Uma Avaliação sobre a Tolerância a Falhas no Plano de Dados em Controladores SDN

Lucas V. Ruchel[1,2], **Rogério C. Turchetti**[4], **Edson T. de Camargo**[3,1]

[1]Universidade Estadual do Oeste do Paraná (Unioeste) – Cascavel – PR – Brasil

[2]Instituto Federal do Paraná (IFPR) – Cascavel – PR – Brasil

[3]Universidade Tecnológica Federal do Paraná (UTFPR) – Toledo – PR – Brasil

[4]Universidade Federal de Santa Maria (UFSM) – Santa Maria – RS – Brasil

lukasruchel@gmail.com, edson@utfpr.edu.br, turchetti@redes.ufsm.br

Resumo. *Redes Definidas por Software fazem do controlador o seu elemento central. Portanto, é fundamental que o controlador apresente funcionalidades que, além de melhorar o desempenho global da rede, assegurem a continuidade dos serviços perante falhas. OpenDayLight (ODL) e o Open Network Operating System (ONOS) são dois importantes controladores de código aberto e com plano de controle distribuído. Este trabalho avalia a tolerância a falhas no plano de dados dos controladores ODL e ONOS considerando diversos cenários de falhas. Os experimentos consideram falhas e recuperações de links e switches, com diferentes fluxos tanto no modo reativo quanto proativo. Os resultados experimentais exibem o tempo que os controladores levam para detectar as falhas, direcionar o tráfego para caminhos redundantes e identificar o retorno dos links e switches.*

Abstract. *Software-Defined Networking relies on the controller as its main element. Therefore, it is critical for the controller presenting features that improve the overall network performance while ensuring continuity of service despite failures. OpenDayLight (ODL) and Open Network Operating System (ONOS) are two important open-source and distributed controllers. The objective of this paper is to evaluate how ODL and ONOS deal with different failures scenarios in their data plane. The evaluation analyzes both link and switches failures and recoveries, with different flows in reactive and proactive mode. Experimental results show the time the controllers take to circumvent and detect faults, forward traffic to redundant paths, and identify the return of links and switches.*

1. Introdução

Redes definidas por software (*Software Defined-Network* – SDN) é um paradigma emergente, empregado em diversos ambientes para a comunicação efetiva em redes de computadores: computação em nuvem, redes wireless, internet das coisas, telefonia para celular, datacenters, entre outros [Bannour et al. 2018]. SDN versa sobre o desacoplamento dos elementos de rede para torná-la mais flexível. A arquitetura SDN é composta por 3 camadas: aplicação, plano de controle e plano de dados. A camada de aplicação hospeda as aplicações projetadas para a rede. Sua comunicação com os elementos do plano de dados, como comutadores (*switches*) e roteadores é intermediada pelo plano de controle cuja

responsabilidade é gerenciar o plano de dados. No plano de dados estão os dispositivos responsáveis por implementar os fluxos de encaminhamento de pacotes.

Em uma rede SDN o controlador executa um papel fundamental. Por este motivo, pode-se observar diversos esforços concentrados no controlador no sentido de avaliar seu desempenho [Zulu et al. 2018], torná-lo distribuído [Kohler et al. 2018], escalável [Bianco et al. 2017, Karakus and Durresi 2017], confiável [Curic et al. 2018, Venâncio et al. 2018], seguro e tolerante a falhas [Gonzalez et al. 2016]. Estes atributos foram ou estão sendo implementados nos diversos controladores disponíveis para uso.

Parte importante no processo para a definição de qual controlador utilizar, é realizar um estudo sobre o seu comportamento e desempenho diante de situações adversas. Nota-se uma carência de trabalhos práticos que analisam, com profundidade, aspectos de tolerância a falhas em controladores tradicionalmente empregados em redes SDN. Por exemplo, é importante para o operador de rede conhecer se o controlador consegue manter o funcionamento da rede mesmo havendo uma falha permanente de um *switch* ou na interrupção de um *link*. Além disso, o controlador precisa ser capaz de reagir rapidamente a alterações de topologia devido a falhas no plano de dados.

Em [Obadia et al. 2014] os autores propõem migrar o controle de *switches* órfãos para outros controladores que ainda estão ativos, de forma a reagir rapidamente a falhas do controlador e manter conectividade de toda a rede. Da Silva et al. [da Silva et al. 2015] apresentam uma pesquisa que fornece uma visão geral estruturada do suporte à resiliência em SDN. Os autores concluem que o plano de dados pode ser protegido contra interrupção do *link*, falhas de dispositivos e ataques maliciosos. Ambos trabalhos propõem ou citam soluções resilientes. Já o trabalho de Vilchez et al. [J. M. S. Vilchez 2018] avalia, de forma prática, a tolerância a falhas no plano de dados considerando o comportamento dos controladores OpenDayLight (ODL) e ONOS. ODL e ONOS são dois dos principais controladores SDN utilizados pela indústria, possuem características distribuídas e são projetos open-source [Darianian et al. 2017]. Apesar de um importante trabalho prático, os autores avaliam exclusivamente a interrupção e reestabelecimento de *links*.

Este artigo compara a tolerância a falhas no plano de dados dos controladores ODL e ONOS de forma mais abrangente que os demais trabalhos descritos na literatura. Além de falhas nos *links*, este trabalho analisa o comportamento dos controladores e sua reação diante de parada e retorno de *switches*. A avaliação também inclui o comportamento dos controladores perante falhas no plano de dados quando há fluxos TCP e UDP. Além disso, versões mais atuais dos controladores ODL e ONOS são avaliadas para acompanhar a evolução das funcionalidades. Resultados experimentais exibem o tempo que os controladores levam para detectar as falhas, direcionar o tráfego para caminhos redundantes e identificar o retorno dos *links* e dos *switches*. Há ainda uma comparação entre os resultados obtidos neste artigo e o trabalho de [J. M. S. Vilchez 2018].

Este trabalho segue organizado da seguinte forma. A Seção 2 apresenta os trabalhos encontrados na literatura que destacam os dois controladores. A Seção 3 descreve os controladores ODL e ONOS. A Seção 4 apresenta os cenários de comparação e os resultados obtidos. Por fim, a conclusão é apresentada na Seção 5.

2. Trabalhos Relacionados

Os autores em [Zhu et al. 2019] afirmam que controladores SDN são usados para gerenciar todas as operações do plano de dados representando um elemento fundamental na arquitetura. Neste sentido, conhecer o desempenho e a capacidade oferecida por um controlador é extremamente importante na visão do operador da rede. Os autores realizam uma análise quantitativa e comparativa entre diversos controladores, incluindo uma análise qualitativa de suas propriedades. Os experimentos são conduzidos considerando as seguintes métricas de comparação: latência, *throughput* e utilização de CPU. Os experimentos consideram tanto o desempenho dos controladores quanto das ferramentas de *benchmarks* frequentemente utilizadas para análises de desempenho. Os autores concluíram que os controladores Floodlight, OpenMul, Beacon, Maestro, ONOS e ODL, por serem *multi-thread*, possuem latência e *throughput* melhor do que os demais controladores testados. No entanto, ONOS e ODL se destacam por apresentarem o plano de controle distribuído. Apesar do trabalho fazer um profundo estudo comparativo entre controladores SDN, os autores não trataram sobre a questão de disponibilidade perante falhas, nem no contexto do plano de dados e tampouco no contexto do plano de controle.

Na mesma linha de [Zhu et al. 2019], uma avaliação experimental dos controladores é realizada em [Darianian et al. 2017], mas somente considerando o ONOS e ODL. Os autores construíram uma plataforma de testes e usaram a ferramenta Cbench para avaliar seu desempenho dos controladores de acordo com as seguintes métricas: taxa de transferência (*throughput*), latência e escalabilidade tanto em ambientes físicos quanto virtualizados. Os resultados experimentais mostraram que o ONOS fornece maior taxa de transferência e menor latência que o ODL, que sofre de problemas de desempenho em modelos de rede maiores.

O trabalho de [Bannour et al. 2018] realiza uma pesquisa e uma análise detalhada das plataformas de controladores SDN distribuídos. Os autores avaliam as vantagens e desvantagens dos controladores e oferecem duas formas de classificá-los: ONOS e ODL são classificados como fisicamente distribuídos, pois permitem distribuir o plano de controle, e logicamente centralizados porque apesar de sua arquitetura distribuída permitem uma visão centralizada da rede. Distribuir o plano de controle é uma solução potencial para mitigar os problemas trazidos pelas arquiteturas SDN centralizadas (baixa escalabilidade, ponto único de falha, gargalos de desempenho etc.). Os autores destacam que ambos controladores são semelhantes: mantêm uma visão consistente da rede, possuem notável escalabilidade e confiabilidade. Afirmam ainda que tanto o ONOS quanto o OpenDayLight representam outra categoria de soluções SDN logicamente centralizadas que se destacam das plataformas de controladores SDN distribuídos de última geração. [Bannour et al. 2018] fazem várias considerações sobre tolerância a falhas no plano de controle distribuído e destacam as características e o protocolo de recuperação presente no ONOS, porém não realizam uma comparação em termos de tolerância a falhas dos controladores.

Considerando a resiliência no plano de dados utilizando os controladores ODL e o ONOS, os autores em [J. M. S. Vilchez 2018] avaliaram a continuidade do serviço oferecido em três diferentes casos, considerando a comunicação fim a fim: (i) caminho único (ii) um caminho alternativo (iii) dois caminhos alternativos. Desta forma, os autores concluíram que o ODL falhou em garantir a continuidade da comunicação entre hosts lo-

calizados em *switches* distintos, mesmo havendo caminhos alternativos. Conforme apontado no trabalho, o ONOS superou diversos aspectos do ODL. Os autores atribuíram à implementação do módulo *l2switch* (provê funcionalidades da camada 2 ao controlador) a falha de identificar caminhos alternativos e o atraso para reestabelecimento do *link*. No trabalho apresentado foram considerados apenas fluxos UDP, não sendo considerado o comportamento de fluxos TCP. As versões dos controladores utilizados pelos autores receberam atualizações nas funcionalidades avaliadas, sendo necessário que novas análises sejam feitas.

ONOS e ODL são também avaliados experimentalmente em [T.BAH et al. 2019] quanto a descoberta e atualização de topologia no modo distribuído e centralizado. Uma mudança de topologia significa adicionar ou remover tanto um *link* quanto um nodo. Nesse sentido, os autores destacam a importância do protocolo LLDP (*Link Layer Discovery Protocol*) e OFDP (*OpenFlow Discovery Protocol*). O LLDP permite que um nó em uma rede local IEEE 802 anuncie seus recursos a seus vizinhos e o OFDP é o protocolo padrão de descoberta de topologia SDN. Nos testes realizados, o ODL obteve menor tempo de descoberta da topologia em relação ao ONOS. No entanto, considerando atualizações na topologia o ONOS obteve menor tempo de resposta. Apesar disso, os autores concluíram que o ONOS reage melhor quando ocorrem falhas de *links*, com tempo de reação próximo de 100ms. Destaca-se que a avaliação dos autores é realizada considerando fluxos TCP e em versões antigas dos controladores.

Considerando funcionalidades para tolerância a falhas em redes SDN, é possível observar diversos outros trabalhos que incluem a sincronização consistente no plano de controle distribuído. Estas funcionalidades se traduzem em diferentes algoritmos, tais como: Paxos [Ho et al. 2016, Venâncio et al. 2018], Replicação de Máquina de Estados [Canini et al. 2015, Mahajan et al. 2016], Zookeeper [Koponen et al. 2010], Raft [Medved et al. 2014], entre outros algoritmos frequentemente empregados no contexto de tolerância a falhas em sistemas distribuídos. Também são encontradas soluções que procuram garantir a execução consistente do plano de dados. Neste contexto, em geral, são verificadas soluções com mudanças mais intrusivas, como por exemplo a inclusão de novas funcionalidades implementadas diretamente no *firmware* dos *switches* SDN [Schiff et al. 2016, Dang et al. 2015, Katta et al. 2015].

Outros trabalhos propõem técnicas que procuram garantir a ordem dos eventos, tanto no plano de dados quanto no plano de controle. Por exemplo, as estratégias procuram garantir que as mensagens processadas nos *switches* e comandos executados nos controladores sejam encaminhadas de maneira ordenada e de forma a garantir uma sincronização consistente entre componentes que executam nos diferentes planos da rede [Katta et al. 2015]. Da Silva et al. [da Silva et al. 2015] realizam um estudo sobre o suporte à resiliência em redes SDN, ou seja, a capacidade da rede de manter um nível aceitável de serviço quando confrontada com desafios operacionais que envolvem: segurança, capacidade de sobrevivência (incluindo tolerância a falhas), desempenho, tolerância ao tráfego, tolerância a interrupções e confiabilidade. Entre as conclusões dos autores, após avaliarem diversas pesquisas, estão a de que o plano de dados pode ser protegido contra interrupções de *links*, falhas de dispositivos e ataques maliciosos usando aplicativos colocados nos planos de controle ou aplicação.

Por fim, vale ressaltar que este trabalho não propõe a inclusão de novas funciona-

lidades, mas procura explorar o desempenho e o comportamento das funcionalidades já implementadas e colocadas em produção no domínio dos controladores ONOS e ODL.

3. ONOS e OpenDayLight: características e propriedades.

O ONOS e o OpenDaylight (ODL) são controladores que se destacam em relação ao estado da arte de controladores distribuídos por serem de código aberto, bem como por possuírem as funcionalidades necessárias para o gerenciamento e operacionalização de uma rede. Ambos controladores são escritos em JAVA. No entanto, possuem diferenças em relação a características como a sua estrutura, público-alvo, áreas de atuação e funcionalidades [Bannour et al. 2018].

Ao contrário do ODL que possui fins mais gerais, o ONOS, coordenado pela *Open Networking Foundation* (ONF), possui um foco voltado para provedores de serviço, de tal forma que componentes estruturais foram desenvolvidos para atingir alta disponibilidade, escalabilidade e desempenho. Em sua arquitetura, além da *NorthBound API* e a *South-Bound* modular, o ONOS oferece dois principais mecanismos de sincronização de estados que utilizam troca de mensagens, permitindo que alcance a flexibilidade necessária para aplicações que necessitam alta disponibilidade e desempenho [Muqaddas et al. 2016]. A Tabela 1 apresenta, de forma resumida, as características do ONOS e ODL em relação a diversos aspectos, conforme apresentadas em [Darianian et al. 2017, Zhu et al. 2019].

Controladores	**ONOS**	**OpenDayLight**
Consistência fraca	sim	não
Consistência forte	sim	sim
Interface Gráfica	sim	sim
Usuários Finais	provedores de serviço	uso geral
Diferenças Arquiteturais	Intent Framework	MD-SAL
Foco	Explorar todas as capacidades do SDN em provedores de serviços	Integração com dispositivos legados
Número de versões	11	19
Multithread	Sim	Sim
Documentação	Boa	Boa
Modularidade	Boa	Boa
Linguagem de Programação	Java	Java

Tabela 1. Características ONOS e do OpenDayLight.

A sincronização de estados dos componentes da rede, no ONOS, facilita o gerenciamento e a coordenação de *clusters*, provendo mecanismos para lidar com diferentes tipos de estados no plano de controle. Conforme a Tabela 1, o ONOS oferece duas primitivas para a sincronização e controle de estados, incluindo *ConsistentMap* para estados que precisam consistência forte e a primitiva *EventuallyConsistentMap* para aplicações que necessitam desempenho e toleram a utilização de consistência fraca [Darianian et al. 2017].

O OpenDayLight é administrado pela *Linux Foundation* e mantido pela comunidade, é um controlador de uso geral, projetado para diversos domínios de aplicações

(datacenter, provedores de serviço, entre outros). Uma importante funcionalidade arquitetural do ODL é o *Model-Driven Service Abstraction Layer (MD-SAL)* que permite a fácil e flexível integração de serviços de rede, independente dos protocolos utilizados na *SouthBound* e a possibilidade de controlar dispositivos heterogêneos (SDN ou não). A sincronização de estados no plano de controle utiliza consistência forte [Bannour et al. 2018], provida pelo *framework* akka [1].

O principal foco do ODL foi acelerar a integração de dispositivos SDN em ambientes com redes legadas, permitindo que dispositivos não-SDN se comuniquem com dispositivos Openflow. De tal forma, o projeto adotou soluções proprietárias para interoperar com diversas marcas de dispositivos. Ao contrário do ODL, o ONOS foi projetado com o intuito de aproveitar todas as funcionalidades SDN em provedores de serviço.

Em SDN, fluxos podem ser instalados no *switches* de três modos diferentes. No modo reativo, ao identificar um novo fluxo o *switch* solicita ao controlador a decisão sobre a instalação do fluxo. Isso é feito através de uma mensagem *PACKET_IN* do OpenFlow. No modo proativo, um controlador previamente identifica a necessidade de instalar um fluxo no *switch*. O terceiro modo é chamado de híbrido pois é uma mistura das anteriores [Sinha et al. 2017]. O ONOS permite empregar um modo híbrido. Usa o modo reativo por padrão, mas possui o *Framework de Intents* [ONOS 2016] que permite que aplicações instalem proativamente políticas de rede. O ODL utiliza o modo proativo por padrão e através de seus arquivos de configuração também permite os três modos destacados acima. No entanto, no modo proativo o ODL, ao invés de encaminhar pacotes ao controlador, realiza o *flood* nas portas conectadas do *switch* [OpenDaylight 2019]. Neste trabalho, ONOS é avaliado somente no modo reativo, enquanto que o ODL é avaliado no modo proativo e reativo.

Na próxima seção, são apresentados os resultados experimentais que permitem descrever, na prática, o comportamento de cada controlador considerando diversos cenários de testes.

4. Resultados

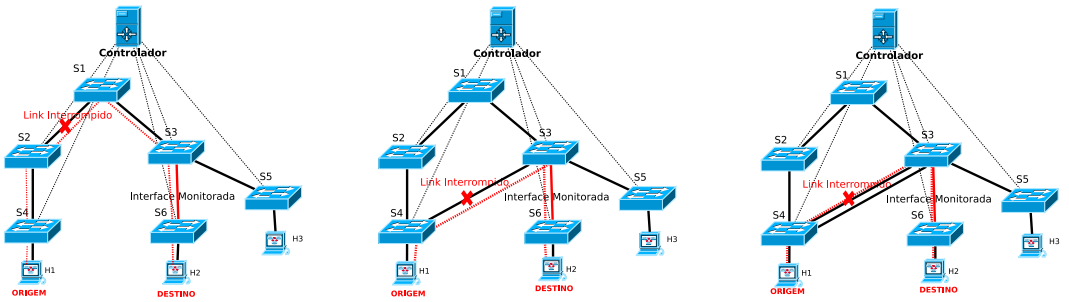
Nesta seção são apresentados experimentos executados para avaliar o comportamento e desempenho dos controladores ODL e ONOS diante de situações de falhas no plano de dados. Foram utilizadas duas máquinas com processador Intel Core i7-8700 de 3.20GHz, 16GB memória com o Ubuntu 18.04.3. A simulação da topologia foi realizada utilizando a ferramenta Mininet[2]. O tráfego entre os hosts da topologia foi gerado pela ferramenta *netperf*[3]. Os casos de testes avaliam o comportamento dos controladores diante de falhas no plano de dados e a continuidade dos serviços executados com a utilização de fluxos TCP e UDP. Como critério de avaliação, considerou-se o tempo para reestabelecer a conexão após a falha de um *link* ou *switch*, utilizando caminhos de comunicação redundantes e não-redundantes. Para melhor avaliar o comportamento dos controladores e para que a carga necessária de geração do tráfego não influencie nos resultados obtidos, os controladores foram isolados em máquinas separadas do simulador Mininet.

As topologias utilizadas consideram os *links* de comunicação entre hosts: (i) sem

[1]https://akka.io/docs/

[2]http://mininet.org

[3]https://hewlettpackard.github.io/netperf/



(a) Topologia do caso 1. **(b) Topologia do caso 2.** **(c) Topologia do caso 3.**

Figura 1. Topologias utilizadas nos 3 casos testes.

caminhos alternativos (Figura 1a); (ii) um caminho alternativo (Figura 1b); e (iii) dois caminhos alternativos (Figura 1c). São consideradas falhas temporárias nos *links*, dado que o *link* é reestabelecido. Falhas do tipo *crash* são injetadas nos testes com o *switch*, o reestabelecimento do serviço se dá pela inserção de um novo switch, reestabelecendo a topologia inicial. O período de falha no *switch* é determinado pelo momento em que ocorre a parada do *switch* até que um novo seja iniciado. O tráfego gerado consiste em *streams* TCP e UDP gerados pela ferramenta *netperf*. Os dados coletados foram extraídos através da captura do tráfego durante a execução do *netperf* entre os hosts *H1* e *H2*. Para cada teste foram realizadas 3 repetições e analisado se realmente o comportamento se repete. Realizada esta análise com a constatação da igualdade entre os resultados, então os dados são gerados graficamente utilizando uma das repetições obtidas no experimento.

Nos experimentos dos casos de 1 a 3 (Seções 4.1 a 4.3) os controladores estão no modo reativo. A seção 4.4 analisa o ODL no modo proativo. Utilizou-se um *timeout* de 5s para que as regras instaladas nos *switches* expirem. De forma que, regras instaladas anteriormente não interfiram no comportamento do controlador. A Tabela 2 exibe as versões utilizadas na execução dos experimentos. As versões Beryllium-SR4 e Junco dos controladores ODL e ONOS, respectivamente, utilizadas em [J. M. S. Vilchez 2018] são versões antigas destes controladores. A avaliação dos controladores foi realizada comparando as versões antigas com as versões mais recentes (Oxygen-SR4 e Sparrow, do ODL e do ONOS respectivamente) que possuem a funcionalidade de encaminhamento de pacotes na camada 2.

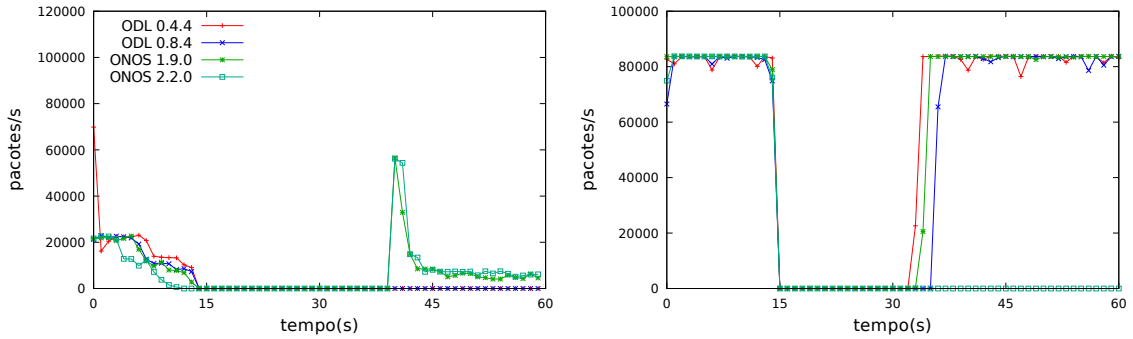
Tabela 2. Versões utilizadas dos controladores.

OpendayLight	**ONOS**
Beryllium-SR4 (0.4.4)	Junco (1.9.0)
Oxygen-SR4 (0.8.4)	Sparrow (2.2.0)

Nas subseções a seguir serão apresentados os dados coletados, evidenciando o tempo de resposta de cada um dos controladores. Em cada experimento, a falha de *link* ocorre exatamente aos 15s de execução e restaurada aos 30s. Perfazendo um período de 15s de inatividade. Para a interrupção foi utilizado o *link* em que os dados estavam sendo transmitidos, forçando o controlador a instalar novas regras para o caminho alternativo.

4.1. Caso 1 - Falha de *link* sem caminhos alternativos.

Uma falha nos *links* de comunicação, quando não existem caminhos alternativos, implica na interrupção do serviço até que o *link* seja restaurado. A topologia da figura 1a representa o cenário de falha com apenas um caminho entre origem (H1) e destino (H2), linha pontilhada em vermelho ilustra o caminho em que o tráfego é encaminhado.



(a) Fluxo TCP **(b) Fluxo UDP**

Figura 2. Sem caminhos alternativos.

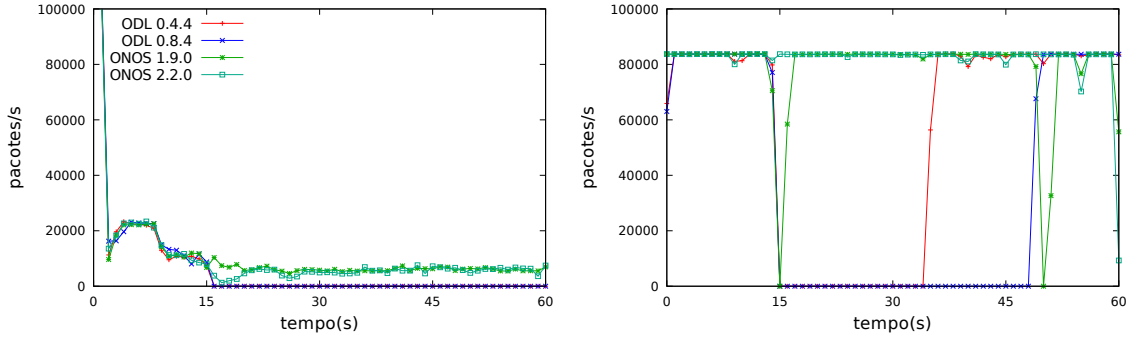
Na Figura 2 é apresentada, no eixo y, a taxa de pacotes por segundo durante o período de captura, com a falha no *link* entre os *switches S2* e *S1* . No eixo x, é possível observar o instante da falha que ocorre entre o período de 15s até 30s. Conforme esperado, durante o período de inatividade, não houve tráfego entre os *hosts H1* e *H2*. No entanto, depois de reestabelecida a conexão, é possível notar a diferença no comportamento entre os experimentos. Utilizando fluxos TCP (Figura 2a) o ONOS teve um atraso de 10 segundos para reestabelecer o serviço, independente da versão utilizada. O ODL, mesmo com a restauração do *link*, não conseguiu reestabelecer o serviço para a aplicação em ambas as versões. Com fluxos UDP (figura 2b) são obtidos resultados diferentes em relação à utilização do TCP. O controlador ONOS na versão Junco (1.9.0) foi capaz de reestabelecer o serviço com um atraso de 3 segundos. Na versão Sparrow (2.2.0) do ONOS, após a falha do *link*, a ferramenta *netperf* não pode continuar a execução como nos demais cenários. O controlador ODL, na versão Beryllium-SR4 (0.4.4) reestabeleceu a conexão após 3 segundos do reestabelecimento do *link*. Ao passo que, na versão Oxygen-SR4 (0.8.4), houve um atraso de 6 segundos para a restauração do serviço.

Como pode se observar, em fluxos TCP, o ONOS respondeu rapidamente ao reestabelecimento do *link* na versão mais antiga. Não foi possível avaliar a versão mais nova do ONOS devido a um comportamentos inesperado da ferramenta *netperf*. O ODL, por sua vez, não foi capaz de reestabelecer a conexão em nenhuma de suas versões. Com fluxos UDP, o ONOS na versão Junco apresentou o mesmo atraso que o ODL na versão Berrylium-SR4. A versão mais recente do ODL demorou 3 segundos a mais para detecção da volta do *link* que sua versão anterior.

4.2. Caso 2 - Falha de *link* com um único caminho alternativo.

Neste cenário existem dois caminhos entre os *hosts* avaliados (*H1* e *H2*), conforme a topologia da Figura 1b. Os *links* escolhidos pelos controladores para interligar os hosts foram *s4-s3* em todas as versões dos controladores, indicado pela linha pontilhada em vermelho.

Em um cenários ideal, espera-se que na falha de um dos *links* a comunicação ocorra através do caminho alternativo, de forma a garantir a continuidade do serviço oferecido. Neste experimento são avaliados: o *link* escolhido pelo controlador para comunicação entre os dois equipamentos, a utilização de um *link* alternativo, após uma falha e o comportamento do controlador com a restauração do *link* falho.



(a) Fluxo TCP **(b) Fluxo UDP**

Figura 3. Um caminho alternativo.

Utilizando fluxos TCP, a Figura 3a mostra o comportamento dos controladores durante a execução do experimento. Em ambas versões, o ONOS, além de garantir a disponibilidade do serviço, o tempo para detecção de falha no *link* e restaurar o serviço foi menor que 1 segundo, utilizando o caminho alternativo para encaminhamento dos fluxos, sem que o serviço fosse interrompido. Com o retorno do *link* falho o caminho instalado no *switch*, pelo ONOS, continuou a ser utilizado. O ODL, além de interromper o serviço quando houve queda no *link*, não reestabeleceu a conexão após o *link* ser restaurado.

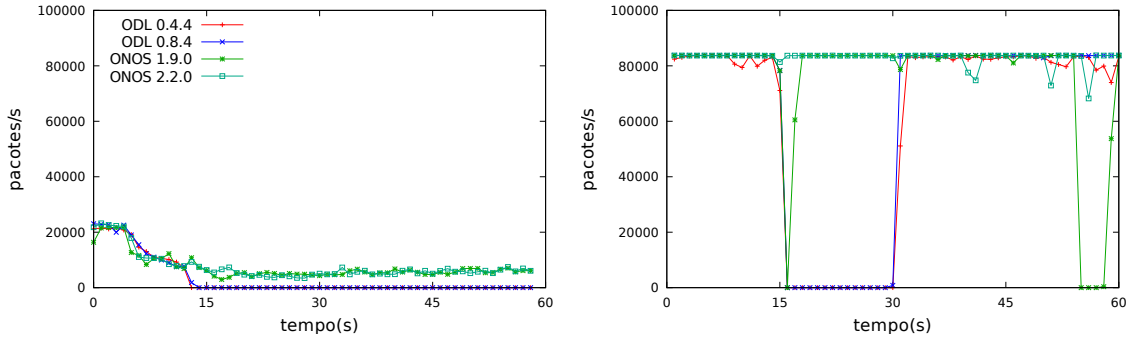
No experimento com fluxos UDP na topologia na Figura 1b , observa-se na Figura 3b diferenças entre a utilização de fluxos TCP (Figura 3a). O ONOS na versão 1.9.0, detectou a falha do *link* em um tempo menor que 2 segundos, enquanto que na versão mais recente o tempo para detecção e restauração do serviço foi menor que 1 segundo, não interrompendo a execução do serviço. O ODL não foi capaz de manter a comunicação entre os hosts durante o período de falhas. Considerando a versão 0.4.4 do ODL, após 5 segundos da restauração do *link*, a comunicação foi reestabelecida. Na versão 0.8.4 do ODL, foram necessários 19 segundos para que a comunicação fosse reestabelecida. O ODL não utilizou o caminho alternativo durante ou após a falha.

Como se observou, os resultados obtidos mostram que o ODL age diferente aos fluxos (TCP e UDP): com fluxos TCP o serviço não é reestabelecido após a falha do *link*; já com fluxos UDP o ODL reestabelece a comunicação com atraso após o *link* ser restaurado, apesar de deixar o caminho alternativo ocioso. O ONOS, por sua vez, utilizou o *link* alternativo em ambas versões, independente de fluxos TCP ou UDP e não interrompeu a continuidade do serviço.

4.3. Caso 3 - Falha de *link* com dois caminhos alternativos.

Este cenário avalia a queda de conexão e a possibilidade do controlador escolher mais do que um caminho alternativo (ver Figura 1c). A topologia permite, de acordo com o instante do experimento, que o controlador escolha um dentre três *links* para estabelecer a conexão. A Figura 4 ilustra o comportamento do tráfego durante o período de falhas.

Tanto o ONOS e o ODL escolheram utilizar o *link* com o *switch* mais próximo *(S3)*. No ODL, o tempo de reestabelecimento de *link* foi menor do que o observado no caso da topologia da Figura 1b. No ONOS, em ambas versões, utilizando fluxos TCP (Figura 4a), o serviço não foi interrompido. Com fluxos UDP (Figura 4b), a versão 1.9.0 do ONOS interrompeu o serviço até a detecção da falha e a instalação de regras nos *switches* para utilização de fluxos alternativos.



(a) Fluxo TCP **(b) Fluxo UDP**

Figura 4. Vários caminhos alternativos.

Na Figura 4a podemos observar a captura com fluxos TCP. O ODL, como nos casos anteriores, não manteve a continuidade do serviço após a queda do *link*, mesmo com um caminho alternativo e ocioso. O ONOS, por sua vez, com tempo de detecção inferior a 1 segundo, não interrompeu o serviço, visto a presença de *links* alternativos.

Com fluxos do tipo UDP, analisando a Figura 4b, observa-se que, em ambas as versões do ODL, retorna a comunicação somente após o reestabelecimento do *link*. Observa-se também que após a restauração do *link*, com atraso menor que 1 segundo, o ODL reestabelece a comunicação, apesar de haver dois *links* disponíveis e operacionais durante o período de falha. O ONOS, na versão 2.2.0, reestabelece a conexão com atraso menor que 1 segundo. Ao passo que, a versão 1.9.0 do ONOS possui um atraso entre 1 e 2 segundos para detecção e instalação de novos fluxos, interrompendo o serviço durante este período.

Vale destacar que a utilização de fluxos com o protocolo UDP, permitiu demonstrar uma deficiência do controlador ONOS na versão 1.9.0, apresentando interrupções quando executado com este protocolo de comunicação. Vale ressaltar também que não temos o conhecimento, no contexto dos trabalhos relacionados, da constatação deste fato apresentado por este controlador referente ao comportamento apresentado.

A seguir compara-se os resultados obtidos, para os mesmos casos de teste de [J. M. S. Vilchez 2018], em que foi utilizado fluxos UDP. Para o caso 1 do controlador ODL, o trabalho de Vilchez et al. verifica um tempo de 20 segundos para reação à falha, 17 segundos a mais do que o verificado nos experimentos deste trabalho. Para o ONOS, ainda no caso 1, Vilchez et al. indicou que não houve atraso para o reestabelecimento da conexão. Já os resultados obtidos neste artigo foram necessários 3 segundos para que o serviço fosse reestabelecido. Para o caso 2 do ODL, os resultados foram semelhantes aos obtidos no trabalho de Vilchez et al.: o ODL falhou ao manter a continuidade do serviço após a falha. O ONOS, diferentemente do que observado na versão avaliada por Vilchez

et al., não reestabeleceu a conexão instantaneamente, foi necessário um tempo menor que 2 segundos. No caso 3 o ODL reagiu da mesma forma que observado no trabalho de Vilchez et al. reagindo instantaneamente após a volta do link. O ONOS, ao contrário do que observado na versão avaliada por Vilchez et al, apresenta um atraso menor do que 2 segundos. Ou seja, os resultados obtidos por Vilchez et al. são semelhantes aos obtidos neste artigo, com diferenças para o tempo de resposta para restaurar a conexão após o reestabelecimento dos *links*, o que pode ser explicado pelo ambiente de execução.

4.4. ODL - Modo proativo

O ODL em modo proativo não recebe eventos *PACKET_IN*, ou seja, novas regras para tratamento de fluxos devem ser inseridas proativamente. São poucas as aplicações que suportam a utilização do modo proativo puro [OpenDaylight 2019]. Comparar o modo proativo é fundamental para analisarmos o comportamento do ODL, em seus modos de funcionamento. Neste experimento é analisado comportamento do controlador diante de falhas no plano de dados e a continuidade dos serviços executados utilizando fluxos TCP. Além disso, é avaliada a evolução do ODL em relação a versão Berrylium-SR4 (0.4.4) com a versão Oxygen-SR4 (0.8.4). A execução dos experimentos foi realizada utilizando as topologias presentes na Figura 1.

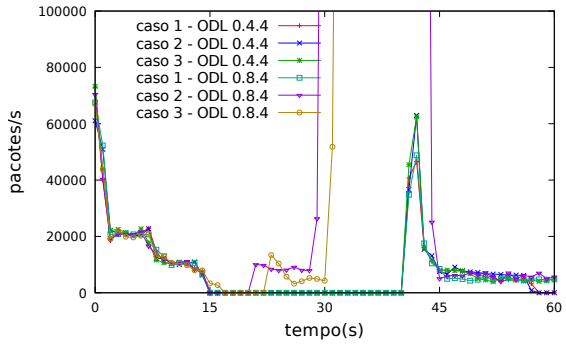


Figura 5. ODL - modo proativo

No caso 1 (Figura 1a) em que não existem caminhos alternativos, durante a interrupção do *link* o tráfego deve ser interrompido, sendo avaliado o tempo necessário para que o serviço seja reestabelecido. No experimento realizado, o ODL na versão 0.4.4 e na versão 0.8.4, foram necessários 10 segundos para que o serviço fosse reestabelecido após a restauração do *link*.

No caso 2 (Figura 1b) existe dois caminhos entre o host de origem (H1) e o host de destino (H2). Avaliou-se o comportamento do controlador e o tempo necessário para o reestabelecimento do serviço. No experimento ambas as versões utilizaram o caminho *S3-S4* para encaminhar os fluxos. No ODL na versão 0.4.4 foram necessários 10 segundos para o reestabelecimento do serviço após o *link* ser restaurado. Na versão 0.8.4, foram necessários 6 segundos para que o controlador identificasse a falha e redirecionasse para o caminho alternativo. Contudo, durante 10 segundos, após o reestabelecimento do *link*, houve uma inundação de pacotes, com a taxa de pacotes, em média, 26 vezes maior do que a taxa normal.

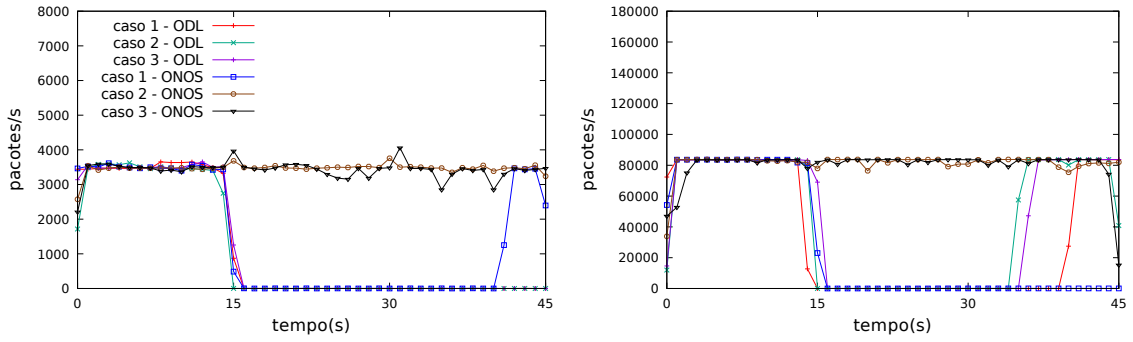
No caso 3 (Figura 1c), existem três caminhos entre o host de origem (H1) e o host de destino (H2), sendo que um caminho é sempre utilizado, restando dois de redundância.

O ODL na versão 0.4.4, como no caso 2 (Figura 1b), levou 10 segundos após a restauração do *link* para que o serviço fosse restaurado. Na versão 0.8.4 do ODL, foram necessários 8 segundos para que o serviço fosse restaurado. Conforme observado no caso 2, utilizando o modo reativo, houve um período de 10 segundos de inundação de pacotes após a restauração do *link*.

O ODL, mesmo em modo reativo, não reagiu a falhas de forma satisfatória, visto que o ONOS possibilita que com a presença de *links* redundantes seja garantida a disponibilidade nos serviços do plano de dados. No ODL, as melhorias no modo proativo não foram capazes de fazer com que o serviço fosse mantido sem interrupção, mesmo com a presença de caminhos alternativos para os fluxos.

4.5. Falha do switch

A falha em *switches* é semelhante à falha de *link*, visto que o controlador deve implementar mecanismos que detectam que houve uma falha no *switch* e redirecionar o tráfego para um caminho alternativo. No entanto, os *switches* iniciam com a tabela de fluxo vazia, podendo impactar no comportamento dos controladores. Os experimentos são executados utilizando as topologias descritas anteriormente (Figura 1). Neste experimento é avaliado o comportamento do ONOS e do ODL utilizando o modo reativo, nas versões Sparrow (2.2.0) e Oxygen-SR4 (0.8.4), respectivamente.



(a) Fluxo TCP **(b) Fluxo UDP**

Figura 6. Falha do *switch*, levando em consideração as topologias apresentadas na Figura 1

Nos experimentos com o fluxo TCP, utilizando a topologia do caso 1 (Figura 1a), o ONOS levou 11 segundos para que o serviço fosse reestabelecido, após a restauração do *link*. O ODL não foi capaz de restaurar o serviço como observado na Seção 4.1 . Utilizando fluxos UDP, o ODL levou 10 segundos para que o serviço fosse restaurado, após o retorno do *link*. A ferramenta *netperf* falhou ao realizar a geração de tráfego para o ONOS com fluxos UDP.

Avaliando o caso 2 (Figura 1b) com fluxos TCP, o ODL não foi capaz de reestabelecer o serviço, mesmo após o *link* ser restaurado. O ONOS, não interrompeu o serviço, com o tempo de detecção da falha do *switch* menor que 1 segundo. Utilizando fluxos UDP, o ODL levou 5 segundos para que o *link* alternativo fosse utilizado, visto que havia um caminho ocioso. O ONOS com fluxos UDP não interrompeu o serviço, com o tempo de detecção de falhas e reconfiguração dos equipamentos menor que 1 segundo.

No caso 3 (Figura 1c) os resultados foram semelhantes ao caso 2, visto que o ONOS, em fluxos TCP e UDP, mantém a disponibilidade do serviço. O ODL, utilizando fluxos TCP não consegue reestabelecer o serviço após a restauração do *link*. Utilizando fluxos UDP, o ODL levou 6 segundos, para reestabelecer o serviço com o retorno do *link*.

5. Conclusão

Este trabalho apresentou uma análise comparativa entre os controladores ODL e ONOS no contexto de tolerância a falhas no plano de dados considerando diversos cenários de falhas. Em particular, foram consideradas falhas e reestabelecimentos de *links* e paradas e retornos de *switches*, com diferentes fluxos tanto no modo reativo quanto no modo proativo. Os resultados experimentais descreveram o tempo que os controladores levaram para detectar as falhas, redirecionamento do tráfego de dados para caminhos redundantes e identificação do reestabelecimento de *links* e *switches*. De acordo com os experimentos práticos realizados, pode-se concluir que o controlador ONOS foi capaz de identificar caminhos redundantes, quando disponíveis, e manteve a disponibilidade do plano de dados durante falhas de *links* e *switches*. O controlador ODL falhou na utilização de caminhos redundantes, não mantendo a disponibilidade do plano de dados durante a falha de *links* e *switch* e após o reestabelecimento dos mesmos. Apesar de ambos os controladores trabalharem com o reestabelecimento de *links*, as falhas observadas do ODL estão relacionadas com a implementação do módulo responsável pela camada 2, que não detecta alterações no estado das portas dos *switches*. Como trabalhos futuros pretende-se avaliar o comportamento ODL e do ONOS considerando o modo distribuído em relação à tolerância a falhas e a consistência no plano de controle.

Referências

Bannour, F., Souihi, S., and Mellouk, A. (2018). Distributed sdn control: Survey, taxonomy, and challenges. *IEEE Communications Surveys & Tutorials*, 20(1):333–354.

Bianco, A., Giaccone, P., Mashayekhi, R., Ullio, M., and Vercellone, V. (2017). Scalability of onos reactive forwarding applications in isp networks. *Computer Communications*, 102:130–138.

Canini, M., Kuznetsov, P., Levin, D., and Schmid, S. (2015). A distributed and robust SDN control plane for transactional network updates. In *INFOCOM*.

Curic, M., Despotovic, Z., Hecker, A., and Carle, G. (2018). Transactional network updates in sdn. In *EuCNC*, pages 203–208. IEEE.

da Silva, A. S., Smith, P., Mauthe, A., and Schaeffer-Filho, A. (2015). Resilience support in software-defined networking: A survey. *Computer Networks*, 92:189 – 207.

Dang, H. T., Sciascia, D., Canini, M., Pedone, F., and Soulé, R. (2015). Netpaxos: Consensus at network speed. In *SOSR/SIGCOMM*.

Darianian, M., Williamson, C., and Haque, I. (2017). Experimental evaluation of two openflow controllers. In *ICNP*, pages 1–6.

Gonzalez, A. J., Nencioni, G., Helvik, B. E., and Kamisinski, A. (2016). A fault-tolerant and consistent sdn controller. In *LANOMS*, pages 1–6. IEEE.

Ho, C. C., Wang, K., and Hsu, Y. H. (2016). A fast consensus algorithm for multiple controllers in software-defined networks. In *ICACT*.

J. M. S. Vilchez, D. E. S. (2018). Fault tolerance comparison of onos and opendaylight sdn controllers. In *NetSoft*, pages 277–282.

Karakus, M. and Durresi, A. (2017). A survey: Control plane scalability issues and approaches in software-defined networking (sdn). *Computer Networks*, 112:279 – 293.

Katta, N., Zhang, H., Freedman, M., and Rexford, J. (2015). Ravana: Controller fault-tolerance in software-defined networking. In *SIGCOMM*, page 4. ACM.

Kohler, T., Dürr, F., and Rothermel, K. (2018). Zerosdn: A highly flexible and modular architecture for full-range distribution of event-based network control. *IEEE Transactions on Network and Service Management*, 15(4):1207–1221.

Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., and Shenker, S. (2010). Onix: A Distributed Control Platform for Large-scale Production Networks. In *OSDI*.

Mahajan, K., Poddar, R., Dhawan, M., and Mann, V. (2016). Jury: Validating controller actions in software-defined networks. In *DSN*.

Medved, J., Varga, R., Tkacik, A., and Gray, K. (2014). Opendaylight: Towards a model-driven sdn controller architecture. In *2014 IEEE 15th International Symposium on*, pages 1–6. IEEE.

Muqaddas, A. S., Bianco, A., Giaccone, P., and Maier, G. (2016). Inter-controller traffic in onos clusters for sdn networks. In *ICC*, pages 1–6.

Obadia, M., Bouet, M., Leguay, J., Phemius, K., and Iannone, L. (2014). Failover mechanisms for distributed sdn controllers. In *NOF*, volume Workshop, pages 1–6.

ONOS (2016). Intent framework.

OpenDaylight (2019). Opendaylight sdn controller platform (oscp):overview.

Schiff, L., Schmid, S., and Kuznetsov, P. (2016). In-Band Synchronization for Distributed SDN Control Planes. *SIGCOMM*, 46(1).

Sinha, Y., Haribabu, K., et al. (2017). A survey: Hybrid sdn. *Journal of Network and Computer Applications*, 100:35–55.

T.BAH, M., Azzouni, A., Nguyen, M. T., and Pujolle, G. (2019). Topology discovery performance evaluation of opendaylight and onos controllers. In *ICIN*.

Venâncio, G., Turchetti, R., Camargo, E. T., and Duarte Jr, E. (2018). Vnf-consensus: A virtual network function for maintaining a consistent distributed sdn control plane. In *GLOBECOM*, pages 1–8. IFIP Open Digital Library.

Zhu, L., Karim, M. M., Sharif, K., Li, F., Du, X., and Guizani, M. (2019). SDN controllers: Benchmarking & performance evaluation. *CoRR*, abs/1902.04491.

Zulu, L. L., Ogudo, K. A., and Umenne, P. O. (2018). Emulating software defined network using mininet and OpenDaylight controller hosted on Amazon Web Services cloud platform to demonstrate a realistic programmable network. In *ICONIC*.