Avaliação da Robustez do Open vSwitch: Módulo do *Kernel* Linux

José Flauzino[1], Marco Vieira[2], Elias P. Duarte Jr.[1]

[1] Universidade Federal do Paraná (UFPR), Depto. Informática,
Caixa Postal 19081 Curitiba, PR, Brasil
[2] University of North Carolina at Charlotte,
Woodward 205C 9201 Univ. City Blvd. Charlotte, NC 28223, USA

{jwvflauzino,elias}@inf.ufpr.br, marco.vieira@charlotte.edu

Resumo. *O Open vSwitch é uma implementação em software de um switch de rede multicamada projetado para o contexto de virtualização. Sua arquitetura engloba componentes nos espaços de usuário e kernel. Integrado ao kernel Linux desde 2012, o Open vSwitch é hoje um projeto maduro e amplamente adotado em ambientes virtualizados. Contudo, apesar do Open vSwitch ser objeto de estudo de diversos trabalhos com foco no desempenho, sua robustez ainda é uma incógnita. Este trabalho propõe uma abordagem para a avaliação da robustez de um componente fundamental do Open vSwitch, o módulo de kernel. A abordagem é baseada na injeção de falhas na interface Netlink do módulo de kernel. A avaliação de robustez conduzida revela falhas de diversas naturezas e indica uma inconsistência ao lidar com as adversidades impostas pelos testes.*

1. Introdução

A expansão acelerada da Internet, projetada há várias décadas, contribuiu para um cenário de infraestrutura extremamente complexa e que torna mudanças em sua arquitetura e protocolos um desafio substancial. Este problema, conhecido como a Ossificação da Internet [Turner and Taylor 2005], representa uma barreira significativa à inovação. Diante disso, as décadas de 2000 e 2010 foram marcadas pelo surgimento de diversas abordagens para prover maior flexibilidade às redes. Isso inclui novos paradigmas de rede como SDN (*Software-Defined Networking*) e NFV (*Network Functions Virtualization*).

É justamente neste contexto que foi proposto o Open vSwitch [Pfaff et al. 2015], uma implementação em software de um *switch* de rede multicamada projetado especialmente para ser utilizado em ambientes virtualizados. Sua arquitetura engloba tanto componentes do espaço de usuário, quanto do espaço de *kernel*. A principal característica do Open vSwitch é ser flexível e de propósito geral, mas ainda alcançar altos níveis de desempenho. Atualmente, o Open vSwitch é um projeto maduro, mantido pela Linux Foundation, e considerado a principal solução de *switch* virtual, sendo empregado em serviços de rede virtualizados [Fulber-Garcia et al. 2020] e grandes ambientes de produção de modo geral [Foundation 2025].

A flexibilidade e o desempenho, no entanto, não são os únicos aspectos a serem considerados em sistemas modernos. A confiança no funcionamento de um sistema, ou seja, a dependabilidade (*dependability*), é outro fator de relevância crucial [Avizienis et al. 2004, Venâncio et al. 2024]. Isso diz respeito a diversos atributos do sistema, incluindo a robustez. O termo robustez pode ser definido como o grau em que um

sistema ou componente pode funcionar corretamente na presença de entradas inválidas ou condições ambientais estressantes [IEEE 2017].

Neste sentido, desde meados da década de 1990, a robustez de sistemas de software tem sido um tópico relevante e abordado por diversos trabalhos. Na literatura é possível encontrar trabalhos que avaliam a robustez de sistemas em variados contextos, incluindo sistemas operacionais, sistemas embarcados, serviços Web, entre outros [Laranjeiro et al. 2021]. No caso do Open vSwitch, no entanto, os esforços de pesquisa têm sido voltados principalmente a seu desempenho no processamento de pacotes, não havendo (até onde se sabe) estudos referentes à sua robustez.

O presente trabalho propõe uma abordagem para a avaliação da robustez de um componente que pode ser considerado a base do Open vSwitch, o módulo de *kernel* Linux. A abordagem é baseada em uma técnica de injeção de falhas na qual entradas sistematicamente selecionadas são submetidas ao módulo de *kernel* do Open vSwitch. Isso é feito através da interface de comunicação entre o espaço de usuário e o módulo, a qual é estabelecida por meio de *sockets* Netlink. A avaliação de robustez conduzida revela falhas do módulo de *kernel*, as quais são mapeadas e as possíveis causas raiz são apontadas. Uma das conclusões é que mesmo com a ampla disseminação de técnicas para avaliação de robustez de software, iniciadas há 3 décadas atrás, sistemas de software maduros e estáveis ainda apresentam falhas de robustez.

O restante do trabalho está organizado da seguinte forma. A seção 2 apresenta uma visão geral do Open vSwitch, incluindo detalhes sobre a sua interface Netlink, cuja robustez está sendo avaliada. Na Seção 3 são apresentados os principais trabalhos relacionados. A Seção 4 apresenta a abordagem proposta para a avaliação da robustez do módulo de *kernel* do Open vSwitch. Os resultados da avaliação são apresentados na Seção 5. Por fim, a Seção 6 conclui o trabalho.

2. Open vSwitch: Arquitetura & Funcionamento

O Open vSwitch possui componentes tanto no espaço de usuário quanto no espaço de *kernel*. A Figura 1 apresenta uma visão geral da sua arquitetura. O maior componente do Open vSwitch é o ovs-vswitchd, implementado como um *daemon* (*i.e.*, um processo em plano de fundo) que é executado no espaço de usuário. Outro componente fundamental é um módulo do *kernel* que implementa os chamados *datapaths* (caminhos de dados), que já faz parte oficialmente do *kernel* Linux desde a versão 3.3, lançada em março de 2012.

Os pacotes de rede são efetivamente recebidos e encaminhados no espaço de *kernel*, pelo módulo de *kernel* do Open vSwitch. Assim, quando chega um pacote no módulo, é realizada uma consulta nas regras de fluxos já recebidas (se houver). Estas regras estabelecem, por exemplo, as portas físicas ou virtuais pelas quais os pacotes de determinado fluxo devem ser transmitidos. Portanto, se existir uma regra para o fluxo daquele pacote, o módulo de *kernel* realiza o devido encaminhamento. Caso contrário, ele encaminha o pacote para o ovs-vswitchd, no espaço de usuário. O ovs-vswitchd se encarrega de descobrir se há uma regra de fluxo definida para aquele pacote (isso é realizado através da interação com o ovsdb-server ou o Controlador SDN, ambos descritos adiante). Caso o ovs-vswitchd encontre uma regra correspondente, devolve o pacote para o módulo de *kernel* junto às ações a serem desempenhadas para encaminhá-lo. Caso contrário, o pacote é descartado (*dropped*). As ações recebidas pelo módulo de *kernel* são armazenadas em

sua cache (volátil), para o tratamento de futuros pacotes semelhantes.

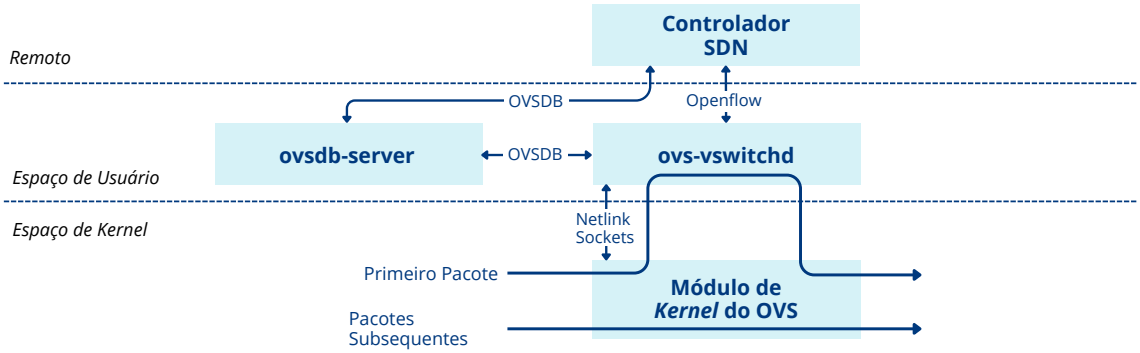

Figura 1. Visão geral da arquitetura do Open vSwitch.

A Figura 1 apresenta ainda o ovsdb-server (também implementado como um *deamon*). Este é um banco de dados próprio do Open vSwitch que é responsável por persistir as configurações relacionadas ao *switch*. Isto inclui, por exemplo, as configurações dos *datapaths*, as regras de fluxos de pacotes, as portas virtuais configuradas, entre outras. Este banco de dados é gerenciado através de um protocolo desenvolvido especialmente para ele, o OVSDB (especificado na RFC 7047) [Pfaff and Davie 2013].

Para viabilizar o uso do Open vSwitch em ambientes SDN, o ovs-vswitchd disponibiliza uma interface compatível com o protocolo OpenFlow. Através desta interface, Controladores SDN podem gerenciar as regras de fluxo. No entanto, o protocolo OpenFlow não suporta operações que vão além das regras de fluxo, como adicionar e remover portas no *switch*, por exemplo. Por isso, para um Controlador SDN gerenciar o Open vSwitch é preciso se comunicar com o ovsdb-server através do protocolo OVSDB. Uma vez que o OSVDB é formalmente especificado na RFC 7047, a maioria dos controladores SDN existentes oferecem suporte a este tipo de gerenciamento.

Na Figura 1 é possível notar que a comunicação entre o ovs-vswitchd e o módulo de *kernel* do Open vSwitch ocorre através de *sockets* Netlink. O *socket* é uma interface que provê uma abstração para comunicação entre processos. Um *socket* possui três atributos essenciais: *i*) *Address Family* (AF), ou simplesmente a família do *socket* (*e.g.*, `AF_INET`, `AF_INET6` e `AF_UNIX`); *ii*) um tipo (*e.g.*, `SOCK_STREAM`, `SOCK_DGRAM` e `SOCK_RAW`), e *iii*) uma família de protocolos (ou *Protocol Family* - PF), como `PF_INET`, `PF_INET6`, `PF_UNIX`, entre outros. Geralmente, existe apenas uma única PF compatível com cada AF. No entanto, há casos em que uma única AF possui múltiplas PFs correspondentes.

Netlink [Neira-Ayuso et al. 2010, Kernel 2025a] é uma família de *sockets* (AF) implementada no Linux, a chamada `AF_NETLINK`. Através de *sockets* Netlink, processos no espaço de usuário podem se comunicar com módulos do *kernel* Linux que implementem esta interface, e vice-versa. Na versão atual do *kernel* Linux (6.14), existem 23 PFs registradas para a `AF_NETLINK`. Entre as PFs mais conhecidas, é possível citar a `NETLINK_NETFILTER`, usada pela ferramenta `iptables` para acessar o Netfilter; e a `NETLINK_SELINUX`, que transmite eventos de notificação do SELinux (*Security-Enhanced Linux*). No entanto, é possível ainda utilizar uma família de *socket* genérica, conhecida como Generic Netlink (ou `NETLINK_GENERIC`), para criar outras PFs.

Um módulo do *kernel* Linux pode definir uma ou mais famílias Generic Netlink

(*i.e.*, PFs), cada uma para se referir a um tipo de funcionalidade do módulo. Assim, quaisquer mensagens encaminhadas através de um *socket* criado para uma família Generic Netlink é entregue ao módulo para qual a família foi definida (sem a necessidade de um endereço de destino – como é realizado em IP, por exemplo). Para cada uma dessas famílias é preciso ainda definir um conjunto de comandos (*i.e*, operações a serem disponibilizadas) e atributos para serem informados ao executar cada comando. O Open vSwitch define um conjunto de 6 famílias Generic Netlink, as quais são descritas na Tabela 1.

Tabela 1. Famílias Generic Netlink do módulo de *kernel* do Open vSwitch.

Nome	Descrição
`ovs_datapath`	Utilizada para gerenciar *datapaths*
`ovs_vport`	Utilizada para gerenciar portas virtuais do *switch*
`ovs_flow`	Utilizada para gerenciar a tabela de fluxos
`ovs_packet`	Utilizada para notificações relacionadas a pacotes
`ovs_meter`	Utilizada para gerenciar medições de fluxos de pacotes
`ovs_ct_limit`	Utilizada para limitar as entradas na tabela de *conntracks*

Tabela 2. Comandos do módulo de *kernel* do Open vSwitch.

	Comando	ID	Descrição
`ovs_datapath`	`OVS_DP_CMD_NEW`	1	Adicionar um *datapath*
	`OVS_DP_CMD_DEL`	2	Remover um *datapath*
	`OVS_DP_CMD_GET`	3	Listar *datapaths*
	`OVS_DP_CMD_SET`	4	Modificar um *datapath*
`ovs_vport`	`OVS_VPORT_CMD_NEW`	1	Adicionar uma porta virtual
	`OVS_VPORT_CMD_DEL`	2	Remover uma porta virtual
	`OVS_VPORT_CMD_GET`	3	Listar portas virtuais
	`OVS_VPORT_CMD_SET`	4	Modificar uma porta virtual
`ovs_flow`	`OVS_FLOW_CMD_NEW`	1	Adicionar uma regra de fluxo
	`OVS_FLOW_CMD_DEL`	2	Remover uma regra de fluxo
	`OVS_FLOW_CMD_GET`	3	Listar regras de fluxo
	`OVS_FLOW_CMD_SET`	4	Modificar uma regra de fluxo
`ovs_packet`	`OVS_PACKET_CMD_MISS`	1	Notificar ausência de regra p/ um pacote
	`OVS_PACKET_CMD_ACTION`	2	Notificar ações aplicadas a um pacote
	`OVS_PACKET_CMD_EXECUTE`	3	Aplicar ações a um pacote
`ovs_meter`	`OVS_METER_CMD_FEATURES`	1	Listar funcionalidades suportadas
	`OVS_METER_CMD_SET`	2	Adicionar ou modificar um medidor
	`OVS_METER_CMD_DEL`	3	Remover um medidor
	`OVS_METER_CMD_GET`	4	Obter estatísticas do medidor
`ovs_ct_limit`	`OVS_CT_LIMI_CMD_SET`	1	Criar/modificar um limite de *conntrack*
	`OVS_CT_LIMI_CMD_DEL`	2	Remover um limite de *conntrack*
	`OVS_CT_LIMI_CMD_GET`	3	Listar limites de *conntrack*

Além disso, a implementação do módulo de *kernel* do Open vSwitch define um conjunto de comandos para cada Família Generic Netlink, descritos na Tabela 2. Por fim, são definidos diversos atributos que podem ser passados como parâmetros aos comandos. Contudo, a lista de todos os atributos existentes é muito extensa para ser apresentada neste documento. A título de exemplo, a Tabela 3 mostra os 4 primeiros atributos definidos para a família Generic Netlink `ovs_flow`, a qual permite gerenciar a tabela de fluxos através da classificação de pacotes. Alguns destes atributos são de um tipo primitivo, como é o caso do atributo `OVS_FLOW_ATTR_TCP_FLAGS`. Já outros atributos são estruturas de dados (e possuem atributos aninhados), como os atributos de ID 1, 2 e 3 da Tabela 3.

Através desses atributos (e outros não apresentados na tabela), é possível caracterizar o tipo de pacote a ser tratado em uma regra, bem como quais ações devem ser tomadas. Por exemplo, uma regra de encaminhamento de pacotes (camada 2) pode definir algo como: um pacote que entrou pelo *switch* na porta virtual 15, com o endereço Ethernet de destino 02:42:07:5f:22:2c, deve ser encaminhado para a porta 2 do *switch*. Além disso, é possível ainda definir regras de roteamento (camada 3), tunelamento, entre outras.

Tabela 3. Exemplos de atributos para a classificação de fluxos de pacotes.

Atributo	ID	Tipo	Descrição
`OVS_FLOW_ATTR_KEY`	1	struct	Uma “chave”(regra) que caracteriza o fluxo de pacotes
`OVS_FLOW_ATTR_ACTIONS`	2	struct	Ações a serem tomadas
`OVS_FLOW_ATTR_STATS`	3	struct	Estatísticas do fluxo de pacotes
`OVS_FLOW_ATTR_TCP_FLAGS`	4	uint8	O valor OR de todos os sinalizadores TCP vistos nos pacotes do fluxo
`OVS_FLOW_ATTR_USED`	5	uint64	Há quanto tempo (ms) que um pacote foi processado pela última vez em um determinado fluxo

3. Trabalhos Relacionados

Nesta seção são apresentados os esforços de pesquisa mais relevantes no contexto deste trabalho. Neste sentido, são abordados tanto trabalhos focados no Open vSwitch, quanto na avaliação da robustez de sistemas de software.

Dada sua relevância na indústria e academia, o Open vSwitch tem sido objeto de estudo de diversos trabalhos. Grande parte dos trabalhos encontrados na literatura atual tem focado principalmente em seu desempenho. Isto inclui, por exemplo, trabalhos que realizam o aperfeiçoamento [Yan and Wang 2016, Tseng et al. 2017, Jackson et al. 2016], a validação [Shanmugalingam et al. 2016], bem como a avaliação [Sans and Gamess 2013] de desempenho do Open vSwitch. Nesses trabalhos, geralmente é considerado o uso do DPDK (*Data Plane Development Kit*), aceleração via processamento em GPU e, até mesmo, uma nova arquitetura.

A avaliação da robustez de sistemas de software é um tema relevante e abordado em diversos contextos [Laranjeiro et al. 2021]. Em um trabalho pioneiro foi proposto o Ballista [Koopman and DeVale 1999], um método que combina valores aceitáveis e excepcionais para gerar testes para funções individuais de um sistema (como chamadas de

sistema, por exemplo) – este foi utilizado para avaliar a robustez de 13 sistemas operacionais POSIX nos anos 90. Em [Arlat et al. 2003] foi proposta a ferramenta MAFALDA (*Microkernel Assessment by Fault injection AnaLysis and Design Aid*), a qual permite a caracterização do comportamento de *microkernels* na presença de falhas.

Na literatura é possível encontrar ainda trabalhos que avaliam a robustez de sistemas embarcados [Ait-Ameur et al. 2003], sistemas de comunicação [Cavalli et al. 2008], serviços Web [Laranjeiro et al. 2008a], sistemas autônomos [Hutchison et al. 2018] e adaptativos [Cámara et al. 2015], além de vários outros. Já no contexto de SDN, há ainda trabalhos voltados tanto ao projeto, quanto a avaliação da robustez de controladores SDN ao considerar diferentes falhas de componentes do plano de controle [Scott-Hayward 2015, Ruchel et al. 2022].

Contudo, até onde se sabe, não há quaisquer trabalhos que avaliem a robustez do Open vSwitch. O presente trabalho visa preencher esta lacuna no atual estado da arte ao propor uma abordagem para avaliar a robustez de um componente fundamental do Open vSwitch, o módulo de *kernel*. Além disso, este é o primeiro trabalho a avaliar a robustez de um sistema através de uma interface Netlink.

4. Avaliação da Robustez do Módulo de *Kernel* do Open vSwitch

O método para avaliação de robustez proposto é baseado uma técnica de injeção de falhas que utiliza entradas inválidas sistematicamente selecionadas, considerando características da comunicação via Netlink. Em síntese, os testes de robustez propostos consistem em gerar mensagens Netlink (válidas e inválidas), encaminhá-las ao módulo de *kernel* do Open vSwitch e observar seu comportamento a fim de detectar eventuais falhas. Para isso, é levado em conta o cenário ilustrado na Figura 2.

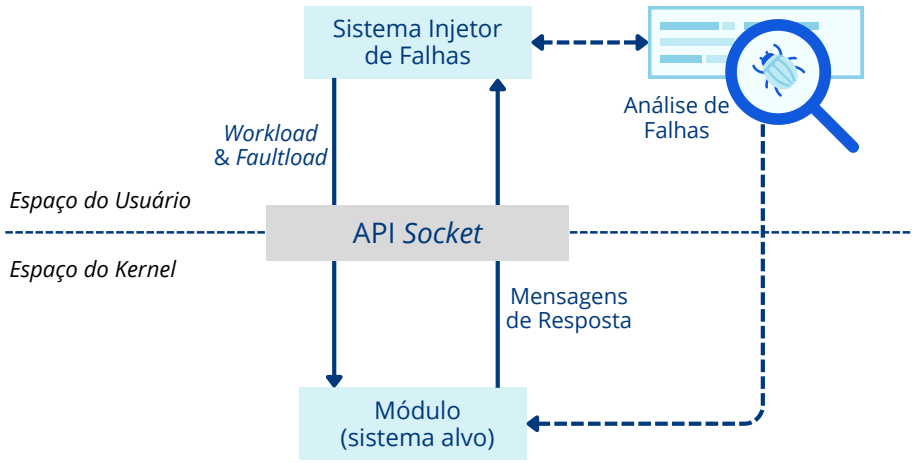

Figura 2. Visão geral do método proposto.

Neste cenário, o teste de robustez é realizado por um sistema injetor de falhas que consiste em um processo em execução no espaço de usuário. Este sistema injetor de falhas é responsável por gerar tanto a *workload*, quanto a *faultload* e encaminhá-las para o módulo de *kernel* do Open vSwitch através de um *socket* Netlink. O módulo processa as entradas e retorna uma saída (seja ela correta ou não) sempre que não ocorra uma falha que o impeça disso. As eventuais falhas ocorridas são identificadas a partir das saídas retornadas (ou a ausência delas) e um monitoramento do módulo.

4.1. Perfil de Teste de Robustez

Uma parte fundamental de um teste de robustez é a abordagem utilizada para gerar os conjuntos de entradas a serem submetidos ao sistema alvo, bem como a forma em que o teste será conduzido. Em outras palavras, é preciso definir um perfil de teste de robustez. Neste sentido, inspirado em um trabalho encontrado na literatura [Laranjeiro et al. 2008b], é proposta uma abordagem que consiste em três fases de teste, ilustradas na Figura 3.

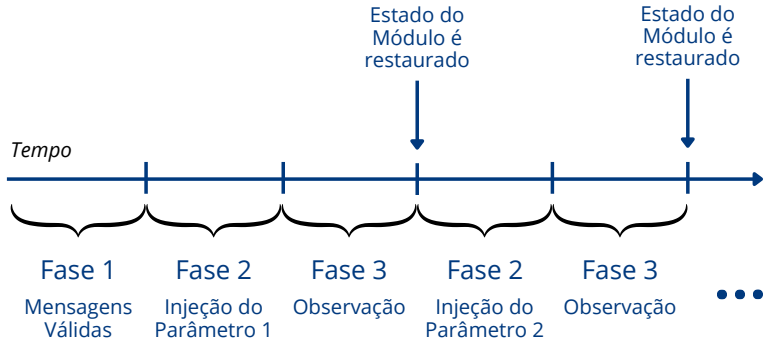

Figura 3. Perfil de teste de robustez.

Primeiramente, uma família Generic Netlink é selecionada para dar-se início à execução das fases. A Fase 1 é executada apenas uma única vez, já as fases 2 e 3 são repetidas de forma cíclica por várias vezes, uma para cada atributo Generic Netlink da família selecionada. Na Fase 1, é gerado um conjunto de mensagens com valores válidos, de modo a compor a *workload*. Essas mensagens são então enviadas ao módulo de *kernel* do Open vSwitch com o objetivo de compreender seu comportamento sem a ocorrência de falhas (propositais).

Tabela 4. Regras para gerar valores inválidos para atributos Generic Netlink.

Tipo	Nome da Regra	Descrição
String	StrNull	Atribuir um valor nulo (*i.e., Null*)
	StrEmpty	Atribuir uma string vazia
	StrNonPrintable	Atribuir uma string com caracteres não imprimíveis
	StrAlphanumeric	Atribuir uma string alfanumérica
	StrOverflow	Atribuir caracteres que excedam o tamanho máximo
Número	NumNull	Atribuir um valor nulo (*i.e., Null*)
	NumMinType	Atribuir o número mínimo válido para o tipo do dado
	NumMaxType	Atribuir o número máximo válido para o tipo do dado
	NumUnderflow	Atribuir um número negativo que extrapole o tipo do dado
	NumOverflow	Atribuir um número positivo que extrapole o tipo do dado
Estrutura	StructNull	Atribuir um valor nulo (*i.e., Null*)
	StructPrimitive	Atribuir um valor primitivo (Boolean, Int, Float, Double, *etc.*)
	StructCommon	Atribuir uma estrutura de dados comum (List, Map e Date)

Na Fase 2, *um* atributo é escolhido a partir de uma lista de atributos ainda não testados da família. De acordo com o tipo do atributo, são gerados todos os possíveis

valores inválidos seguindo as regras propostas na Tabelas 4. Estas regras exploram os limites dos tipos de dados dos atributos Netlink afim de desencadear falhas no módulo testado. Os valores gerados são usados para construir as mensagens que constituem a *faultload*. Cada mensagem deste conjunto é, portanto, um caso de teste. Neste momento, as mensagens com valores inválidos são então enviadas ao módulo de *kernel* do Open vSwitch.

Logo após, o módulo é revinculado ao *kernel* do Linux (*i.e.*, desabilidado e habilitado), fazendo com que seu estado seja restaurado. Isto evita que uma falha ocorrida ao testar um atributo afete o teste do próximo. Por fim, é realizada uma observação na Fase 3, visando identificar as possíveis falhas. As fases 2 e 3 são então repetidas para cada atributo da Família Gereric Netlink selecionada. Ao testar todos os atributos da família atual, a próxima família é selecionada e o processo se repete até que o conjunto completo de atributos de todas as famílias Generic Netlink do módulo tenha sido testado.

4.2. Modos de Falha

Ao se conduzir testes de robustez é preciso distinguir falhas e comportamentos corretos do sistema. Outro ponto fundamental é compreender a gravidade das falhas reveladas durante os testes. Neste sentido, foi proposta uma adaptação da escala CRASH [Koopman and DeVale 1999] para o contexto de interação com o módulo de *kernel* do Open vSwitch via Netlink. Assim, foram definidos os seguintes modos de falha (*i.e.*, *failure modes*):

- ***C**atastrophic*: o módulo não responde a novos estímulos e precisa ser revinculado ao *kernel* (*i.e.*, desabilidado e habilitado) para voltar a operar corretamente, ou a falha no módulo afeta todo o *kernel* e o sistema operacional falha por completo;
- ***R**estart*: uma requisição ao módulo nunca retorna uma saída, resultando em uma tarefa travada que precisa ser terminada/reiniciada à força;
- ***A**bort*: o envio da *workload* ou *faultload* é interrompido abruptamente;
- ***S**ilent*: nenhum erro ou código de erro é retornado pelo módulo testado, quando deveria ter sido;
- ***H**indering*: um código de erro incorreto é retornado pelo módulo testado.

Um comportamento correto ocorre quando o sistema sob teste reage de acordo com sua especificação. Neste caso, a geração de um código de erro apropriado ao receber uma entrada inválida, é considerado um comportamento correto do módulo.

4.3. Método de Análise

Em testes de robustez de sistemas de software a especificação do sistema é, geralmente, a base para se caracterizar falhas (*i.e.*, diferenciar comportamentos corretos de incorretos). A documentação atual do *kernel* Linux [Kernel 2025b], no entanto, não especifica todas as famílias Generic Netlink implementadas. No caso do Open vSwitch, apenas as famílias `ovs_datapath`, `ovs_vport` e `ovs_flow` são documentadas. Além disso, a documentação existente não especifica explicitamente quais são as entradas aceitas para cada atributo (apenas o tipo dos atributos). Isso representa um desafio ao determinar se o comportamento do módulo está correto diante a uma entrada e, portanto, dificulta a identificação de falhas.

Neste sentido, é proposto um método de análise orientado à especificação dos códigos de erro do sistema. O objetivo é permitir caracterização de falhas do sistema mesmo sem a existência de uma especificação a respeito do conjunto de entradas aceito por cada atributo. Para isso, é preciso que o sistema sob teste implemente um conjunto padronizado de códigos de erros. No caso do *kernel* Linux (e seus módulos), é adotado um sistema de código de erros definido pelo Padrão IEEE 1003.1-2001 (também conhecido como POSIX.1-2001) [IEEE 2001].

O método consiste em, diante a uma entrada gerada a partir de uma das regras definidas na Tabela 4, comparar o código de erro dado como saída com a especificação de códigos de erro do sistema. Se o código de erro retornado for apropriado à entrada submetida, de acordo com a especificação, o comportamento do sistema é considerado correto. Caso nenhum código de erro seja retornado ou o erro não seja apropriado, o comportamento é considerado uma falha do sistema que deve ser categorizada de acordo com a escala CRASH. Outro caso possível é o retorno de um código de sucesso. Contudo, esta deve ser uma situação mais rara se a validação de atributos de entrada do módulo for eficaz, já que um dos atributos da requisição sempre possui um valor que objetiva ser inválido. Quando isso ocorre, é verificado se a operação requisitada foi de fato executada corretamente (*e.g.*, se um comando de remover realmente removeu o que deveria, sem afetar outros registros e funcionalidades). Se a operação não foi bem-sucedida, o caso é considerado uma falha.

Por fim, é realizada ainda uma análise da consistência das saídas do sistema. Por exemplo, considere que ao requisitar a execução de um determinado comando submetendo um atributo x com um valor y o sistema retorna um erro indicando que o valor do atributo é inválido. Neste caso, uma inconsistência é identificada se o mesmo atributo e valor forem considerados válidos na execução de outro comando.

5. Resultados

Através da abordagem proposta, foi conduzida uma avaliação do módulo de *kernel* do Open vSwitch. O sistema injetor de falhas foi implementado em Python através da biblioteca pyroute2[1], a qual serve como base para a comunicação Netlink. Por meio deste, foram executados testes em todas as 6 famílias Generic Netlink implementadas pelo Open vSwitch. No total, foram testados cada um dos 31 atributos suportados em requisições de todos os 19 comandos que permitem solicitar operações ao módulo do *kernel*. A avaliação foi realizada em um sistema operacional Ubuntu 24.10 com a versão 6.11.0-19 do *kernel* Linux. A seguir é apresentada a análise de falhas e inconsistências encontradas através dos testes de robustez.

5.1. Análise de Falhas

Uma parte considerável da *faultload* submetida ao módulo de *kernel* do Open vSwitch foi tratada de forma correta. Em diversos casos de teste a saída retornada foi o erro `EINVAL` (*Invalid argument* - código 22). Este é um código de erro apropriado, já que o sistema indica que o valor submetido é inválido e não prossegue com a execução do comando requisitado. Além disso, há casos também considerados corretos em que (apesar da requisição conter um atributo com valor inválido) o módulo do *kernel* executa o comando de forma bem-sucedida e retorna um código de sucesso.

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

No entanto, diversas falhas foram descobertas a partir dos testes realizados, como pode ser visto na Tabela 5. A família com o maior percentual de casos de teste que geraram falhas foi a `ovs_meter`, alcançando 51,61 % dos casos. Já na família `ovs_ct_limit`, 100 % dos casos de teste resultaram em comportamento normal. Porém, enquanto a família `ovs_ct_limit` possui apenas 1 único atributo e 3 comandos, a família `ovs_meter` possui 5 atributos aceitos em 4 comandos. Essa diferença faz com que o número de casos de testes gerados seja significativamente maior para a família `ovs_meter` do que para a `ovs_ct_limit` – o que influencia este resultado. O maior número de falhas foi encontrado na família `ovs_flow`, totalizando 32 falhas.

Tabela 5. Resultados da avaliação da robustez do módulo de *kernel* do Open vSwitch via Netlink.

Família Generic Netlink	Casos de Teste		Falhas				
	Comportamento Esperado	Falha	C	R	A	S	H
`ovs_datapath`	70,37 %	29,63 %	-	-	-	-	16
`ovs_vport`	62,16 %	37,84 %	-	-	-	-	14
`ovs_flow`	66,32 %	33,68 %	-	-	-	1	31
`ovs_packet`	62,50 %	37,50 %	-	-	-	-	9
`ovs_meter`	48,39 %	51,61 %	-	-	-	4	12
`ovs_ct_limit`	100 %	0 %	-	-	-	-	-

Foram encontradas apenas falhas *silent* (S) e *hindering* (H), sendo H a mais comum. Este resultado destaca a capacidade do *kernel* em evitar falhas que causem sua interrupção parcial (R) ou total (C). Entre as falhas do tipo H (*i.e.*, o retorno de um código de erro incorreto), a de maior recorrência é o erro `ERANGE` (*Numerical result out of range* - código 34). De acordo com o Padrão POSIX.1-2001, este erro é retornado quando o resultado de uma função é muito grande (*overflow*) ou muito pequeno (*underflow*) para ser representado no espaço disponível. Portanto, a ocorrência deste erro significa que o valor inválido não foi apropriadamente validado pelo módulo do Open vSwitch e, em algum ponto adiante, o *kernel* se deparou com um *overflow* ou *underflow* desencadeado pelo valor inválido. Um exemplo concreto desta situação ocorre com a família `ovs_flow` ao se executar o comando para adicionar uma regra de fluxo (`OVS_FLOW_CMD_NEW`) aplicando a regra StructNull no atributo `OVS_FLOW_ATTR_UFID` (um identificador único do fluxo).

Já na família `ovs_datapath` o erro `ERANGE` também é retornado ao aplicar a regra StructNull no atributo `OVS_DP_ATTR_UPCALL_PID` e comando `OVS_DP_CMD_NEW`. O atributo `OVS_DP_ATTR_UPCALL_PID` se refere ao ID de processo para o qual o módulo do *kernel* deve encaminhar eventuais *upcalls* (*i.e.*, mensagens do *kernel* em direção ao espaço do usuário) relacionadas ao *datapath*. O maior número válido para um ID de processo no Linux é 4.194.304, o qual é limitado através da macro `PID_MAX_LIMIT`. Portanto, este atributo deveria ser validado de modo a aceitar apenas valores entre 0 (que significa não enviar *upcalls*) e `PID_MAX_LIMIT`. Quaisquer valores diferentes disso (incluindo um valor nulo) deveria resultar em um erro `EINVAL` ao invés de `ERANGE` – motivo pelo qual este caso também é considerado uma falha H.

Ainda na Tabela 5 é possível notar falhas do tipo S. Entre as 5 falhas S reportadas, 4 ocorreram na família `ovs_meter` com o atributo `OVS_METER_ATTR_ID`,

ao executar comandos OVS_METER_CMD_DEL. Mais especificamente, ao se aplicar as regras NumMinType, NumMaxType, NumUnderflow e NumOverflow no atributo OVS_METER_ATTR_ID, o módulo retornou um código de sucesso mesmo não removendo quaisquer *meters*, já que durante os testes não haviam correspondentes às IDs utilizadas. O resultado ideal, no entanto, deveria ser o código EINVAL. Contudo, o mínimo aceitável seria um erro como ENOENT (*No such file or directory* - código 2).

Outra falha S acontece ao se criar uma regra de fluxo aplicando a regra StructPrimitive no atributo OVS_FLOW_ATTR_UFID, ao invés de uma ID corretamente estruturada. A saída retornada é um código de sucesso, sugerindo que não houve qualquer erro na operação. No entanto, não é possível obter o a regra de fluxo criada através desta requisição. Inclusive, isso impede até mesmo a listagem de todas as regras de fluxo através da ferramenta de linha de comando (chamada ovs-dpctl), gerando um erro *"Failed to dump flows from datapath (Invalid argument)"*.

A Tabela 6 apresenta os atributos e comandos de cada família Generic Netlink do Open vSwitch em que se observou falhas. Nas famílias ovs_flow e ovs_packet, todos os comandos que permitem requisitar operações ao módulo apresentaram falhas.

Tabela 6. Atributos e comandos nos quais ocorreram falhas.

Família Generic Netlink	Atributos	Comandos
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
ovs_ct_limit	-	-

5.2. Análise de Inconsistências

Além das falhas encontradas, a avaliação de robustez revelou inconsistências no modo de validação dos atributos e códigos de erro retornados. Um exemplo disso ocorre com o atributo OVS_DP_ATTR_NAME da família ovs_datapath. Enquanto o comando para criar *datapaths* (OVS_DP_CMD_NEW) retorna o erro EINVAL para valores gerados com StrEmpty e StrNonPrintable, os comandos para listar, alterar e remover *datapaths* retornam o erro ENODEV (*No such device* - código 19) ao receber os mesmos valores. Em

outras palavras, estes comandos não validam corretamente o atributo e, de fato, processam os valores inválidos a ponto de buscá-los nos registros e informar que não existem.

Outra inconsistência similar é apresentada no comando `OVS_VPORT_CMD_NEW` da família `ovs_vport`. Este comando adiciona uma nova porta no *datapath*, associando a ela uma interface virtual previamente criada. Ao se executar o comando `OVS_VPORT_CMD_NEW` informando o atributo `OVS_VPORT_ATTR_NAME` com caracteres especiais (StrNonPrintable) é obtido o erro `ENODEV` – indicando que foi realizada uma busca pelo nome informado, mas nenhuma interface foi encontrada. No entanto, tal interface jamais existiria, pois não é possível atribuir um nome com caracteres especiais a uma interface virtual do Linux – ao se tentar fazer isso pela linha de comando é obtido o erro `EPERM` (*Operation not permitted* - código 1). Apesar desta inconsistência, este caso não foi considerado uma falha, já que de fato não existia uma interface com o nome requisitado durante o teste (como indica o erro `ENODEV`).

Uma situação relevante também ocorre com o atributo `OVS_METER_ATTR_ID` da família `ovs_meter` (que representa um identificador de uma *meter*). A ausência de uma validação apropriada deste atributo permite, por exemplo, a atribuição de um ID 0 (zero) ou, até mesmo, negativo a uma *meter*. Isso pode vir a desencadear falhas em outros componentes do sistema ao se depararem com um ID com esses valores.

6. Conclusão

Este trabalho propõe uma abordagem para a avaliação da robustez do Open vSwitch, em especial, seu módulo do *kernel* Linux. Concretamente, é empregada uma técnica de injeção de falhas na qual valores de entrada inválidos são sistematicamente selecionados e submetidos à interface Netlink implementada pelo módulo. Para isso é definido um perfil de testes de robustez que estabelece uma sequência de fases de teste e um conjunto de regras para geração de valores de entrada inválidos. Estas regras visam explorar limites dos tipos de dados dos atributos Netlink afim de desencadear falhas no módulo testado. Através da execução dos casos de testes gerados foram identificadas diversas falhas. Por meio de uma adaptação da escala CRASH para o contexto de comunicação Netlink e uma análise dos resultados orientada à especificação de códigos de erro do sistema, essas falhas foram devidamente caracterizadas. A avaliação conduzida revelou falhas em 5 das 6 famílias Generic Netlink do Open vSwitch, todas do tipo *hindering* e *silent*. Foram identificadas ainda inconsistências na validação de atributos, com um mesmo par de atributo e valor inválido sendo aceito por determinado comando e não permitindo por outros comandos da mesma família.

Trabalhos futuros incluem uma avaliação dos possíveis efeitos das falhas nas funcionalidades fundamentais do sistema, *i.e.*, no processamento de pacotes. Uma avaliação da robustez diante a um ambiente estressante (como um grande volume de requisições e até mesmo de tráfegos de rede) também é relevante. Por fim, está prevista ainda uma investigação sobre as possíveis vulnerabilidades que podem ser exploradas a partir das falhas e inconsistências reveladas, bem como meios para mitigá-las eficientemente.

Agradecimentos

Este trabalho foi parcialmente apoiado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Programa de Excelência Acadêmica (PROEX) – Código

de Financiamento 001; e o CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) - projeto 308959/2020-5.

Referências

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

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

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

Cámara, J., De Lemos, R., Laranjeiro, N., Ventura, R., and Vieira, M. (2015). Robustness-driven resilience evaluation of self-adaptive software systems. *IEEE Transactions on Dependable and Secure Computing*, 14(1):50–64.

Cavalli, A., Martins, E., and Morais, A. (2008). 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)*, pages 21–30.

Foundation, T. L. (2025). Open vSwitch. https://www.openvswitch.org/. Acessado em abril de 2025.

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

Hutchison, C., Zizyte, M., Lanigan, P. E., Guttendorf, D., Wagner, M., Le Goues, C., and Koopman, P. (2018). 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)*, pages 276–285.

IEEE (2001). IEEE Standard for IEEE Information Technology - Portable Operating System Interface (POSIX). Padrão 1003.1-2001, IEEE.

IEEE (2017). Systems and Software Engineering–Vocabulary. Padrão 24765:2017, IEEE.

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

Kernel (2025a). Introduction to Netlink – The Linux Kernel Documentation. https://docs.kernel.org/userspace-api/netlink/intro.html. Acessado em abril de 2025.

Kernel (2025b). Netlink Family Specifications. https://docs.kernel.org/networking/netlink_spec. Acessado em abril de 2025.

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

Laranjeiro, N., Agnelo, J., and Bernardino, J. (2021). A systematic review on software robustness assessment. *ACM Computing Surveys (CSUR)*, 54(4):1–65.

Laranjeiro, N., Canelas, S., and Vieira, M. (2008a). wsrbench: An On-Line Tool for Robustness Benchmarking. In *Proceedings of the 5th IEEE International Conference on Services Computing (SCC'2008)*, volume 2, pages 187–194. IEEE.

Laranjeiro, N., Vieira, M., and Madeira, H. (2008b). Experimental Robustness Evaluation of JMS Middleware. In *Proceedings of the 5th IEEE International Conference on Services Computing (SCC'2008)*, volume 1, pages 119–126. IEEE.

Neira-Ayuso, P., Gasca, R. M., and Lefevre, L. (2010). Communicating between the kernel and user-space in Linux using Netlink sockets. *Software: Practice and Experience*, 40(9):797–810.

Pfaff, B. and Davie, B. (2013). The Open vSwitch Database Management Protocol. RFC 7047.

Pfaff, B. et al. (2015). The Design and Implementation of Open vSwitch. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'2015)*, pages 117–130. USENIX.

Ruchel, L. V., Turchetti, R. C., and de Camargo, E. T. (2022). Evaluation of the robustness of sdn controllers onos and odl. *Computer Networks*, 219:109403.

Sans, F. and Gamess, E. (2013). Analytical performance evaluation of different switch solutions. *Journal of Computer Networks and Communications*, 2013(1):953797.

Scott-Hayward, S. (2015). Design and deployment of secure, robust, and resilient sdn controllers. In *Proceedings of the 2015 1st IEEE conference on network Softwarization (NetSoft)*, pages 1–5. IEEE.

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

Tseng, J., Wang, R., Tsai, J., Wang, Y., and Tai, T.-Y. C. (2017). Accelerating open vswitch with integrated gpu. In *Proceedings of the Workshop on Kernel-Bypass Networks*, pages 7–12.

Turner, J. S. and Taylor, D. E. (2005). Diversifying the Internet. In *Proceedings of the 17th IEEE Global Telecommunications Conference (GLOBECOM'2005)*, volume 2, pages 6–pp. IEEE.

Venâncio, G., Fulber-Garcia, V., Flauzino, J., Alchieri, E. A., and Duarte, E. P. (2024). 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)*, pages 1–6. IEEE.

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