Estratégias para Resiliência em SDN : Uma Abordagem Centrada em Multi-Controladores Ativamente Replicados

Eros S. Spalla[1], Diego R. Mafioletti[1], Alextian B. Liberato[1]
Christian Rothenberg[2], Lasaro Camargos[3], Rodolfo da S. Villaca[1], Magnos Martinello[1]

[1]Universidade Federal do Espirito Santo (UFES) – Vitória/ES
[2]Universidade Estadual de Campinas (UNICAMP) – Campinas/SP
[3]Universidade Federal de Uberlândia (UFU) – Uberlândia/MG

spalla@ifes.edu.br, diego@saorc.com.br, alextian@ifes.edu.br
chesteve@dca.fee.unicamp.br, lasaro@ufu.br
rodolfo.villaca@ufes.br, magnos@inf.ufes.br

Abstract. *Software Defined Networking (SDN) are based on the separation of control and data planes. The SDN controller, although logically centralized, should be effectively distributed for high availability. Since the specification of OpenFlow 1.2, there are new features that allow the switches to communicate with multiple controllers that can play different roles –* master*,* slave*, and* equal*. However, these roles alone are not sufficient to guarantee a resilient control plane and the actual implementation remains an open challenge for SDN designers. In this paper, we explore the OpenFlow roles for the design of resilient SDN architectures relying on multi-controllers. As a proof of concept, a strategy of active replication was implemented in the Ryu controller, using the OpenReplica service to ensure consistent state among the distributed controllers. The prototype was tested with commodity RouterBoards/MikroTik switches and evaluated for latency in failure recovery and switch migration for different workloads. We observe a set of trade-offs in real experiments with varyin workloads at both the data and control plane.*

Resumo. *As Redes Definidas por Software (SDN) separam os planos de dados e de controle. Embora o controlador seja logicamente centralizado, ele deve ser efetivamente distribuído para garantir alta disponibilidade. Desde a especificação OpenFlow 1.2, há novas funcionalidades que permitem aos elementos da rede se comunicarem com múltiplos controladores, que podem assumir diferentes papéis –* master*,* slave*, e* equal*. Entretanto, esses papéis não são suficientes para garantir resiliência no plano de controle, pois delega-se aos projetistas de redes SDN a responsabilidade por essa implementação. Neste artigo, exploramos os papéis definidos no protocolo OpenFlow no projeto de arquiteturas resilientes SDN com base em multi-controladores. Como prova de conceito uma estratégia de replicação ativa foi implementada no controlador Ryu usando o serviço OpenReplica para garantir a consistência dos estados. O protótipo foi testado com* switches *RouterBoards/MikroTik comerciais avaliando-se a latência na recuperação de falha e na migração de switches entre controladores. Observamos diferentes compromissos de projeto em experimentos reais sujeitos a várias cargas nos planos de dados e de controle.*

Agradecimentos à CAPES, CNPq e FAPEMIG pelo suporte à execução deste trabalho.

1. Introdução

A flexibilidade oferecida pela arquitetura das Redes Definidas por *Software* (SDN – *Software-Defined Networking*) apoia-se na separação do plano de controle do plano de dados, permitindo a implementação das funções de controle de rede a partir de uma visão centralizada [Kreutz et al. 2015]. Nesta abordagem, o estado dos dispositivos da rede é determinado por um controlador, que envia suas decisões ao dispositivo, sob a forma de entradas nas tabelas de fluxos, por meio do protocolo OpenFlow [Rothenberg et al. 2010].

Atualmente, o protocolo OpenFlow é considerado o padrão *de facto* em SDN. Porém, garantir alta disponibilidade no plano de controle é um desafio em aberto para o qual o protocolo OpenFlow não oferece uma solução pronta, e sim um conjunto de possibilidades. Desde a versão 1.2 da especificação, o OpenFlow tem incorporado novas funcionalidades que permitem aos *switches* se comunicarem com múltiplos controladores. Uma dessas funcionalidades são os papéis (*roles*) de *master*, *equal* e *slave*, assumidos por múltiplos controladores na comunicação com os dispositivos no plano de dados. Entretanto, apenas essas novas *features* não são suficientes para garantir uma solução resiliente, uma vez que, o padrão OpenFlow define os papéis, mas deixa a cargo dos projetistas de redes SDN a escolha de estratégias para implementá-las.

Uma onda de trabalhos recentes tem abordado o problema de resiliência em redes SDN (ex: Sec.V [Kreutz et al. 2015]), incluindo o posicionamento de controladores distribuídos [Heller et al. 2012], clustering [Penna et al. 2014], particionamento do controle [Koponen et al. 2010], garantias de consistência [Botelho et al. 2013], entre outros. Uma característica comum a todos os trabalhos relacionados que encontramos é não terem explorado as opções de conectividade simultânea com múltiplos controladores, usando as três opções de papéis disponíveis no protocolo OpenFlow.

A primeira contribuição do trabalho, é apresentar e discutir diferentes estratégias de replicação do plano de controle e seus *trade-offs*, abordando aspectos práticos de cada uma. Nossa proposta baseia-se no levantamento das características e componentes que tornam esses sistemas funcionais, tais como a relação dos papéis OpenFlow dos controladores e a distribuição de estados, assim como o comportamento da rede em caso de falha e restauração nos controladores.

Como prova de conceito de um plano de controle resiliente, uma estratégia de replicação ativa foi implementada no controlador Ryu[1], e o protótipo foi testado com *switches* RouterBoards comerciais, modificados por uma arquitetura aberta [Liberato et al. 2014], onde foram observados a latência na recuperação de falha e restauração de falha/migração de *switches* entre controladores. A descrição do protótipo implementado e a análise experimental são nossa segunda contribuição ao estado da arte, já que trabalhos relacionados apenas consideraram ambientes emulados.

O restante deste artigo está estruturado da seguinte forma: a Seção 2 discute diferentes abordagens para implementação de resiliência em controladores OpenFlow; a Seção 3 apresenta uma proposta para controladores resilientes; a Seção 4 indica a metodologia experimental; a Seção 5 expõe os resultados obtidos e aponta o comportamento do experimento em um ambiente real; a Seção 6 reporta os trabalhos relacionados; e por fim, a Seção 7 tece considerações finais e indica os trabalhos futuros.

[1] https://github.com/osrg/ryu

2. O problema arquitetura multi-controlador

As características arquiteturais e inovações advindas das SDN são objeto de inúmeros estudos e discussões [Kreutz et al. 2015]. Nesse contexto, um ponto controverso é justamente uma de suas principais características: a centralização do controle. Isto porque, caso tal visão centralizada seja implementada por um único controlador, uma falha nesse elemento pode levar à indisponibilidade total da rede. O problema fundamental e os *trade-offs* por trás da distribuição do controle em redes SDN podem ser modelados adaptando o teorema de CAP [Panda et al. 2013]. [2]

Embora tenha ficado claro desde a especificação inicial do OpenFlow que o controlador, apesar de logicamente centralizado, poderia ser implementado de forma distribuída, somente a partir da versão 1.2 é que os mecanismos para esta distribuição começaram a ser especificados. Nas versões 1.0 e 1.1, cada *switch* se conecta apenas a um controlador, que torna-se um ponto único de falha e demanda ações não especificadas para reposição. Desde a versão 1.2, *switches* OpenFlow podem se conectar a diversos controladores simultaneamente, que devem assumir um dentre três papéis: *master*, *slave*, e *equal*. Contudo, na literatura, não há sequer *guidelines* ou melhores práticas de como estes papéis devam ser usados. Assim, cabe aos projetistas de SDN o uso de técnicas que garantam a disponibilidade dos serviços executados pelo controlador e a dependabilidade da rede. A abordagem óbvia para se atender tal requisito é a redundância do estado do controlador via alguma forma de replicação, como até é sugerido pelos nomes dos papéis, permitindo que outro controlador continue a atuar sobre o mesmo estado, em caso de falha do primeiro ou, ainda, que múltiplos controladores compartilhem tal tarefa.

Na implementação da replicação em ambiente SDN multicontrolador, diversas perguntas precisam respostas, dentre elas destacamos as seguintes: Vários controladores processam comandos que atualizam o estado ou somente um processa e informa aos outros o novo estado? No primeiro caso, a mesma ordem de processamento é imposta para garantir consistência e, no segundo, como o estado é compartilhado? Todos acessam o estado global ou ele é particionado entre os controladores? Como são mascaradas as falhas de um coordenador? Na seção seguinte, discutiremos tais perguntas e possíveis respostas.

2.1. Suporte a múltiplos controladores

Em uma rede implementada com o protocolo OpenFlow, os *switches* mantém tabelas de fluxo que ditam como o tráfego deve ser roteado. Quando um pacote de dados não pode ser roteado, o controlador responsável pelo *switch* é informado via um evento *packet-in*. Baseado em seu estado, o controlador gera uma nova regra a ser inserida na tabela de fluxo do *switch* para lidar com o pacote e outros similares. Assim, se há apenas um controlador por *switch*, a tabela de fluxos é sempre consistente com o estado do controlador.

Contudo, a partir da definição do conceito de papéis, um *switch* pode se conectar a mais de um controlador. Neste caso, quando a conexão ocorre, por padrão, o controlador assume o papel *equal*, implicando que todos os controladores têm igual poder de atualizar tabelas do *switch*. Neste cenário os controladores devem, de uma forma não especificada,

[2] O teorema CAP (Consistency, Availability e Partition Tolerance), também cohecido como Teorema de Brewer, afirma afirma a impossibilidadem de um sistema computacional distribuído garantir simultaneamente consistência, disponibilidade e tolerância ao particionamento.

coordenar a atualização de seus estados, mantendo uma visão logicamente centralizada da rede e garantindo que suas intervenções sejam consistentes.

Outros papéis que podem ser assumidos pelo controlador são *slave* e *master*. Ao assumir o papel *master*, assim como no *equal*, o controlador passa a ter privilégios sobre o *switch*, com permissão de leitura e escrita, além de receber mensagens assíncronas. Já no modo *slave*, o controlador tem limitado seu acesso ao *switch*, podendo realizar apenas operações de leitura, isto é, nenhuma mensagem que altere a tabela de fluxos do *switch* é processada. Caso comandos de escrita sejam enviados ao *switch*, uma mensagem de erro será gerada. Mensagens assíncronas também não são enviadas ao controlador, à exceção das mensagens de *port-status*, que informam, por exemplo, sobre o estado das portas.

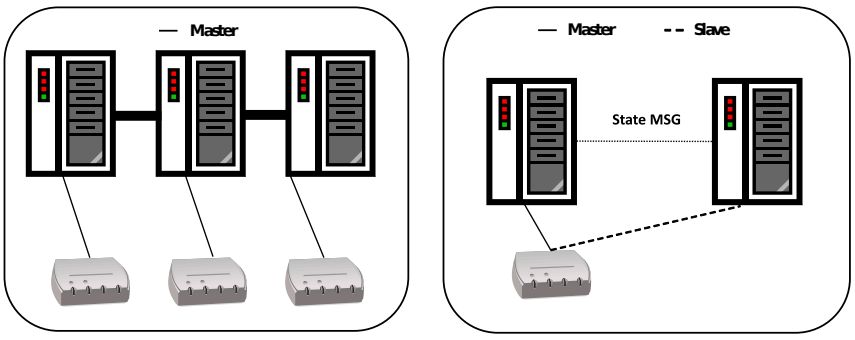
2.2. Configuração Equal

Com base nas definições dos papéis OpenFlow, podemos descrever uma configuração básica para múltiplos controladores rodando uma aplicação de *learning switch*:

1. um *switch* se conecta a *N* controladores;
2. todos os controladores assumem o papel *equal*.

Neste arranjo simples, todos os *switches* são configurados com os endereços dos controladores, e estes não demandam nenhuma configuração, uma vez que o papel padrão no OpenFlow é o *equal*. Um *switch* ao receber um pacote que não tenha fluxo instalado em sua tabela, encaminha um *packet-in* ao plano de controle, isto é, a todos os controladores. Tais controladores, ao receberem o pacote, realizam o seu processamento de forma concorrente, e enviam um *packet-out* para o *switch*. Desde que processem os mesmos pacotes, na mesma ordem, e que o processamento seja determinístico, os estados dos controladores evoluirão consistentemente. Neste cenário, em caso de falha, os controladores restantes continuam a operar a rede, normalmente. Este é o princípio da replicação de máquinas de estados [Lamport 1978, Schneider 1990], também conhecido como *Replicação Ativa*, ilustrado na Figura 1(a). Obviamente, a dificuldade na implementação desta técnica está em garantir a entrega, totalmente ordenada, das mensagens aos controladores, o que requer alguma forma de comunicação em grupo [Défago et al. 2004].

Apesar de conceitualmente simples, essa implementação tem desvantagens claras: maior uso de recursos – já que todos os controladores processam todos os pacotes;



(a) Replicação Ativa (b) Replicação Passiva

Figura 1. Arquiteturas Multicontroladores

sobrecarga da rede de controle – devido ao *broadcast* dos *packet-in* e múltiplas respostas; a capacidade de processamento do plano de controle não escala com o número de controladores, já que todos processam os mesmos pacotes e mantém o mesmo estado; e por fim, a menos que os comandos que alteram as tabelas de fluxo sejam idempotentes, o resultado pode não refletir o que foi determinado pelos controladores – por exemplo, em nossos experimentos, observamos a duplicação de pacotes causada pelo processamento de comandos duplicados pelo *switch*.

2.3. Configuração Master-Slave

Uma solução alternativa, que minimiza alguns dos efeitos indesejáveis listados anteriormente, consiste em ter apenas um dos controladores processando as mensagens e respondendo ao *switch*, e repassando suas mudanças de estado aos outros controladores. Esta técnica, conhecida como *Replicação Passiva*, *primary-backup* ou, finalmente, *master-slave*, ilustrado na Figura 1(b) [Budhiraja et al. 1993] diminui o processamento nos controladores, elimina o não determinismo na atualização do estado e evita as respostas redundantes e seus efeitos. Do ponto de vista do *OpenFlow*, esta técnica pode ser implementada da seguinte forma:

1. um controlador assume o papel *master* para todos os *switches* da rede;
2. os demais controladores assumem o papel *slave* para todos os *switches* da rede.

Assim, como no caso da replicação ativa, apenas atribuir os papéis não é o suficiente para levar a uma solução completa. Se na replicação ativa era necessário garantir a entrega e ordenação das mensagens, na passiva é necessário garantir que as atualizações feitas no controlador mestre sejam replicadas para os escravos, novamente usando alguma forma de comunicação em grupo. Além disso, é também necessário que os escravos monitorem o mestre e que um deles assuma seu papel no caso de falha, informando os *switches*.

2.4. Configuração Multi-Master / Multi-Slave

A terceira proposta discutida para prover resiliência ao plano de controle, utiliza mecanismos para que os controladores não sejam o gargalo da rede. Em um arranjo com M *switches* e N controladores pode-se ter a seguinte configuração:

1. para cada *switch* da rede, um controlador deve assumir o papel *master*;
2. para cada *switch* da rede, $N - 1$ controladores devem assumir o papel *slave*.

Essa proposta permite que os N controladores processem *packet-in*, já que os *switches* são organizados de modo que um controlador atue como *master* para um *switch* e *slave* para os demais. Nos *switches* nenhuma configuração extra é necessária, além dos endereços dos controladores, levando a uma rede operacional com "mínimo" esforço. Todavia, ainda existem questões que precisam ser respondidas, sendo provavelmente a mais importante a seguinte: se os estados dos *switches* estão particionados entre os controladores, como implementar a visão centralizada do controlador OpenFlow?

2.5. Configuração via Data Store

Uma variação das configurações anteriores é utilizar um *data store* externo, tolerante a falhas, no qual o mestre espelha seu estado interno, e que é usado pelos escravos para atualizarem seus estados. Com o uso desta abstração extra, os controladores a utilizam para manter atualizados os seus estados; a visão centralizada passa a ser implementada em uma

base externa, e a visão particionada, por exemplo, corresponderia a simplesmente o controlador ignorar parte dos dados da base externa. Tal arquitetura tem a vantagem de abstrair a comunicação entre controladores, possivelmente simplificando a implementação.

Outra questão importante é como os papéis são divididos entre os *switches*. Novamente um serviço externo pode ser usado, como por exemplo os serviços de coordenação ZooKeeper [Hunt et al. 2010] ou Doozer[3]. Esses serviços podem ser usados para monitorar as réplicas e, em caso de falhas, eleger um mestre para cada *switch*. O novo mestre deve então agregar o estado do controlador que falhou ao seu, a partir das informações contidas no *data store*, e, por fim, informar os *switches* "órfãos" quem é seu novo *master*.

2.6. Considerações parciais

Algumas considerações podem ser feitas em relação a essa última perspectiva de um plano de controle resiliente. O processamento de pacotes por múltiplos controladores de forma concorrente é uma interessante escolha, visto que, uma importante característica para sistemas distribuídos é adicionada: escalabilidade. A medida que controladores são adicionados a rede, o *throughput* do plano de controle aumenta. No entanto, ter o estado da rede distribuído entre os controladores impõe desafios aos projetistas de redes SDN, tais como: a manutenção da consistência das informações e a coordenação das réplicas. Para avaliar essas conjecturas e os *trade-offs* em implementações reais, e avançar na viabilidade de uma solução de SDN com alta disponibilidade, optamos por desenvolver uma prova de conceito.

3. Proposta de Arquitetura Resiliente e Implementação do Protótipo

Após as discussões sobre diferentes estratégias e relações de compromisso na implementação de redes SDN com múltiplos controladores, nesta seção é apresentada a proposta de um plano de controle resiliente e o protótipo implementado.

Com o objetivo de lidar com os diferentes *trade-offs* apresentados nos exemplos anteriores, os seguintes pré-requisitos foram priorizados: (1) todos os controladores devem estar aptos a processar *packet-in*, isto é, atuar com o papel *master* de forma concorrente; (2) a distribuição de estados deve ser ativa, levando à resiliência no plano de controle, enquanto o estado da rede é replicado proativamente entre os controladores.

O protótipo implementado faz uso de dois controladores e duas RouterBoards RB2011iLS-IN, organizados de acordo com a Figura 2(a).

Um importante ponto no projeto de um plano de controle resiliente, é a implementação da visão centralizada (estado da rede compartilhado de forma consistente) em controladores distribuídos. Nossa proposta utiliza o OpenReplica[4] para manter a consistência do estado da rede. OpenReplica é um serviço que provê replicação e sincronização para sistemas distribuídos em larga escala. Baseado em uma nova implementação de máquinas de estados replicadas via Paxos [Lamport 1998], que utiliza uma abordagem orientada a objetos, o sistema, ativamente, cria e mantém réplicas vivas para objetos fornecidos pelo usuário. Assim, de forma transparente aos clientes, os objetos replicados são acessados como se fossem objetos locais (vide Figura 3).

[3]https://github.com/ha/doozerd

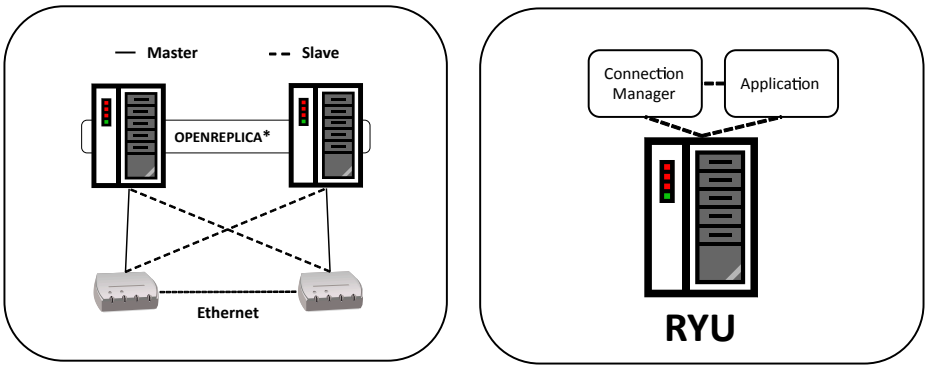
[4]http://openreplica.org/

A Figura 2(b) apresenta a organização do controlador com as funções do sistema divididas em dois módulos. O primeiro, chamado *ConnectionManager*, gerencia a comunicação entre os controladores, coordena as operações de troca de papéis, recuperação e restauração de falhas; já o segundo, chamado *Application*, carrega a aplicação a ser executada, neste caso, um *learning switch* com suporte ao OpenFlow 1.3.

Ao iniciar a execução dos controladores, os módulos são carregados em sequência, sendo *ConnectionManager* o primeiro. Como a conexão de todos *switches* da rede aos controladores é uma premissa para o funcionamento do sistema, o *ConnectionManager* mantém uma função aguardando os eventos de conexão dos *switches*. À medida que *switches* conectam-se aos controladores, o módulo *ConnectionManager* executa uma operação que configura os papéis para cada *switch*, na qual cada *switch* se conecta a um controlador com o papel *master* e a *N* controladores com o papel *slave*. De modo a tornar essa operação mais simples, já que o evento de conexão dos *switches* é aleatório, os controladores ordenam esses eventos de acordo com o *dpid* do *switch*. Por exemplo, suponha dois *switches*: 01 e 02. Estes podem conectar-se aos controladores em qualquer ordem, entretanto, os controladores ao ordenarem esses eventos, garantem que a conexão sempre será feita primeiro pelo *switch* 01 e depois pelo *switch* 02.

Os controladores são responsáveis pela detecção das falhas do plano de controle, e para tratar estes eventos, o protótipo utiliza dois procedimentos: recuperação de falha, e a restauração de falha. A situação de falha ocorre quando há indisponibilidade de um dos controladores. Na implementação protótipo, é considerado um modelo de sistema assíncrono com falhas de controladores, do tipo *fail-stop*; os canais de comunicação não duplicam nem perdem mensagens, e a perda de conexão é utilizada como indicador de falha nos controladores

A **recuperação de falha** consiste em um processo que faz com que o controlador, que continua ativo, torne-se o novo *master* para todos os *switches* que perderam o seu controlador principal. Os passos executados na operação, são: no momento em que a falha é detectada, o módulo *ConnectionManager* executa uma função que verifica quais são os *switches* que têm como *master* o controlador que falhou, e os envia uma mensagem *role-request master*. Cada *switch* responde com uma mensagem *role-reply* para confirmar



(a) Proposta de Resiliência para o plano de controle OpenFlow

(b) Organização do Controlador

Figura 2. Protótipo Implementado.

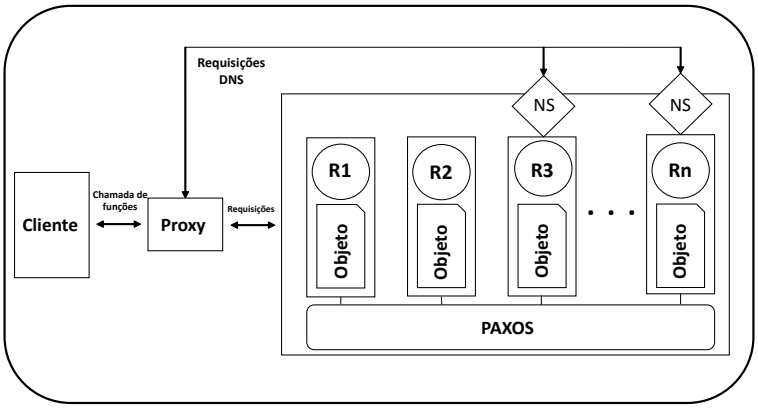


Figura 3. OpenReplica - Replicação e sincronização do estado da rede.

a operação. Já a restauração, ocorre quando o controlador que falhou se recupera da falha, ou seja, está apto novamente a processar pacotes. Nesse caso, o controlador, que volta da falha, executa o protocolo de migração de *switches* e recupera os *switches* que o tinham como *master* no momento anterior a falha.

No processo de **restauração de falha**, nossa proposta apresenta uma pequena variação da técnica para migração de *switches* entre controladores em [Dixit et al. 2013]. O controlador que volta de uma falha, no momento da inicialização de seus módulos, já carrega os *switches* em modo *equal*. Esse procedimento otimiza a execução da migração, já que as mensagens de troca de papel são enviadas na inicialização do controlador. Após essa operação, uma mensagem de migração é enviada para o controlador que já estava ativo, e então, a mesma sequência descrita por [Dixit et al. 2013] é seguida.

4. Avaliação Experimental

Na avaliação experimental foram usados *switches* comerciais modificados conforme [Liberato et al. 2014]. A ideia é explorar a capacidade de alguns equipamentos existentes no mercado, que podem ser usados para prototipação, com desempenho compatível e baixo custo. Com base em experimentos realizados com RouterBoards, é possível inferir como o protótipo se comporta em um ambiente real.

O ambiente utilizado para realizar os experimentos é um computador (Core i5 3.2GHz, 6GB de RAM, HD 320GB 5400rpm, SO Ubuntu 14.04 64bits), executando os controladores; duas Routerboards RB2011iLS-IN; e dois computadores (CPU AMD Athlon 64 X2 4000+, 2GB de RAM, HD 160GB 7200rpm, SO Ubuntu 14.04 64bits).

4.1. Metodologia

Para permitir o mínimo de interferência entre os cenários, os equipamentos foram reinicializados em todos os experimentos, garantindo que informações contidas no *cache* não influenciassem os resultados. Nos testes e na avaliação da proposta apresentada os seguintes parâmetros foram medidos: latência de recuperação e restauração de falha dos controladores, com diferentes taxas de tráfego no plano de controle; latência de *packet-in* nos controladores, antes e após a falha; latência de recuperação e restauração de falha dos controladores, com diferentes taxas de tráfego no plano de dados; e o consumo de processador das Routerboards nos diferentes cenários apresentados.

4.2. Carga de trabalho

De modo a gerar o tráfego necessário de acordo com as demandas de cada cenário, foi criado um gerador de tráfego em Perl que utiliza a extensão NET::RawIP para criar pacotes IP a uma taxa pré-definida. No ambiente de experimentação, os controladores não foram o *bottleneck* dos testes, ao contrário, por conta do *hardware* utilizado, seria possível aumentar a taxa de envio de pacotes. Devida a limitação do *hardware* utilizado, especial da *CPU*, limitamos a taxa de geração de pacotes a no máximo 1000 *packet-in/s*. Então, as taxas de envio pré-definidas foram: 250, 500, 750 e 1.000 *packet-in/s*.

Para os cenários com propósito de gerar tráfego no plano de controle, a aplicação de *switching* foi alterada para que todos os pacotes gerados pelos *hosts* fossem enviados aos controladores, ou seja, não haviam regras instaladas nos *switches*. Já para os testes com foco no plano de dados, foram instaladas regras apenas para fazer o encaminhamento entre os dois *hosts* da rede, assim, nenhum *packet-in* foi enviado aos controladores.

Em cenários cujo objetivo era gerar tráfego no plano de dados, utilizou-se a ferramenta Iperf [5]. Com o *throughput* das RouterBoards conhecido, as taxas de envio foram definidas de forma que houvesse um aumento gradual até o limite dos equipamentos. As taxas pré-definas foram: 180, 360, 540 e 720 Mbps.

4.3. Métricas

Para cada parâmetro a ser medido coletaram-se 30 amostras nos diferentes cenários. Foi calculado o intervalo de confiança de 95% para essas amostras e este se mostrou aceitável para garantir a confiabilidade do resultado as experimentações.

Para medir as operações realizadas pelos controladores, foram inseridos quatro *timestamps* no código. Na recuperacao de falha, assim que o controlador envia a mensagem *change-role-request* ao *switch*, e logo após a recepção da mensagem *change-role-reply*; na restauração de falha, logo que a migração é iniciada, e no momento em que o controlador, que voltou da falha, recebe a mensagem de término do processo.

Na medição da latência dos pacotes nos controladores, o tráfego foi capturado por meio da ferramenta *tshark*, e posteriormente analisada com o *wireshark*[6], de modo que cada *packet-in* tivesse o tempo medido de acordo com o seu *packet-out*. Foram coletados pacotes antes e após as simulações de falha.

Finalmente, para medir o percentual de uso de CPU das RouterBoards, a ferramenta de monitoramento *mpstat*[7] foi utilizada.

5. Discussão dos Resultados

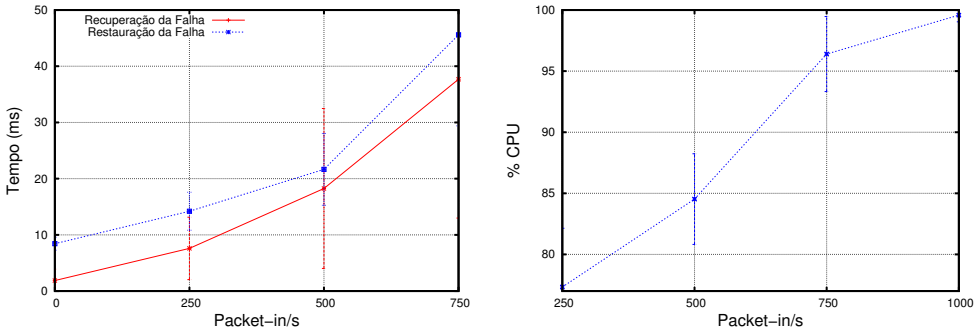
Nesta seção discute-se os resultados obtidos, organizados na seguinte sequência: primeiro, são apresentados os resultados dos testes com foco no plano de controle; em seguida, os números do plano de dados; e por fim, uma análise dos resultados.

Os valores de 1.000 *packet-in/s* e 720 *Mbps* representam o limite prático para o protótipo, isto é, não recomenda-se a utilização dos equipamentos com essa carga de

[5]https://github.com/esnet/iperf

[6]https://www.wireshark.org/

[7]http://www.linuxcommand.org/man_pages/mpstat1.html



(a) Recuperação e Restauração da Falha (b) Carga de Processamento nas RouterBoards

Figura 4. Tráfego no Plano de Controle

trabalho. Assim, de modo a melhorar a visualização dos gráficos, a latência das operações de Recuperação e Restauração da Falha, com taxas de 1.000 *packet-in/s* e 720 *Mbps*, não são apresentadas nos gráficos, apenas no texto.

5.1. Desempenho do plano de controle

Nas medições realizadas para o processo de recuperação de falha, ilustrados na Figura 4(a), o primeiro resultado é referente ao sistema sem carga, isto é, quando não há tráfego no plano de controle. Nesse estado, a latência do processo de recuperação de falha foi de 1,87 ms; no início do tráfego de pacotes, com taxa de 250 *packet-in/s*, o tempo atingiu 7,59 ms; com envio de 500 *packet-in/s*, chegou a 18,25 ms; a 750 *packet-in/s*, subiu para 37,68; e por fim, com taxa de 1.000 *packet-in/s*, alcançou 227,22 ms.

No processo de restauração de falha, apresentado na Figura 4(a), a mesma sequência do experimento anterior foi seguida. A latência da restauração de falha, sem tráfego foi de 8.42 ms; com tráfego de 250 *packet-in/s*, subiu para 14,19 ms; com taxas de 500 e 750 *packet-in/s*, aumentou para 21,65 e 45,84 ms respectivamente; já com a taxa de 1.000 *packet-in/s*, atingiu quase 5 s (em média), e com alta variabilidade.

Os resultados para a carga de processamento das RouterBoards, apresentados na Figura 4(b), de acordo com as taxas de geração de pacotes, variaram entre 77,33% e 99%. Com o envio a 250 *packet-in/s*, obteve-se o menor percentual de uso de processador, 77,33%; ao subir a taxa para 500 *packet-in/s*, o uso foi a 84,53%; no terceiro experimento, a taxa de 750 *packet-in/s*, chegou a 96,4%; e por fim, quando a taxa de envio era 1.000

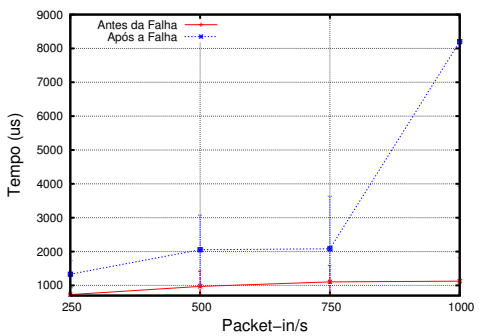
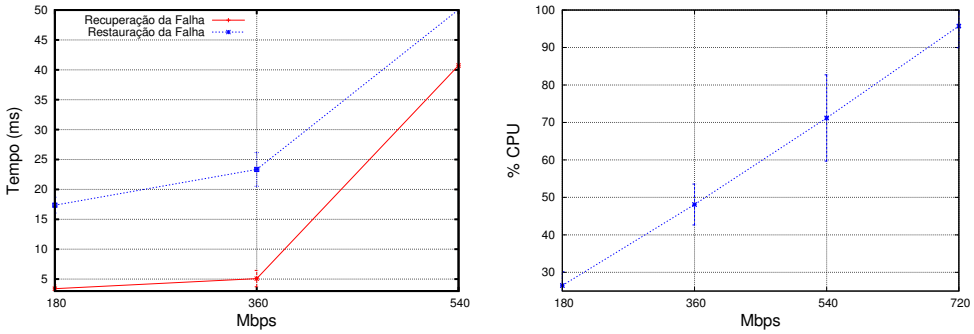


Figura 5. Tráfego no Plano de Controle: Latência de Packet-in.



(a) Recuperação e Restauração da Falha (b) Carga de Processamento nas RouterBoards

Figura 6. Tráfego no Plano de Dados

packet-in/s, alcançou 99% de uso, número próximo a ocupação total do processador.

Em nossos experimentos também avaliamos a latência do processamento de pacotes pelos controladores. Os valores para a latência do processamentos de *packet-in* são apresentados na Figura 5.1. Essa medida varia de acordo com os eventos de falha no plano de controle. Por exemplo, em um dos cenários apresentados, temos uma taxa de 250 *packet-in/s*. Ao ocorrer uma falha, o controlador que continua ativo, passa a receber todo o tráfego que tinha como destino o *faulty-controller*, ou seja, 500 *packet-in/s*. Com início dos testes, a latência de processamento variou entre 729 μs e 1.126 μs, e após a falha de um controlador, entre 1.332 μs a 8.200 μs.

5.2. Desempenho do plano de dados

Após os resultados dos experimentos com tráfego no plano de controle apresentamos os dados dos experimentos com foco na variação de carga no plano de dados.

A sequência dos resultados segue a mesma ordem dos experimentos apresentados anteriormente, e são mostrados na Figura 6(a). O primeiro resultado refere-se a recuperação de falha. Repetindo a abordagem em que o tráfego da rede aumenta de forma gradual, a latência para a recuperação de falha, neste caso, variou entre 3,39 ms e 398,2 ms. No primeiro cenário, com tráfego de 180 Mbps, o tempo para recuperação de falha, foi 3,39 ms; com o aumento da taxa para 360 Mbps, a latência subiu para 5,08 ms; a 540 Mbps, alcançou 40,73 ms; e por fim, com a taxa de 720 Mbps, a máxima foi 398,2 ms.

No processo de restauração de falha, os resultados apresentados estão ilustrados na Figura 6(a): com taxa de 180 Mbps, a latência ficou em 17,37 ms; a 360 Mbps, subiu para 23,34ms; com o aumento do tráfego para 540 Mbps, alcançou 50,06 ms; e ao final, com taxa de 720 Mbps, chegou a 119,28ms.

Na última série de experimentos realizados, exibimos os dados sobre a utilização de CPU das RouterBoards, Figura 6(b). Para os cenários apresentados, o percentual de utilização variou entre 26,48% e 95.73% e seguiu um comportamento linear.

5.3. Comentários finais

Algumas considerações podem ser feitas em relação aos resultados dos experimentos. No geral, o processo de restauração de falha tem maior latência que o de recuperação, e isso deve-se ao fato de que a quantidade de mensagens trocadas no processo de migração de

switches é maior. E mais, algumas dessas mensagens, como a *barrier-message*, indicam que, uma vez recebida, o *switch* deve processar todas as mensagens pendentes antes de qualquer outra mensagem que tenha chegado após a *barrier-request*, o que pode aumentar consideravelmente a latência.

Os *switches*, nos cenários com 1.000 *packet-in/s* e 720 Mpbs, operaram no limite de sua capacidade, o que causou grande variação nos resultados. Isso pode ser observado na variabilidade dos resultados encontrados. Ainda sobre o desempenho das RouterBoards, os resultados apontam que a CPU dos *switches* são mais sensíveis ao encaminhamento de *packet-in* para ao plano de controle do que ao encaminhamento de pacotes no plano de dados. Desde o primeiro cenário, com tráfego no plano de controle, os *switches* apresentaram alta carga de CPU, com 250 *packet-in/s* a taxa de utilização foi de 77,33%. Já nos experimentos com tráfego no plano de dados, a medida que a taxas variaram, o aumento do consumo de CPU se deu de maneira gradual.

Um outro importante levantamento diz respeito a latência de processamento de *packet-in*. Durante os experimentos, as latências aumentaram de forma gradual à medida em que a taxa de pacotes gerados variava, partindo de 729 e chegando a 1.126 μs. Após a falha de um controlador, para os cenários com taxa de 250, 500, 750 *packet-in/s*, o tempo de processamento para cada pacote, praticamente, dobrou; e para 1.000 *packet-in/s* a latência foi basicamente 8 vezes maior, atingindo 8200 μs. Essa análise reforça a importância do plano de controle ser escalável.

6. Trabalhos Relacionados

Há uma miríade de questões de pesquisa em recentes trabalhos relacionados à distribuição de controle em redes SDN objetivando soluções de alta disponibilidade. Nesta seção citamos apenas os trabalhos mais próximos ao foco da proposta deste artigo, que é a solução do problema de conectividade com múltiplos controladores e o uso de *data stores* para consistência do estado.

ONIX [Koponen et al. 2010] foi a proposta pioneira no controle distribuído em redes SDN particionando o escopo dos controladores, agregando as informações, e compartilhando o estado via APIs para diferentes tipos de *data stores* em função dos requisitos de consistência. O trabalho, porém, não discute a conectividade com múltiplos controladores nem o tratamento de falhas, que ficam sob a responsabilidade das aplicações.

Em [Fonseca et al. 2013] os autores apresentam um estudo em SDN em que as estratégias ativa e passiva são usadas para implementar controladores distribuídos. Além disso, indicam cenários em que utilização de cada técnica pode ser mais efetiva. No experimento que adota replicação passiva, os *switches* se conectam a dois controladores: um primário e outro secundário. O controlador primário é responsável por processar todos os pacotes da rede e manter uma cópia de seu estado no secundário. Na implementação utilizando replicação ativa, os *switches* se conectam a todos os controladores da rede, e estes, processam os pacotes de forma simultânea. Neste método, após receber e processar os pacotes, apenas um controlador responde cada requisição, e para tal, trocam mensagem de controle para definir o controlador que processou o pacote no menor tempo. Essa técnica permite que os estados da rede estejam replicados em cada controlador, já que todos controladores processam o mesmo pacote.

Apesar do estudo levantar pontos relevantes para a área, as duas implementações

apresentadas utilizam a versão 1.0 do protocolo OpenFlow, deixando a cargo dos projetistas de SDN implementar seus próprios mecanismos para que *switches* lidem com múltiplos controladores. Outra limitação da estratégia ativa implementada, segundo os autores, é que a sincronização dos controladores usando consenso distribuído aumenta consideravelmente a latência da rede e reduz a capacidade de processamento, já que adicionar controladores a arquitetura não aumenta o *throughput* do plano de controle.

Em [Botelho et al. 2014] apresenta-se uma proposta de uma arquitetura para múltiplos controladores chamada SmartLight. A estratégia é baseada em uma variação da abordagem passiva, em que há um controlador principal, que processa *packet-in*, e *N* controladores *backup*. Também, foi implementado um serviço de coordenação entre os controladores, de modo que em caso de falha no controlador principal, uma das réplicas possa assumir o controle da rede. Um dos pontos centrais do trabalho, é a utilização de um *data store* como meio de distribuir os estados da rede. Apenas o controlador principal interage com o *data store*, escrevendo e recuperando informações, e em caso de falha, o novo controlador principal deve recuperar os estados armazenados.

O trabalho apresenta uma linha promissora, no entanto uso de apenas um controlador principal limita a vazão do plano de controle. Um outro ponto levantado é que, em caso de falha, o controlador que assumir as operações da rede, precisa recuperar todos os estados no *data store*, o que adiciona uma maior latência na transição dos controladores.

7. Conclusão

A implementação de um sistema distribuído tolerante a falhas, tal como o plano de controle de redes SDN, envolve vários *trade-offs*. Neste trabalho, discutimos aqueles relacionados à implementação de controladores replicados ativa e passivamente.

Provavelmente o *trade-off* mais importante neste cenário é o conflito entre consistência forte nos estados das réplicas, o que permite aos diversos controladores atuar prontamente nos *switches*, e o custo em termos de comunicação para implementá-la. Neste trabalho, implementamos e avaliamos uma proposta de plano de controle distribuído usando replicação ativa, que provê consistência forte. Nossa avaliação, em ambiente real, utilizando elementos de rede de prateleira (COTS), o controlador *open-source* Ryu e o serviço OpenReplica, sugere que é viável o uso de consistência forte. Trabalhos relacionados da literatura ficaram limitados a ambientes de emulação/simulação e não focaram nas configurações de papéis no OpenFlow usando replicação via *data stores*.

Em relação aos resultados dos experimentos realizados, observamos o comportamento do ambiente, introduzindo falhas nos controladores da rede com variações de tráfego no plano de controle e no plano de dados. Nas operações de recuperação e restauração de falha, ficou caracterizado que a segunda, em geral, tem maior latência. Já sobre o desempenho dos *switches*, os resultados mostram que o encaminhamento de pacotes para o plano de controle, do ponto de vista da CPU, é mais custoso que o encaminhamento no plano de dados.

Para trabalhos futuros, pretendemos utilizar a capacidade de troca de controladores de *switches* para explorar o balanceamento de carga entre controladores. Na mesma linha, pretendemos habilitar os controladores a trabalhar com uma visão parcial da rede, compatível com a visão global, de forma a reduzir a carga em cada controlador.

Referências

Botelho, F. A., Bessani, A. N., Ramos, F. M. V., and Ferreira, P. (2014). Smartlight: A practical fault-tolerant SDN controller. *CoRR*, abs/1407.6062.

Botelho, F. A., Ramos, F. M. V., Kreutz, D., and Bessani, A. N. (2013). On the feasibility of a consistent and fault-tolerant data store for sdns. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pages 38–43. IEEE.

Budhiraja, N., Marzullo, K., Schneider, F. B., and Toueg, S. (1993). The primary-backup approach. *Distributed systems*, 2:199–216.

Défago, X., Schiper, A., and Urbán, P. (2004). Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys (CSUR)*, 36(4):372–421.

Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., and Kompella, R. (2013). Towards an elastic distributed sdn controller. *SIGCOMM Comput. Commun. Rev.*, 43(4):7–12.

Fonseca, P., Bennesby, R., Mota, E., and Passito, A. (2013). Resilience of sdns based on active and passive replication mechanisms. In *Global Communications Conference (GLOBECOM), 2013 IEEE*, pages 2188–2193.

Heller, B., Sherwood, R., and McKeown, N. (2012). The controller placement problem. In *HotSDN '12*, pages 7–12, New York, NY, USA. ACM.

Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. (2010). Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX ATC*, volume 8, page 9.

Koponen et al. (2010). Onix: A distributed control platform for large-scale production networks. *OSDI, Oct*.

Kreutz, D., Ramos, F. M. V., Veríssimo, P., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):63.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565.

Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169.

Liberato, A. B., Mafioletti, D. R., Spalla, E. S., Martinello, M., and Villaca, R. S. (2014). Avaliação de desempenho de plataformas para validação de redes definidas por software. *Wperfomance - XIII Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, pages 1905–1918.

Panda, A., Scott, C., Ghodsi, A., Koponen, T., and Shenker, S. (2013). Cap for networks. In *HotSDN '13*, pages 91–96, New York, NY, USA. ACM.

Penna, M. C., Jamhour, E., and Miguel, M. L. (2014). A Clustered SDN Architecture for Large Scale WSON. In *AINA*, pages 374–381. IEEE.

Rothenberg, C. E., Nascimento, M. R., Salvador, M. R., and Magalhães, M. F. (2010). OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. *Cad. CPqD Tecnologia, Campinas*, 7(1):65–76.

Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319.