no such failure was found.
∗ Indicates families that could not be tested due to technical reasons.
∅ Indicates families that had not been implemented in the given version.

whether this was due to a syntax variation in this old version or if some incompatibility between software versions in user and kernel space caused the error. The affected results are represented by ∗ in Table VI. Furthermore, the `ovs_meter` and `ovs_ct_limit` families had not been implemented in this version, as indicated by the ∅ symbol in the table.

In all kernel versions tested, a considerable portion of the faultload submitted to the Open vSwitch kernel module was handled correctly. Nevertheless, the tests revealed multiple failure cases, but only silent (S) and hindering (H) failures were observed, with hindering being the most prevalent. This result demonstrates the kernel's resilience in preventing failures that lead to partial (R) or total (C) disruption of the system.

B. Robustness Assessment across Kernel Versions: Highlights

This subsection presents highlights of the results obtained across all kernel versions tested. Highlights are presented from two different perspectives. First, the focus is on the evolution of the Generic Netlink families implemented by the Open vSwitch module in new kernel versions in comparison with older ones. Then, our attention shifts to the behavior of individual commands within these families, examining how their robustness evolved over time.

1) Evolution of Generic Netlink Families: Table VI indicates that some families have shown improvements in robustness, while others have either deteriorated or remained unchanged throughout the versions. In the case of the `ovs_vport` family, there was a decrease from 18 failures in version 4.20.9 to 14 total failures in the subsequent tested versions (5.19.9 and 6.14.6). On the other hand, a significant increase was observed in two families: `ovs_datapath` and `ovs_packet`.

The most significant increase in the number of failures occurred with the `ovs_datapath` family, which experienced an increase from 3 to 15 failures between versions 3.19.8 and 6.14.6, respectively. In the case of the `ovs_flow` and `ovs_meter` families, the number of failures remained unchanged across versions. Despite this, the `ovs_flow` family had the highest total number of failures: 33, including both silent and hindering failures.

To facilitate the visualization of the results presented in Table VI, Figure 4 shows the distribution of the total number of failures of each kernel version – those families that could not be tested in certain kernel versions appear with values set to 0 (zero). Thus, it is possible to see in Figure 4 that the `ovs_flow` family presents a substantially higher number of failures than the other families. Furthermore, the overall darker shading that can be perceived towards the right side of the figure indicates that the number of failures in more recent kernel versions has actually *increased*.

For a better comparison between the Netlink families that were tested, Figure 5 illustrates the progression in terms of the failure rate; *i.e.*, the proportion of test cases that triggered a failure. Observing the figure, it is possible to note that from version 5.19.9 onwards, all families (except `ovs_ct_limit`) had a failure rate above 20%. Most of them, in fact, were close to (or even above) 40%.

In the case of the `ovs_flow` and `ovs_meter` families, the failure rates remained unchanged over versions and were

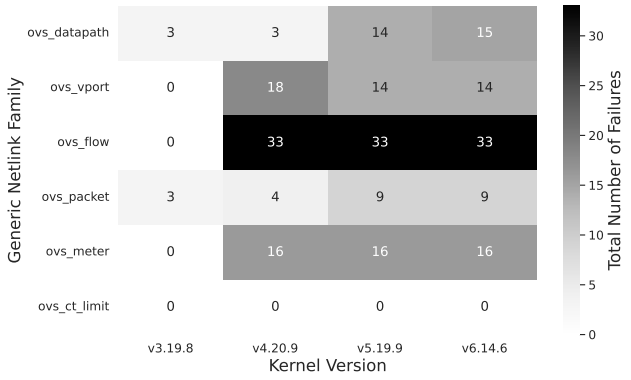

Fig. 4. Distribution of the total number of failures in the Open vSwitch Generic Netlink families across kernel versions.

notably high. This, combined with the observation that only the `ovs_vport` family has shown a modest improvement in robustness, suggests that robustness has historically not been a priority in the development of the Open vSwitch kernel module.

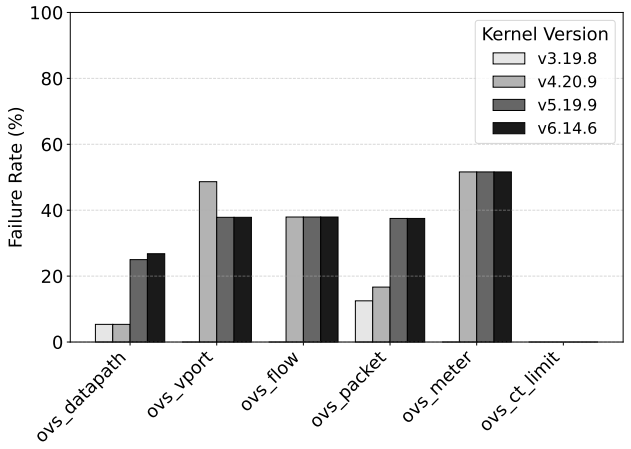

Fig. 5. Evolution of failure rates across kernel versions.

2) Evolution of Commands: The Open vSwitch kernel module has a well-defined set of commands for each Generic Netlink family (as shown in Table II). The table reveals the evolution of the robustness of each of those commands over time. Table VII presents the number of failures identified for each command across the four different Linux kernel versions tested.

A particular highlight is the behavior of the `*_NEW` and `*_SET` commands (*i.e.*, commands to add or modify elements). Notably, these commands presented very similar tendencies to improve or degrade, suggesting they may share implementation dependencies. For example, in the case of the `ovs_datapath` family, the `OVS_DP_CMD_NEW` and `OVS_DP_CMD_SET` commands experienced an increase of 6 failures between the oldest and newest kernel versions

TABLE VII
NUMBER OF FAILURES OBSERVED PER COMMAND AND KERNEL VERSION.

	Command	Failures by Kernel Versions			
		v3.19.8	v4.20.9	v5.19.9	v6.14.6
ovs_datapath	OVS_DP_CMD_NEW	2	2	7	8
	OVS_DP_CMD_DEL	0	0	0	0
	OVS_DP_CMD_GET	0	0	0	0
	OVS_DP_CMD_SET	1	1	7	7
ovs_vport	OVS_VPORT_CMD_NEW	*	15	13	13
	OVS_VPORT_CMD_DEL	*	0	0	0
	OVS_VPORT_CMD_GET	*	0	0	0
	OVS_VPORT_CMD_SET	*	3	1	1
ovs_flow	OVS_FLOW_CMD_NEW	*	8	8	8
	OVS_FLOW_CMD_DEL	*	7	7	7
	OVS_FLOW_CMD_GET	*	7	7	7
	OVS_FLOW_CMD_SET	*	11	11	11
ovs_packet	OVS_PACKET_CMD_MISS	-	-	-	-
	OVS_PACKET_CMD_ACTION	-	-	-	-
	OVS_PACKET_CMD_EXECUTE	3	4	9	9
ovs_meter	OVS_METER_CMD_FEATURES	∅	-	-	-
	OVS_PACKET_CMD_EXECUTE	∅	10	10	10
	OVS_METER_CMD_DEL	∅	5	5	5
	OVS_METER_CMD_GET	∅	1	1	1
ovs_ct_limit	OVS_CT_LIMIT_CMD_SET	∅	0	0	0
	OVS_CT_LIMIT_CMD_DEL	∅	0	0	0
	OVS_CT_LIMIT_CMD_GET	∅	0	0	0

– Does not apply (command only for upcalls).
* Indicates that the family of the given command could not be tested for technical reasons.
∅ Indicates that the family of the given command was not implemented in that version.

tested. Both commands invoke several functions in common, including the `ovs_dp_change()` function.

Given all versions tested, the highest number of failures was found in the `OVS_VPORT_CMD_NEW` command, reaching 15 failures in the worst case (kernel 4.20.9). As mentioned earlier, the `ovs_ct_limit` family is considerably leaner than the others, containing only a single Generic Netlink attribute, and does not reveal failures in any of the 3 available commands.

C. Robustness Testing on the Latest Kernel

Having examined the evolution across various Linux kernel versions, this subsection provides a more in-depth analysis of the Open vSwitch kernel module as implemented in the latest stable kernel release, version 6.14.6. In total, all 31 attributes supported by the 19 commands capable of executing requests to the Open vSwitch kernel module were tested. Next, we present the analysis of failures and inconsistencies found through the robustness tests. We have also investigated which rules for generating invalid values triggered the most failures.

1) Analysis of Failures: A considerable portion of the faultload submitted to the Open vSwitch kernel module was handled correctly in this version. In several test cases, the modul5 returned the `EINVAL` error code (Invalid argument – code 22), which is appropriate, as it indicates that the submitted value was invalid and the system does not proceed to execute the requested command. Additionally, there were also cases that can be considered correct in which, despite the request containing an attribute with an invalid value, the kernel

module executed the command successfully and returned a success code — nevertheless, from a strict point of view, this behavior is not really "correct".

As summarized in Table VI and presented at a high level throughout Subsection V-A, the robustness testing of the current stable kernel version also revealed several failures. The ovs_meter family exhibited the highest failure rate, with 51.61% of test cases resulting in failures. In contrast, all test cases for the ovs_ct_limit family demonstrated correct behavior. It is important to note, however, that the ovs_ct_limit family comprises only one attribute across 3 commands, whereas the ovs_meter family includes 5 attributes used in 4 different commands. This difference leads to a significantly larger number of test cases for ovs_meter, which impacts the observed failure rate. The family with the highest absolute number of failures was ovs_flow, with a total of 32 failures detected.

Among the hindering failures (*i.e.*, cases in which the module returns an incorrect error code), the most frequent was ERANGE (Numerical result out of range – code 34). According to the POSIX.1-2001 standard, this error indicates that a computed result is too large (overflow) or too small (underflow) to be represented. Its occurrence suggests that the Open vSwitch module failed to validate the input properly, allowing the invalid value to propagate until the kernel encountered an overflow or underflow. A concrete example arises in the ovs_flow family when executing the flow rule insertion command (OVS_FLOW_CMD_NEW) with the StructNull rule applied to the OVS_FLOW_ATTR_UFID attribute, which represents the unique identifier of the flow.

In the ovs_datapath family, an ERANGE error is also returned when the StructNull rule is applied to the OVS_DP_ATTR_UPCALL_PID attribute of the OVS_DP_CMD_NEW command. This attribute specifies the process ID to which the kernel module should send upcalls (*i.e.*, messages from the kernel to user space) related to the datapath. In Linux, the maximum valid process ID is 4,194,304, as defined by the PID_MAX_LIMIT macro. Consequently, this attribute should only accept values in the range 0 (which indicates that no upcalls should be sent) to PID_MAX_LIMIT. Any value outside this range (including a null value) should trigger an EINVAL error rather than ERANGE. For this reason, that case is considered a hindering (H) failure.

Table VI also reports S (silent) failures of the Open vSwitch module in kernel version 6.14.6. Out of the 5 silent failures identified, 4 occurred within the ovs_meter family, specifically with the OVS_METER_ATTR_ID attribute when executing OVS_METER_CMD_DEL commands. These failures were observed when applying the NumMinType, NumMaxType, NumUnderflow, and NumOverflow rules to the OVS_METER_ATTR_ID attribute. In these cases, despite not having any matches for the IDs used (which meant no meters were actually removed), the module returned a success code. The correct behavior should have been the return of an EINVAL error code, indicating that the operation could not be performed due to invalid input. Given this behavior, one would expect at least an error such as ENOENT ("No such file or directory" - code 2), meaning that no meter was found for the specified ID value.

Another example of silent failure was identified when creating a flow rule by applying the StructPrimitive rule to the OVS_FLOW_ATTR_UFID attribute, instead of using a correctly structured ID. In this case, the module returned a success code, indicating that no error occurred during the operation. However, despite this indication, the flow rule could not be retrieved using the corresponding request. Additionally, this failure prevented the listing of all flow rules via the command line tool (ovs-dpctl), resulting in the error message: "Failed to dump flows from datapath (Invalid argument)."

Table VIII presents the attributes and commands of each Open vSwitch Generic Netlink family in which failures were observed. Notably, in the ovs_flow and ovs_packet families, all commands that support operation requests to the kernel module exhibited failures.

TABLE VIII
ATTRIBUTES AND COMMANDS IN WHICH FAILURES OCCURRED.

	Attribute	Command
ovs_datapath	OVS_DP_ATTR_MASKS_CACHE_SIZE OVS_DP_ATTR_UPCALL_PID OVS_DP_ATTR_USER_FEATURES	OVS_DP_CMD_NEW OVS_DP_CMD_SET
ovs_vport	OVS_VPORT_ATTR_TYPE OVS_VPORT_ATTR_IFINDEX OVS_VPORT_ATTR_PORT_NO	OVS_VPORT_CMD_NEW OVS_VPORT_CMD_SET
ovs_flow	OVS_FLOW_ATTR_UFID OVS_FLOW_ATTR_UFID_FLAGS OVS_FLOW_ATTR_PROBE OVS_FLOW_ATTR_CLEAR	OVS_FLOW_CMD_NEW OVS_FLOW_CMD_GET OVS_FLOW_CMD_SET OVS_FLOW_CMD_DEL
ovs_packet	OVS_PACKET_ATTR_HASH OVS_PACKET_ATTR_PACKET OVS_PACKET_ATTR_MRU	OVS_PACKET_CMD_EXECUTE
ovs_meter	OVS_METER_ATTR_ID OVS_METER_ATTR_CLEAR OVS_METER_ATTR_KBPS OVS_METER_ATTR_STATS	OVS_METER_CMD_SET OVS_METER_CMD_DEL

2) Analysis of Predominant Failure Triggers: An important aspect to analyze is which injected faults triggered the greatest numbers of failures. Figure 6 captures the relationship between the rules used to generate invalid values (*i.e.*, inject faults) and the total number of failures triggered, grouped by Generic Netlink families. As shown, the rule that resulted in the highest number of failures was NumNull, with 6 occurrences in the ovs_flow family. Overall, the results indicate that injecting faults into numeric attributes of the Open vSwitch kernel module triggered more failures. A contributing factor to this result is that in almost all of the families implemented by the module, numerical attributes constitute the majority. In any case, the reported results imply that those attributes require greater attention regarding robustness.

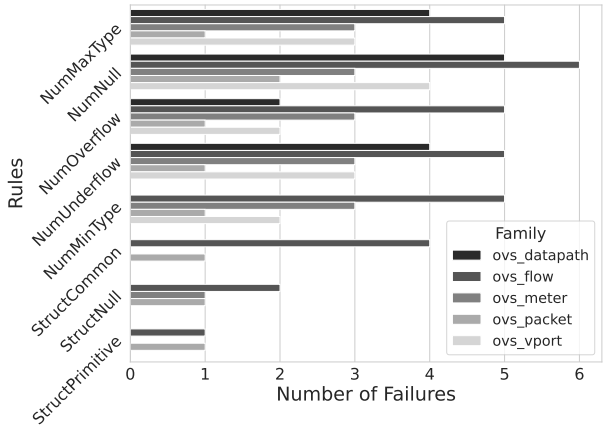

Fig. 6. Distribution of failures by rules used to generate invalid inputs.

Furthermore, the absence in Figure 6 of some of the string-related rules listed in Table IV also gives relevant indications: faults injected via string attributes did not trigger failures in the current version of the Open vSwitch kernel module. However, subsection V-C3 reports another sort of issue with this type of attribute.

3) Analysis of Inconsistencies: In addition to the detected failures, the robustness assessment also revealed inconsistencies in attribute validation and the corresponding error codes returned by the Open vSwitch kernel module. A clear example occurs with the `OVS_DP_ATTR_NAME` attribute from the `ovs_datapath` family. When creating a datapath using the `OVS_DP_CMD_NEW` command, the module correctly returns an `EINVAL` error for values generated with the StrEmpty and StrNonPrintable rules. However, when the same invalid values are submitted to the commands for listing, modifying, or removing datapaths, the module instead returns an `ENODEV` error ("No such device" - code 19). This behavior indicates that these commands fail to properly validate the attribute, proceeding to process the invalid values to the point of searching for them in the records and reporting that they do not exist.

Another similar inconsistency was observed in the `OVS_VPORT_CMD_NEW` command of the `ovs_vport` family. This command is responsible for adding a new port to the datapath by associating it with a previously created virtual interface. When executing `OVS_VPORT_CMD_NEW` with the `OVS_VPORT_ATTR_NAME` attribute set to a value containing non-printable characters (as defined by the StrNonPrintable rule), the module returns an `ENODEV` error, indicating that it attempted to find a network interface with that name but could not. However, such an interface name would be invalid in Linux, as assigning a name with non-printable characters is not permitted and would result in an `EPERM` error ("Operation not permitted" - code 1) when attempted via the command line. Despite the inconsistency in validation, this case was not classified as a failure because the error returned (`ENODEV`) correctly reflected the absence of an interface with the specified name during the test.

A relevant situation also arises with the `OVS_METER_ATTR_ID` attribute of the `ovs_meter` family, which represents the identifier of a meter. The Open vSwitch kernel module fails to properly validate this attribute, allowing meters to be assigned an ID of 0 (zero) or even a negative value. This lack of validation could lead to cascading failures in other components of the system that are not designed to handle such values, potentially compromising their correct operation.

VI. Discussion

The results reveal that the Open vSwitch kernel module – despite its maturity and wide deployment – still exhibits a considerable range of robustness issues when handling invalid inputs via its Netlink-based interface. Those previously undocumented issues span multiple kernel versions, indicating that robustness concerns are not isolated to specific releases.

The current approach used in the Linux kernel to automatically validate Netlink attributes is based on `nla_policy` (Netlink Attribute Policy) contracts. Therefore, the several robustness failures found suggest that those policies are not being consistently used or are not sufficient to fully validate all Netlink attributes. In this regard, we believe that development efforts should be focused on fostering practices to promote the consistent use of the Netlink Attribute Policies. It may include adding automatic policy coverage checks (*e.g.*, applying linting tools to flag missing `nla_policy` entries) or even adopting development practices that enforce the mandatory use of `nla_policy` contracts at the code level for all Netlink attributes.

Regarding Netlink-based interfaces, the Linux kernel currently includes only functional tests. Incorporating robustness tests, specifically targeting invalid input values, into the continuous integration (CI) pipelines would be essential to expose such issues early in the development stage. Finally, refining both the Netlink documentation for developers and the documentation of existing Netlink interfaces would be significantly beneficial to improve the robustness of the Open vSwitch kernel module and other modules with Netlink-based interfaces.

VII. Threats to Validity and Limitations

In this section, we present a non-exhaustive list of potential threats to the validity, as well as the limitations of the work. These concerns are categorized into three main aspects: internal, external, and construct. To address these issues appropriately, each aspect is discussed individually below, in order to clarify their potential impact and implications on the conclusions drawn.

A. Internal Threats

One potential threat lies in the fault injection strategy employed. Our approach systematically explores Generic Netlink attributes by generating invalid values in an effort to trigger failures. However, it may not cover the entire range of faulty

or malformed inputs that could be encountered in real-world scenarios. We mitigate this by employing rules specifically designed to explore boundary conditions and edge cases of the attribute data types. Despite its inherent limitations, this strategy provides meaningful insights into the robustness of the Open vSwitch kernel module with a reduced number of test cases.

Since the total number of failures identified in each family is directly influenced by the specific rules employed for generating invalid input values, the reported results may differ from the actual number of failures. To mitigate this limitation and provide a more balanced perspective, we also analyzed the failure rate on a per-family basis.

Another limiting factor with regard to failure coverage was the issue faced when testing the older kernel, v3.19.8. This prevented the robustness assessment of two Generic Netlink families: `ovs_vport` and `ovs_flow`. Despite this, the assessment of three other kernel versions, which included these families, enabled a comprehensive evaluation of their robustness over time.

B. External Threats

Our results are based on evaluations of four specific versions of the Linux kernel, selected to span a timeline from the last release of the 3.x series (where the module was initially integrated) to the most recent stable release in the 6.x series. However, these results may not be fully generalized for all possible kernel versions in the range tested (*i.e.*, all releases from 3.x to 6.x). Nonetheless, the selected kernel versions include representative milestones in the evolution of Open vSwitch. Future work could expand coverage to other Linux kernel releases.

Lastly, the results presented cannot be generalized to other Linux kernel modules. However, the proposed robustness evaluation approach can potentially be used to assess the robustness of other Linux kernel modules that implement a Generic Netlink interface.

C. Construct Threats

Another concern is whether our robustness evaluation approach accurately captures the full concept of robustness. In particular, we identify failures based on submitting invalid input values (which is a well-known technique to address robustness [33]). Although it offers important indications, some failures may not arise from these stimuli. In this sense, future work could integrate complementary techniques to generate stressful conditions in the environment to uncover an even more extensive set of failures.

VIII. Conclusion

In this paper, we assessed the robustness of the Open vSwitch kernel module over four different versions of the Linux kernel. The proposed approach relies on a fault injection technique, whereby invalid input values are systematically generated and submitted through the Netlink interface exposed by the Open vSwitch module. In this sense, a robustness test profile was defined to establish the test phases and the set of rules for generating invalid input values that exploit the boundaries of the Netlink attribute data types to trigger failures in the tested module. To overcome the lack of documentation regarding the range of input values accepted for each attribute, an analysis method based on the specification of system error codes was also proposed. The method is based on the POSIX specification and an adaptation of the CRASH scale for Netlink-based communications.

The results revealed several hindering and silent failures in the Open vSwitch kernel module in all versions tested. The case with the highest absolute number of failures is the `ovs_flow` family, with a total of 33 failures in each of the last three kernel versions tested. Furthermore, a significant deterioration in the robustness of the `ovs_datapath` and `ovs_packet` families was observed, reaching a 25% increase in the failure rate of the `ovs_packet` family from version 3.19.8 to 6.14.6. In the latest stable release of the Linux kernel, version 6.14.6, a total of 82 hindering failures and 5 silent failures were identified in the Open vSwitch module. These findings suggest that the Open vSwitch kernel module requires greater attention to its robustness. At the same time, they underscore the ongoing importance of evaluating the robustness of modern software systems.

Future work includes an analysis of the possible effects of failures in the system's fundamental functionalities, which in the case of Open vSwitch is packet processing. An assessment of robustness in a stressful environment (such as under a large volume of requests and even high network traffic loads) is also relevant. Finally, future investigations will explore vulnerabilities that may arise from the failures and inconsistencies that were identified, along with effective strategies for their mitigation.

References

[1] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[2] ETSI, "Network Functions Virtualisation (NFV) Release 5; Management and Orchestration; Architectural Framework Specification," European Telecommunications Standards Institute, Tech. Rep., 2024.

[3] G. Venâncio, R. C. Turchetti, and E. P. Duarte Jr, "Nfv-coin: unleashing the power of in-network computing with virtualization technologies," *Journal of Internet Services and Applications*, vol. 13, no. 1, pp. 46–53, 2022.

[4] B. Pfaff *et al.*, "The Design and Implementation of Open vSwitch." in *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'2015)*. USENIX, 2015, pp. 117–130.

[5] V. Fulber-Garcia, E. P. Duarte Jr, A. Huff, and C. R. dos Santos, "Network service topology: Formalization, taxonomy and the custom specification model," *Computer Networks*, vol. 178, p. 107337, 2020.

[6] T. L. Foundation, "Open vSwitch," https://www.openvswitch.org/, 2025, accessed in May 2025.

[7] Y. Yan and H. Wang, "Open vswitch vxlan performance acceleration in cloud computing data center," in *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*. IEEE, 2016, pp. 567–571.

[8] J. Tseng, R. Wang, J. Tsai, Y. Wang, and T.-Y. C. Tai, "Accelerating open vswitch with integrated gpu," in *Proceedings of the Workshop on Kernel-Bypass Networks*, 2017, pp. 7–12.

[9] E. J. Jackson, M. Walls, A. Panda, J. Pettit, B. Pfaff, J. Rajahalme, T. Koponen, and S. Shenker, "SoftFlow: A Middlebox Architecture for Open vSwitch," in *2016 Usenix Annual Technical Conference (USENIX ATC 16)*, 2016, pp. 15–28.

[10] S. Shanmugalingam, A. Ksentini, and P. Bertin, "Dpdk open vswitch performance validation with mirroring feature," in *2016 23rd International Conference on Telecommunications (ICT)*. IEEE, 2016, pp. 1–6.

[11] F. Sans and E. Gamess, "Analytical performance evaluation of different switch solutions," *Journal of Computer Networks and Communications*, vol. 2013, no. 1, p. 953797, 2013.

[12] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[13] G. Venâncio, V. Fulber-Garcia, J. Flauzino, E. A. Alchieri, and E. P. Duarte, "Dependable Virtual Network Services: An Architecture for Fault-and Intrusion-tolerant SFCs," in *2024 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2024, pp. 1–6.

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

[15] N. Laranjeiro, J. Agnelo, and J. Bernardino, "A systematic review on software robustness assessment," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–65, 2021.

[16] A. Shahrokni and R. Feldt, "A systematic review of software robustness," *Information and Software Technology*, vol. 55, no. 1, pp. 1–17, 2013.

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

[18] 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.

[19] J. Arlat, J.-C. Fabre, M. Rodríguez, and F. Salles, *MAFALDA: A Series of Prototype Tools for the Assessment of Real Time COTS Microkernel-Based Systems*. Boston, MA: Springer US, 2003, pp. 141–156.

[20] 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.

[21] Y. Ait-Ameur, G. Bel, F. Boniol, S. Pairault, and V. Wiels, "Robustness Analysis of Avionics Embedded Systems," in *Proceedings of the 4th ACM SIGPLAN Conference on Language, Compiler, and Tools for Embedded Systems (LCTES'2003)*. New York, NY, USA: Association for Computing Machinery, 2003, p. 123–132.

[22] A. Cavalli, E. Martins, and A. Morais, "Use of invariant properties to evaluate the results of fault-injection-based robustness testing of protocol implementations," in *Proceedings of the 1st IEEE International Conference on Software Testing Verification and Validation Workshop (ICST'2008)*, 2008, pp. 21–30.

[23] N. Laranjeiro, S. Canelas, and M. Vieira, "wsrbench: An On-Line Tool for Robustness Benchmarking," in *Proceedings of the 5th IEEE International Conference on Services Computing (SCC'2008)*, vol. 2. IEEE, 2008, pp. 187–194.

[24] C. Hutchison, M. Zizyte, P. E. Lanigan, D. Guttendorf, M. Wagner, C. Le Goues, and P. Koopman, "Robustness Testing of Autonomy Software," in *Proceedings of the 40th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP'2018)*, 2018, pp. 276–285.

[25] J. Cámara, R. De Lemos, N. Laranjeiro, R. Ventura, and M. Vieira, "Robustness-driven resilience evaluation of self-adaptive software systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 1, pp. 50–64, 2015.

[26] S. Scott-Hayward, "Design and deployment of secure, robust, and resilient sdn controllers," in *Proceedings of the 2015 1st IEEE conference on network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–5.

[27] L. V. Ruchel, R. C. Turchetti, and E. T. de Camargo, "Evaluation of the robustness of sdn controllers onos and odl," *Computer Networks*, vol. 219, p. 109403, 2022.

[28] B. Pfaff and B. Davie, "The Open vSwitch Database Management Protocol," RFC 7047, Dec. 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc7047

[29] 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.

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

[31] N. Laranjeiro, M. Vieira, and H. Madeira, "Experimental Robustness Evaluation of JMS Middleware," in *Proceedings of the 5th IEEE International Conference on Services Computing (SCC'2008)*, vol. 1. IEEE, 2008, pp. 119–126.

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

[33] Z. Micskei, H. Madeira, A. Avritzer, I. Majzik, M. Vieira, and N. Antunes, "Robustness testing techniques and tools," *Resilience assessment and evaluation of computing systems*, pp. 323–339, 2012.